Project 1 - FYS-STK4155

Bjørn Magnus Hoddevik and Hilde Langengen Teigen
(Dated: October 12, 2022)

A regression model aims to build a mathematical representation of an underlying process between a set of independent and dependent variables. The model can then be used to predict the outcome of new independent variables input. Ideally, these predictions should be aligned with real observation, with only some variation. For a good model, this variation should be small and unbiased, such that it can be explained by random variation. Supervised machine learning has proved effective for developing useful regression models, as they are able to dynamically reduce the model error given varying datasets. However, there are a range of methods for linear regression, and for complex and large datasets it can be difficult to determine which method that will give the best predictions. In this project, we evaluated three regression methods – standard OLS, Ridge, and Lasso. We use the two-dimensional Franke function for fitting models, and evaluating their quality through the Mean Squared Error (MSE) and R squared (R2) estimators of prediction variance and bias. We explored two ways of sampling the data, with bootstrapping and k-fold cross validation. We attempted different ways of scaling the data, and evaluated the results. Lastly, we used real data for testing the same methods of model fitting and evaluation. In this attempt, the code did not function well enough for a proper real-data implementation. However, an important ground work have given us tools to adapt such methods in the future.

I. INTRODUCTION

To understand the behaviour of a system, we need to implement models. Supervised machine learning has proved an effective way of fitting good models to observed data, making dependable predictions. However, ensuring an unbiased and precise model is not only crucial, but also difficult, and requires evaluation of the model fit. To reduce the bias of the model, machine learning algorithms usually divide the data set of which it aims to derive parameters into a training set and a testing set. This training set will contain information about the independent variables of the system as well as the response variables. A model is derived which constitutes the parameters that is estimated to determine the interaction between these variables. Exploring differences in testing and training data predictions can reveal over-fitting or under-fitting to the training set – meaning that too much or too little of the variation in the data is implemented into the model.

Evaluating the reliability of the model, will first involve testing the model on the testing subset of the data, that should be thus far unseen. The predictions of the model is compared to the dependent variable data given in this dataset. This comparison will help unravel the quality of the model, and involves evaluating statistical properties like the mean squared error, variance, and bias.

Here, three methods of linear regression were studied: the Ordinary Leasts Squares (OSL) method, the Ridge method, and the Least Absolute Shrinkage and Selection Operator (Lasso) method. These were evaluated considering Mean Squared Error (MSE) as a main performance metric, along with resampling: k-fold cross validation and bootstrapping. The bootstrapping method was in some particular focus as it is a greatly applicable resampling method. It is is based on the principles of the central limit theorem, while the cross-validation method redistibutes the dataset used as training data and test data to check generalizability. Pre-processing of the data was tested in form of different methods of scaling.

Two different data sets were used for model fit generation and comparison. The Franke function was first used for generating a predictable but complex data set, ideal for initial model evaluation. Later, the same methods for data fitting and evaluation were used on real terrain data with the same dimensions. The applicability of the above-mentioned regression methods to the data is discussed. All code can be found here: https://github.com/HildeLa/Project1_FYS-STK.git

II. PRELIMINARIES

A. Model fitting

The two dimensional Franke function was used with $x, y \in [0, 1]$ to make our dataset, with some added noise $\epsilon\ N(0, \sigma^2)$. Using these datapoints, we can approximate a model using machine-learning. When we later apply our models to real data, we also use this underlying assumption of the real data having a similiar type of noise.

Fitting our model is an optimisation problem we need a cost function. We used the used the MSE to have:

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

Where we define $\tilde{z}$ as:

$$\tilde{z} = X\beta \tag{2}$$

Where $X \in \mathbb{R}^{nxp}$, where p is the amount of features and n the number of datapoints. By taking the derivative of this function with regards to β and solving it for when

$\frac{\partial C(\beta)}{\partial \beta} = 0$ we get an expression:

$$\hat{\beta} = (X^T X)^{-1} X^T \vec{y} \tag{3}$$

Our model $\tilde{z} = X\hat{\beta}$ should then be the optimal model given our cost function.

B. Model evaluation

To evaluate our models, we used two commonly used measures – the mean squared error (MSE) and the R2 score, defined in equation 4 and equation 5 respectively.

$$MSE(\vec{z}, \tilde{z}) = \frac{1}{n} \sum_{i=0}^{n-1} (z_i - \tilde{z}_i)^2 \tag{4}$$

$$R^2(\vec{z}, \tilde{z}) = 1 - \frac{\sum_{i=0}^{n-1} (z_i - \tilde{z}_i)^2}{\sum_{i=0}^{n-1} (z_i = \bar{z})^2}, \quad \bar{z} = \frac{1}{n} \sum_{i=0}^{n-1} z_i \tag{5}$$

R2 is context independent, and simply compares the regression line to a baseline model given simply by the mean, telling us how much better our model performs. a perfect model fit would take the value 1, indicating that the regression line in always perfect so that the divisor equals 0. We are therefore aiming for high R2 scores, that indicates that a large part of the variation in the data can be explained by our regression.

C. Franke function

The Franke function is used here to generate a dataset to fit the regressions. It is given by two independent varaibles x and y, which are defined as $x, y \in [0, 1]$. The dependent variable z is given by the function:

$$z = f(x, y) + \epsilon, \quad \epsilon\ N(0, \sigma^2) \tag{6}$$

And $f(x, y)$ is the Franke function given by:

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

D. Scaling

It is a common procedure to scale the data as a preprocessing step when fitting a model. We tested four scaling methods, but we have generally been using the non-scaled data. Scaling is generally necessary when there is much variance, in particular for machine learning models in order to make the spread of values smaller and easier to provide weights to. However, the scaling must be balances in a way that represents the actual data. We have here used the Sklearn (Pedregosa et al. 2011) [1] preprocessing library, and applied the standard scaler, the minmax scaler, and the robust scaler, as well as performing a mean scaling. Th e standard scaler sets the mean value to zero and the variance to one for each design matrix feature. The minmax scaler sets all values in the feature matrix to be between 0 and 1. The robust scaler ignores outlier datapoints, but is otherwise similar to the standard. The mean scaler is simply subtracting the mean value from the features.

III. DERIVING A MODEL EXPRESSION

IV. ORDINARY LEAST SQUARES (OLS) ON THE FRANKE FUNCTION

The ordinary least squares (OLS) method for linear regression aims to minimize the sum og the squared differences between the training data and the model. The above section and the appendix describes the definition of the OLS expression for the optimization of parameters, given by a design matrix. We here tested evaluated this regression method on the Franke function, comparing optimization with different scaling methods and polynomial degrees. In both cases, there were not much room for improvement, likely due to us allowing little variation in the dataset that could othewise be a reason for adaption.

A. Scaling

Preprocessing of the data to be analyzed is often beneficial for facilitate model adaption to more easily interpretable numbers with lower variance. In a dataset with high variance and a lot of outliers, it is particularly important to select the correct scaling method. We tested our OLS on the four different scaling methods discussed above. Table I shows the results. In general, the scaling methods gained us similar or worse MSE and R2 scores. Interestingly, the mean subtraction method is the only method that may seem to enhance the model fit. How-

Scaler	MSE test	R2 test
Standard	0.048000	0.953586
Mean	0.012706	0.929348
MinMax	0.004890	0.929096
Robustscaler	0.019576	0.937013
None	0.010280	0.944845

Table I. Shows the estimators of model prediction ability MSE and R2 for the data preprocessed with four different scaling methods, and without scaling.

ever, the effect is small, and we can conclude that these data is not in need of scaling.

B. Degree

To optimize our model to the data, we generated the OLS for polynomial degrees 1 to 10, and compared the resulting variance and bias estimates. A too low polynomial degree may lead to the regression missing much of the variation in the data – underfitting. However, too high polynomial degree may lead to overfitting of the data to random variation in the training data. Hence, we calculated the MSE and R2 measures of both the training data and the test data. One can expect that the MSE and R2 will be better the more degrees we use to fit the model. This is because more variables allow us to fit the model to more of the variation in the training data set. Some of this variation may however be random, and specific for only the training data, meaning that it will lead to false predictions in the test data. It is therefore useful to compare these predictions. If the predictions made from the training data are better than that for the test data, there is reason to reduce polynomial degree.

We used the Scikit-Learn OLS model as a means to validate our results, as shown in table 2. Our results are slightly higher considering the MSE score than Scikit-Learn, but the values are within a similar range. The training data predictions and the test data predictions are generally similar (table 2, fig. 1), and the differences depend much on each run of the data. We expected to see an increase in the test data and sci-kit learn predictions with higher values, but we cannot see a clear trend of this. In this case, we get a somewhat better MSE with both Scikit-Learn and the test data at 7 degrees, and at 9 degrees for the train data. This difference could be due to some overfitting to the training data that is not detected in the MSE for the training data.

Degree	MSE train	MSE test	MSE sklearn
1	0.041884	0.053196	0.043049
2	0.043687	0.047005	0.045379
3	0.028635	0.030708	0.028144
4	0.015321	0.018876	0.016824
5	0.016745	0.013818	0.017409
6	0.013561	0.014745	0.016833
7	0.011894	0.010073	0.011087
8	0.010897	0.013083	0.012664
9	0.011221	0.011535	0.013664
10	0.011563	0.010437	0.011323

Table II. MSE scores of test-data and training data for polynomial degrees 1 to 10. In addition, the scores for the Scikit-Learn package version of the OLS is shown for validation of our methods.

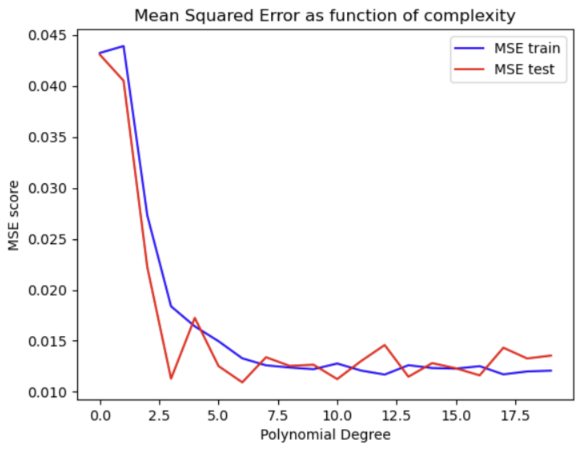


Figure 1. MSE scores of test and training data as a function of polynomial degree in the fitted OLS, with a dataset of 500 datapoints.

C. Bias-Variance tradeoff and resampling techniques

It is essential to have the right complexity level of the function to avoid over – or under-fitting, as discussed above. This is essentially due to a trade-off between the bias and the variance in the model, giving a means to study an increase in error. This bias-variance trade-off derives from the bias being a measure of the deviation of the expected estimator parameters from the true values, while the variance is a measure of variation in the resulting model estimator. This means that increasing the bias of the model parameters, will reduce the overall variation, and vice versa[2]. The total error of the model can be determined by summing up the bias and the variance. The bias of the model will decrease when the complexity is increased, while the variance increases.

D. Bootstrap resampling method

Bootstrapping is based on the central limit theorem, and involves picking random samples of the dataset with replacement of the same size as the actual dataset, and calculate some statistic on this set. This is done for a large number of iterations, often 500 or 1000. The mean and standard deviation of these sample statistics can then be calculated, assumed to give a determination of the real population, as well as an error estimate. The central limit theorem constitutes the main underlying idea of this method, as a large enough sample population will on average share similar distributions with the real population. This way, bootstrapping methods are rigid against outliers and random variation in the dataset, giving us an expected prediction error (Hastie et al. 2001 p. 249-257) [3].

Here, we used a bootstrapping method with 500 iterations to study the bias-variance trade-off in our models. Figure 2 shows us that the bias and error both decreases around four degrees and stays low for all higher degrees. We see little change in the variance, which stays low possibly due to little variance in the data. We could have expected the variance, and therefore also the error to increase as the complexity increased, which could be ascribed to overfitting. However, we are seeing the same lack of error increase with the test data compared with the training data in figure 1. We can therefore assume there to be some unfound issue with our code, or simply an effect of unexplored addition of variance. Increasing the variance in our generated data, would give the model random variation to wich it may overfit.

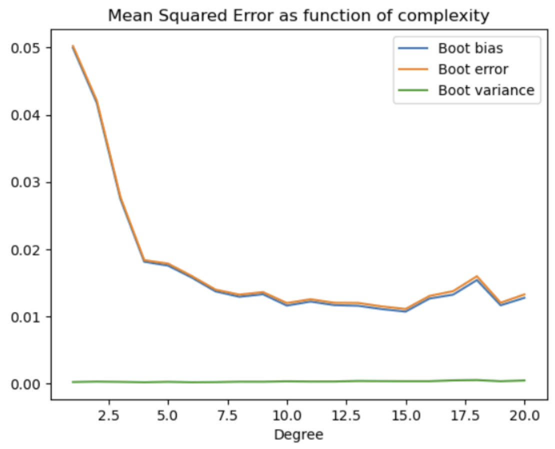


Figure 2. Estimates of error, bias, and variance based on bootstrapping of a linear OLS regression, plotted as a function of model polynomial degree.

E. Cross-validation for resampling, adding more complexity

k-fold cross validation is alongside bootstrapping a commonly resampling technique to validate the reliability of the developed model function for fitting parameters. The data is shuffled and divided into k equal parts. k-1 of the parts are then used to train the data for model fitting. Meanwhile, the last part is not used until the prediction test of the model, from which an evaluation of the model, like the MSE score, is kept. This is repeated k times, such that all k samples are used as the training set once. The total cross validation estimate is then given as the mean MSE of all k validation, as shown below.

$$MSE_{CV} = \frac{1}{k}\sum_{i=0}^{k-1} MSE_i \tag{8}$$

The value of k depends on the data size, where setting k = n can often be useful with small datasets, as it ensures that all datapoints can be used for testing. If the datasets are larger, a small k can give low variance, but may give a higher bias. Meanwhile, a higher k may have the opposite effect. Here, we have made a cross validation test of our regressions using k = 5 and k = 10. k is commonly set to 5 or 10 for sufficiently large datasets, as described by Hastie et al. (2001, p. 241-245)[4] The results can be seen in figure 3. Again is the MSE scores are similar

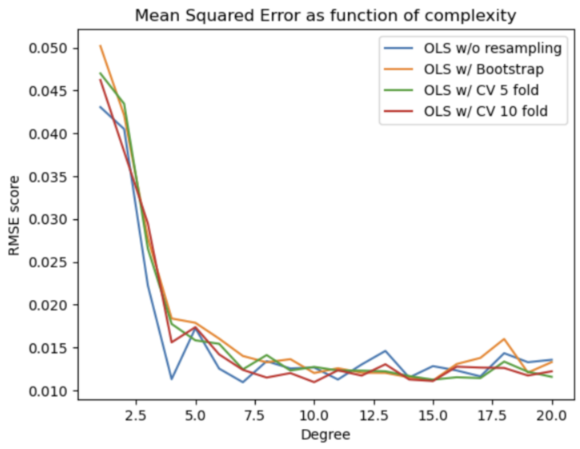


Figure 3. MSE as a function of model complexity for the OLS regression using 10-fold and 5-fold cross validation and bootstrapping, as well as non-resampled.

to the other estimates, with a clear decrease around four degrees. The difference to the other estimates appears to be of no significance, although there could be an indication that there is a slight rise on the bootstrapped and non-resampled MSE, that is visible when compared to the cross-validation versions. This could be random fluctuations, or a slight over-fitting by the bootstrapping, . For 5-fold validation the MSE standard deviation was reduced from 0.005 to 0.001 with increased polynomial degree, and for the 10-fold it was reduced from 0.008 to 0.003, indicating that the variation in the data was better explained for higher polynomial degrees. The bootstrap could in this case be overfitting to the added random fluctuations created by the resampling with replacement step, while the k-fold cross-validation is only reshuffling this low-variation dataset.

V. RIDGE REGRESSION AND RESAMPLING

The Ridge method is an alternative way of performing a polynomial OLS regression, that uses much of the same underlying assumption, except that the error does not need to be normally distributed. Here, λ is added to the equation as a penalty term – a regularization parameter accounting for additional variance. Small contribution variables will have variables reduced depending on the size of λ. At zero, the model will be just a basic OLS, while if it is increased the coefficients will be penalized. This will also lead to a reduction of the model complexity, meaning that a too high λ could lead to underfitting. But ridge regression is also subject to the bias-variance trade-off. As λ increase, the variance will decrease, however, the bias will increase. It is therefore important to fit the right λ. Tuning an appropriate λ parameter will enhance the stability of the model, through its penalizing of larger values.

Here, we have performed the same analysis as above, but with the added ridge regression λ parameter. To determine the best λ value, we tested 5 different λs: 0, 0.0001, 0.01, 1, and 10. Zero would in this case be the same as fitting a normal OLS as before, and functions for comparing the effect of λ, see fig 4. If there was more variation in the data, we would expect a rise in the MSE value for higher complexity models when $\lambda = 0$, that the ridge regressions would be resistant to. However, we do see that a high λ value in this case leads to an increase in prediction error for all degrees.

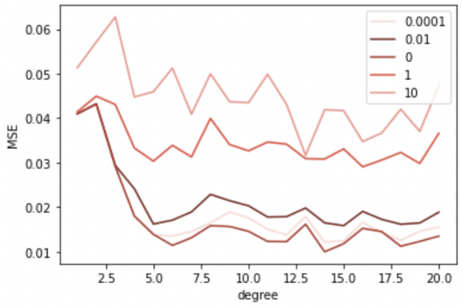


Figure 4. Five different λ values in ridge regression

To study the effect of λ on the out-of-sample error, we used polynomial degree 10, and fitted bootstrapped predictions for a range of λ values between 0.0001 and 10. Figure 5 shows the resulting MSE score as a function of λ value. We can see that in this case, any increase of λ value has a negative effect on the model, implicating that a ridge regression is not appropriate in this case. Figure 6 shows a comparison for all five λs as above with the other model fits used in this report, and we see that the ridge regression performs worse in all cases, except when the λ is set to 0, which leaves it no effect.

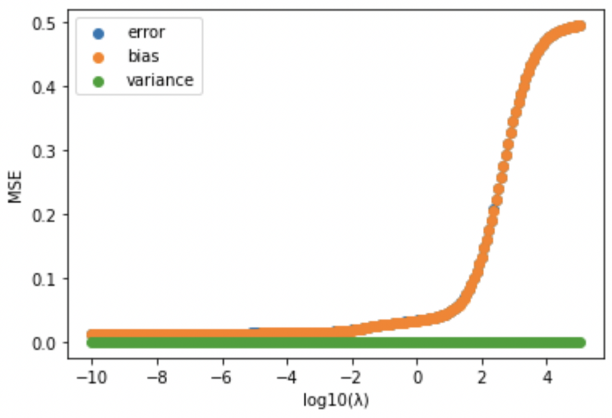


Figure 5. MSE as a function of λ values for ridge regression of dataset with polynomial degree set to 10.

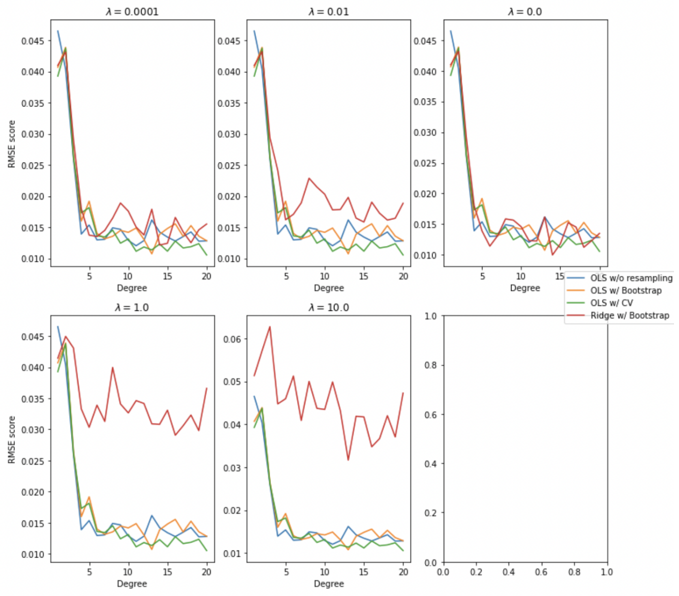


Figure 6. Regression comparisons with diffrent λ values, with MSE as an estimate of error, and as a funciton of polynomial degree.

VI. LASSO REGRESSION AND RESAMPLING

Similar to ridge regression, Least Absolute Shrinkage and Selection Opera- tor (LASSO) regression also works by introducing a regularisation parameter λ. This model uses shrinkage towards a central point to overcome over-fitting. The Lasso regression will reduce some parameters to zero, whereas the ridge regression will only reduce them to a value close to zero. While Lasso regression uses the L1 regularisation method as opposed to L2 as the ridge regression – which results in the removal of parameters, and hence a rection in model complexity. This leaves Lasso regression effective against overfitting, and useful for simplifying models.

We used a mean scaler and bootstrapping on the Sci-Kit Learn version of Lasso regression. Figure 7 compares this model for different λ values to other models explored here, and we find that the regression is better for all λs than the ridge regression. Unfortunately, due to time restrictions, we did not get the code for this regression analysis to work, but the attempt can be seen in the attached jubyterhub script.

VII. REAL DATA ANALYSIS

Lastly, we wanted to adapt the models that we have studied for real data. We used the given topological data from Norwegian terrain [5], and selected an area of interest on which we could analyse the effect of area on altitude. An illustration of the data subset of 100 x 100 that we picked is shown in figure 8.

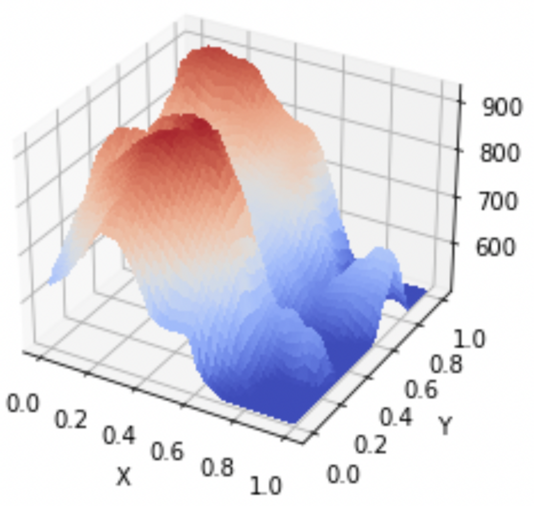
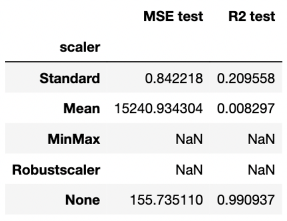


Figure 7. A plot of the subset of terrain data used for analysis

As the dataset differs from that of the Franke function in variability, we assumed that it may need scaling and a high degree of complexity for the model fitting. We attemped a 20 degree polynimial simple regression with different scaling methods, however, as seen in table 3, we had difficulties with obtaining any fits that made sense. This would require more time to develop. For further analysis we went with the standard scaler. Although we are not certain that our fit here is working, at least we could get the scaling function to work. We can assume that scaling will be useful for natural data, as they contain much variation on larger and smaller scales.



	MSE test	R2 test
scaler		
Standard	0.842218	0.209558
Mean	15240.934304	0.008297
MinMax	NaN	NaN
Robustscaler	NaN	NaN
None	155.735110	0.990937

Figure 8. Attempt to fit a 20 degree polynomial simple OLS regression to topoligical data

To evaluate the complexity needed to fit the data, we attempted 1 to 20 polynomial degrees for our regression model, and compared the R2 and MSE scores for the training and test data, shown in figure 9. The results underscores our view that there is something in our code that causes the functions not to work properly for the real dataset, as the results are not meaningful. There is more variation in the test data prediction error, which would make sense as there is a higher degree of randomness in going from one data subset to another. Strangely, it almost looks like the R2 score was higher for low complexity, which is difficult to explain without assuming some essential flaw in the code. For this reason, the further analysis was halted, and will need to be resumed in the future.

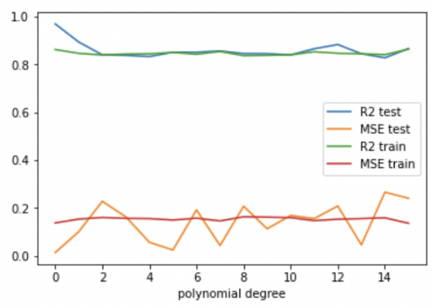


Figure 9. MSE and R2 as funciton of model complexity for training set and test set of the data

Appendix A: Part a, Initial OLS expression inferences

We can write

$$\vec{y} = f(\vec{x}) + \epsilon$$

Where $\epsilon N \sim (0, \sigma^2)$. Where any element $y_i \in \vec{y}$ can be written as:

$$y_I = f(x_i) + \epsilon$$

Then, the expectation value can be written as:

$$\mathbb{E}[y_i] = \mathbb{E}[f(x_i) + \epsilon] = \mathbb{E}[f(x_i)] + \mathbb{E}[\epsilon]$$

By definition $\mathbb{E}[\epsilon] = 0$. Since we do not know $f(x_i)$ we use our model instead $f(x_i) = X_{ij} * \beta$:

$$\mathbb{E}[f(x_i)] = \mathbb{E}[X_{ij} * \beta]$$

However, $X_{ij} * \beta$ is non-stochastic meaning $\mathbb{E}[X_{ij} * \beta] = X_{ij} * \beta$. Giving us finally:

$$\mathbb{E}[y_i] = X_{ij} * \beta$$

Next we look at the variance:

$$\begin{aligned} var[y_i] &= \mathbb{E}[(y_i - \mathbb{E}[y_i])^2] \\ var[y_i] &= \mathbb{E}[y_i^2 - 2y_i\mathbb{E}[y_i] + (\mathbb{E}[y_i])^2] \\ var[y_i] &= \mathbb{E}[y_i^2] - 2\mathbb{E}[y_i\mathbb{E}[y_i]] + (\mathbb{E}[y_i])^2 \end{aligned}$$

Using the tricks from earlier we calculate each part of the above equation:

$$\begin{aligned} \mathbb{E}[y_i^2] &= (X_{ij} * \beta)^2 + 2\mathbb{E}[\epsilon]X_{ij} * \beta + \mathbb{E}[\epsilon^2] \\ \mathbb{E}[y_i\mathbb{E}[y_i]] &= (X_{ij} * \beta)^2 + \mathbb{E}[\epsilon]X_{ij} * \beta \\ (\mathbb{E}[y_i])^2 &= (X_{ij} * \beta)^2 \end{aligned}$$

Inserting these back into the equation we get:

$$\begin{aligned} &var[y_i] = \mathbb{E}[y_i^2] - 2\mathbb{E}[y_i\mathbb{E}[y_i]] + (\mathbb{E}[y_i])^2 \\ &var[y_i] = (X_{ij} * \beta)^2 + 2\mathbb{E}[\epsilon]X_{ij} * \beta + \mathbb{E}[\epsilon^2] - 2((X_{ij} * \beta)^2 + \mathbb{E}[\epsilon]X_{ij} * \beta) + (X_{ij} * \beta)^2 \end{aligned}$$

The $(X_{ij} * \beta)^2$ cancel eachother and $\mathbb{E}[\epsilon] = 0$ so those disappear. Last we know by definition that $\mathbb{E}[\epsilon^2] = \sigma^2$. Using all this we get finally:

$$var[y_i] = \sigma^2$$

Third we look at $\mathbb{E}[\hat{\beta}]$:

$$\mathbb{E}[\hat{\beta}] = \mathbb{E}[(X^TX)^{-1}X^T\vec{y}]$$

Since $(X^TX)^{-1}$ is stochastic we can move it outside and use what we previously found tat $\mathbb{E}[\vec{y}] = X\beta$:

$$\mathbb{E}[\hat{\beta}] = (X^TX)^{-1}X^TX\beta = \beta$$

Last we want to calculate the variance $var[\hat{\beta}]$:

$$var[\hat{\beta}] = var[(X^TX)^{-1}X^T\vec{y}]$$

We can use the identity $var[X\vec{y}] = Xvar[\vec{y}]X^T$ where X is a non-stochastic matrix:

$$var[\hat{\beta}] = (X^TX)^{-1}X^Tvar[\vec{y}]((X^TX)^-1X^T)^T$$

Since $(X^TX)^-1$ is a symmetrical matrix we have that $((X^TX)^-1X^T)^T = X(X^TX)^{-1}$. Using this and that $var[\vec{y}] = \sigma^2$ which we calculated earlier we finally get:

$$var[\hat{\beta}] = \sigma 2(X^TX)^{-1}X^TX(X^TX)^{-1} = \sigma^2(X^TX)^-1$$

Appendix B: Part c

We want to be able to express $\mathbb{E}[(\vec{y}-\tilde{y})^2]$ in terms of variance $var[\tilde{y}]$ and bias $(Bias[\tilde{y}])^2$ of our model and the variance of our data σ^2. We start with the equation:

$$\mathbb{E}[(\vec{y}-\tilde{y})^2] = \mathbb{E}[(f+\epsilon-\tilde{y})^2]$$

Since we can add 0 at any point we do this by $0 = \mathbb{E}[\tilde{y}] - \mathbb{E}[\tilde{y}]$:

$$\mathbb{E}[(\vec{y}-\tilde{y})^2] = \mathbb{E}[(f+\epsilon-\tilde{y}+\mathbb{E}[\tilde{y}]-\mathbb{E}[\tilde{y}])^2]$$

Next we want to expand this equation, grouping expressions so that we can easily identify squares that we can use to simplify again:

$$\mathbb{E}[(\vec{y}-\tilde{y})^2] = \mathbb{E}[(f^2 - 2f\mathbb{E}[\tilde{y}] + (\mathbb{E}[\tilde{y}])^2) + \epsilon^2 + ((\mathbb{E}[\tilde{y})^2 - 2\tilde{y}\mathbb{E}[\tilde{y}] + \tilde{y}^2) + 2\epsilon(f - \mathbb{E}[\tilde{y}]) + 2\epsilon(\mathbb{E}[\tilde{y} - \mathbb{E}[\tilde{y}]) + 2((\mathbb{E}[\tilde{y}] - \tilde{y})(f - \mathbb{E}[\tilde{y}]))]$$

We recoginize the square:

$$(f^2 - 2f\mathbb{E}[\tilde{y}] + (\mathbb{E}[\tilde{y}])^2) = (f - \mathbb{E}[\tilde{y}])^2$$

We can rewrite the last part of the equation:

$$2((\mathbb{E}[\tilde{y}]-\tilde{y})(f-\mathbb{E}[\tilde{y}])) = 2((\mathbb{E}[\tilde{y}]-\tilde{y})(X\beta+\epsilon-X\beta)) = 2\epsilon((\mathbb{E}[\tilde{y}]-\tilde{y})$$

All parts of the equation containing ϵ where it isnt squared can then be removed:

$$\mathbb{E}[(\vec{y}-\tilde{y})^2] = \mathbb{E}[(f-\mathbb{E}[\tilde{y}])^2] + \omega^2 + \mathbb{E}[(\mathbb{E}[\tilde{y}] - \tilde{y}]$$

We recognize these ters as:

$$\mathbb{E}[(f-\mathbb{E}[\tilde{y}])^2] = (Bias[\tilde{y}])^2$$
$$\mathbb{E}[(\mathbb{E}[\tilde{y}] - \tilde{y}] = var[\tilde{y}]$$

Giving us finally:

$$\mathbb{E}[(\vec{y}-\tilde{y})^2] = (Bias[\tilde{y}])^2 + var[\tilde{y}] + \sigma^2$$

[1] Pedregosa et al., JMLR 12, pp. 2825-2830, 2011.

[2] Mehta, Pankaj, Marin Bukov, Ching-Hao Wang, Alexandre G.R Day, Clint Richardson, Charles K Fisher, and David J Schwab. ‘A High-bias, Low- variance Introduction to Machine Learning for Physi- cists.‘ Physics Reports 810 (2019): 1-124.

[3] Hastie, T., Tibshirani, R.,, Friedman, J. (2001). The Elements of Statistical Learning. New York, NY, USA: Springer New York Inc..

[4] Hastie, T., Tibshirani, R.,, Friedman, J. (2001). The Elements of Statistical Learning. New York, NY, USA: Springer New York Inc..

[5] Https://github.com/CompPhysics/MachineLearning/tree/master/doc