University of Oslo.

Project 1.

Regression analysis and resampling methods.

Martin Hansen Bolle, Ingar Andre Benonisen

FYS-STK-4155/Oblig1/Department of Informatics

07.10.2024

Abstract.

In this project, we explore various statistical methods for estimating the relationship between a dependent variable and one or more independent variables using regression analysis. We work with both the Franke function and real-world data separately to analyze how different models, including Ordinary Least Squares (OLS), Ridge, and Lasso, respond to variation in parameters and complexity. Our analysis reveals that Ridge regression consistently outperforms OLS in terms of Mean Square Error for higher polynomial degrees, while Lasso performs best when sparsity is prioritized. We evaluate the performance of these models using the Mean Square Error (MSE) and R-squared metrics (R2), examining their behavior across different polynomial degrees and lambda values. Additionally, our investigation of the bias-variance trade off shows that increasing complexity reduces bias at the cost of increased variance. We implemented cross-validation and bootstrapping as resampling techniques to assess model stability and generalization.

Content

Introduction.

The aim of this project is to study in detail various regression methods. The goal of the regression analysis is to exploit a relationship between **x** and **y** to infer specific dependencies. By estimating these relationships, we can make predictions and gain insight into the underlying processes governing the data. In this report we focus on three different distinct regression techniques; Ordinary least squares (OLS), Ridge, and Lasso regression, each of which has unique characteristics and applications in statistical modeling.

Regression analysis serves as an excellent entry point for understanding central machine learning concepts and features, providing a clear pedagogical connection to various methodologies. In today's data-driven world, the understanding of the relationship between variables is crucial for informed decision-making across multiple fields. By leveraging regression analysis, we can effectively introduce essential concepts such as validation, resampling, and regularization, making it an ideal foundational tool on the journey in the field of machine learning and data analysis.

In our project, we explore the Franke function, a synthetic benchmark function that is often employed in numerical analysis and regression testing due to its complex but well-understood structure. Alongside real-world data, the Franke function helps us investigate how regression models respond to variations in parameters and model complexity. Our primary focus is on the performance metrics Mean Square Error (MSE) and R-squared (R^2), reflecting model accuracy and the fit of the model. Understanding how these metrics change with the degree of complexity is crucial for identifying the optimal polynomial degree and avoiding underfitting or overfitting, as demonstrated by the bias-variance trade off. This is a key concept in predictive modeling that highlights the balance between a model's complexity and generalization.

Lastly, we utilize bootstrapping and cross-validation as resampling techniques to enhance our model evaluation. The structure of this report is as follows: In section 2, we will provide a detailed description of the methods we have used and the theory behind them. In section 3 we will provide an explanation of our implementation and the result we got. Lastly, we will give a brief conclusion and some perspectives on our result and work. We would like to note that all handwritten exercises are attached in our appendix.

Method

Franke Function.

In the first part the foundation of our analysis addresses the fitting of polynomials to a two-dimensional function, more specific, the Franke Function. This function is widely used when testing a variety of interpolation and fitting algorithms. Its complex shape introduces multiple peaks and valleys, making it a challenging test case for various numerical functions. It is a sum of four exponentials and reads as follows:

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

The function combines features like smooth regions, valleys and sharp peaks, features we want our model to capture, making it an excellent function for evaluation. For this project we have defined $x, y \in [0,1]$. One could argue that this scales our data to a particular domain for the input values.

Linear Regression

We have applied three different linear regression techniques, including Ordinary Least Squares (OLS), Ridge-, and Lasso regression, to model the Franke function. They all serve the same purpose, but with distinct approaches to handling model complexity and regularization. We aim to analyze and understand the relationship between one or more independent variables and a dependent variable by fitting a linear model to the data. Each regression technique is applied to the same Franke function which is discussed initially.

Ordinary Least Squares.

We have seen from the data generation that our data consists of a set of inputs $x = [x_0, x_1 x_2, \dots, x_{n-1}]$ and $y = [y_0, y_1 y_2, \dots, y_{n-1}]$. We assume that the output data can be represented by a continuous function f through

$$y_i = f(x_i) + \epsilon_i$$

Where ϵ represent some noise. For us to approximate **y** by using a linear model we express it in terms of the unknown parameter β. This gives us the equation for multiple linear regression

$$y = \beta_0 + \beta_1 X_1 + \beta_2 X_2 + \cdots + \beta_n X_n + \epsilon$$

Since we are dealing with multiple predictors, we express it in matrix form using the design matrix

$$\hat{y} = X\hat{\beta}$$

Where X is the design matrix, $\hat{y}$ is the predicted values, and $\hat{\beta}$ is the vector of coefficients we aim to estimate.

Our goal is to minimize the sum of the squared differences between the observed and the predicted values

$$Minimize \sum_{i=0}^{n-1} (y_i - \hat{y}_i)^2$$

With matrix notation we get the loss function

$$C(\beta) = (y - X\beta)^T (y - X\beta)$$

If, and only if XTX is invertible Taking the derivative with respect to β and if, and only if, $X^T X$ is invertible, we get the solution for our optimal β.

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

With this insight we can perform our Regression analysis on the Franke function using Ordinary least square and gain insight on our model performance by interpret the MSE and R^2 score.

Ridge.

We have only utilized the ordinary least squares (OLS) regression scheme so far, but we want to look at two shrinkage methods. More precisely Ridge and Lasso regression. We will start off by looking at Ridge before we proceed to lasso in the next sub chapter.

Ridge aims to find which variables are significant and which variables are not, and then removing the influence of the insignificant variables by setting them close to zero. This is done by adding the shrinkage parameter lambda to both sides of the OLS equation (Hastie et al., 2017)

With our insight from OLS, we can derive the equation for Ridge regression. Ridge regression is ordinary least squares regression with an L2 penalty term on the weights in the loss function. This is what we call the regularization term. This term penalizes large coefficients and helps prevent overfitting. From the loss function from OLS, we get

$$C(\beta) = (y - X\beta)^T (y - X\beta) + \lambda ||\beta||_2^2$$

Here, the λ is a hyperparameter that controls the strength of the penalty, and $||\beta||_2^2$ is the squared L2 norm of the coefficient vector β, which equals the sum of the squared coefficients

$$||\beta||_2^2 = \sum_{j=1}^{p} \beta_j^2$$

By minimizing the loss function above with respect to β we can obtain an analytical expression for the optimal parameter β

$$\hat{\beta}_{ridge} = (X^T X + \lambda I)^{-1} X^T y$$

Where **I** is the identity matrix and lambda are the penalty term.

Lasso

Lasso regression seeks to identify the variables and their corresponding coefficients that result in a model with minimal prediction error. It does this by applying a constraint on the model's parameters, which reduces the size of the regression coefficients towards zero. Specifically, it ensures that the sum of the absolute values of the coefficients remain below a predetermined threshold lambda (Ranstam & Cook, 2018).

This introduces us for a new interpretation of the known ordinary least square method. We now add a L1 penalty term, which help us with regularization and also enables variable selection by shrinkage. We have our cost/loss function from OLS which we want to minimize

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

By adding our L1 term we have our new optimization equation.

$$C(X, \beta) = (y - X\beta)^T (y - X\beta) + \lambda \, ||\beta||_1$$

As in Ridge regression, lambda is a tuning parameter controlling the strength of the penalty, and $||\beta||_1$ is the L1 norm of the coefficient defined by

$$||\beta||_1 = \sum_{j=1}^{p} |\beta_j|$$

By taking the derivative of the cost function with respect to β we get

$$\frac{\partial c(x, \beta)}{\partial \beta} = -\frac{2}{n} X^T (y - X\beta) + \lambda sgn(\beta) = 0$$

With some reordering we can rewrite our expression and end up with

$$X^T X\beta + \lambda sgn(\beta) + \frac{2}{n} X^T y$$

Since the L1 norm includes the absolute value of the coefficient, the Lasso objective function is not differentiable at zero, unlike Ordinary Least Squares and Ridge Regression. This non-differentiability prevents the derivation of a closed-form solution, necessitating the use of iterative optimization algorithms to find the optimal coefficients. This makes an implementation from scratch challenging. Fortunately, sklearn provides a Lasso-class that will we used.

Bootstrapping

Bootstrapping seeks to estimate the conditional error. The way it works is that we draw B bootstrap samples from our training data. These samples are created randomly by sampling

with replacement (Hastie et al., 2017). This just means that some data may appear multiple times in each of our samples, while other may be excluded.

Cross-validation

This resampling technique splits the dataset into k number of parts, uses one part for validation and the remaining for training. This process is repeated k number of times, switching the testing part each time. As with bootstrapping, the results are averaged for more reliable insights to the performance.

Performance metrics

The performance of each model is measured using Mean Squared Error (MSE) and the coefficient of determination, R-squared (R^2). These performance metrics provide useful insight on the accuracy of the regression model. The MSE correspond to the expected value of the squared error or loss, and is defined as

$$MSE(y, \hat{y}) = \frac{1}{n}\sum_{i=0}^{n-1}(y_i - \hat{y}_i)^2$$

Where $\hat{y}$ is the predicted value and y is the true value. This metric tells us about how far off the predicted value is the actual value. We want this metric to be as close to zero as possible.

The R^2 score on the other hand provides a measure of how well future samples are to be predicted by the model. The best possible score here is 1.0, and the score can be negative. With the same $\hat{y}$ as the predicted value the R^2 is defined as

$$R^2(y, \hat{y}) = 1 - \frac{\sum_{i=0}^{n-1} y_i - \hat{y}_i{}^2}{\sum_{i=0}^{n-1} y_i - \bar{y}_i{}^2}$$

Where we have the mean value $\bar{y}$ defined as

$$\bar{y} = \frac{1}{n}\sum_{i=0}^{n-1} y_i$$

Bias-variance trade off

The bias-variance trade-off is a fundamental concept in statistical learning that describes the balance between a model's ability to minimize error on training data and its performance on unseen data. Understanding this trade-off is crucial for developing models that generalizes well.

As mentioned, our goal is to minimize the expected error on new, unseen data. In this project we have used the Mean squared error. The MSE can be decomposed into three components. Bias, variance and irreducible error (noise). A mathematical background can be found in the appendix.

Scaling

We want our data to lie within reasonable boundaries and maybe most importantly, the same boundaries. Therefore, we have scaled our data. This is a crucial step in machine learning that involves transforming the features of a dataset to ensure they are on a similar scale. As we are using metrics and algorithms like MSE, that are sensitive to the magnitude of the input features this practice become particularly important.

We have chosen to implement the standard scaler from the scikitLearn.preprocessing library. This standardizes features by removing the mean and scaling to unit variance. The standard score of a sample x is calculated as

$$z = (x - u)/\, s$$

Where **u** is the mean of the training samples and **s** is the standard deviation. The same scaling has been applied for all tasks.

Results

In this section we intend to explain how we have implemented the method discussed in the previous section and how we have chosen to structure our algorithms and our code. We want to demonstrate some of the calculations we have done. This is to ensure that our code behave as intended and also reproduces previous results.

We also want to emphasize that the implementation of task A to F has used the same Franke Function, data generation, and splitting of our data.

Test case

For validation purposes we created a short script to test our implementation of our selected regression methods using a simple linear expression. The linear expression, with some added noise, provides a controlled environment where we know the expected outcomes. We conducted a test using an identity matrix to verify the correctness of our implementation.

The coefficient for Ordinary least square, Ridge, and Lasso were all found to be close to the true values.

By setting the design matrix to an identity matrix, we created a scenario where each input directly corresponds to its output. For Ridge and Lasso with alpha = 0, the resulting MSE should be 0 as well. Based on these tests, we conclude that our implementation was correct, and we could confidently proceed to apply these methods to the Franke function.

Exercise a) Ordinary Least Square

From the sklearn library we used PolynomialFeatures and LinearRegression to perform our Ordinary Least square analysis. With these we could fit our training data to our model up to the desired degree and perform our Linear Regression.

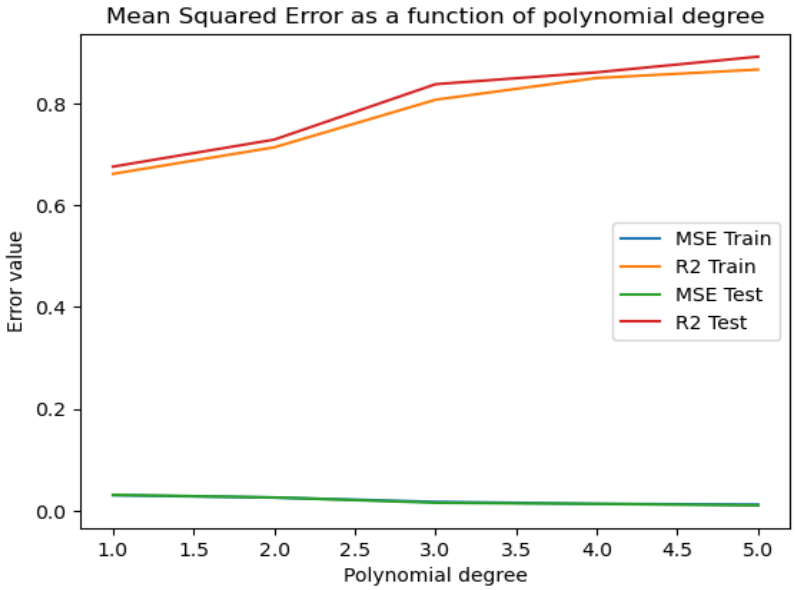


Figure 1 Mean Squared Error and R2 values for both training and test data as a function of polynomial degree. The plot illustrates how model complexity represented by the degree of polynomial, impacts the MSE and R2 scores. The current plot uses n=1000

Both the error metrics, the Mean Square Error (MSE) and R^2, demonstrate improvement as model complexity increases. As shown in figure 1 the MSE steadily decreases, approaching zero as the complexity of the model grows, which aligns with the goal of minimizing prediction error. A lower MSE indicates a better fit, as shown in figure 1. Additionally, increasing the model's polynomial degree enhances its flexibility, enabling it to capture more intricate patterns in the data and a better ability to learn.

Similarly, the R^2 score shows an upward trend towards one, which signifies a stronger explanatory power of the model. As with MSE, the increasing complexity of the model, driven by higher polynomial degrees, allows it to better capture the variability and complexity inherent in the data. This reflects the model's improved ability to generalize and adapt to more sophisticated relationships within the dataset.

This effect becomes evident when we decrease the number of observations by one order of magnitude. As seen in figure 2 below, the test-MSE displays higher values. The model has less information to learn from during the training process, leading to poorer generalization. It is safe to say that the increase in number of observations drastically increases the predictive capability of the model.

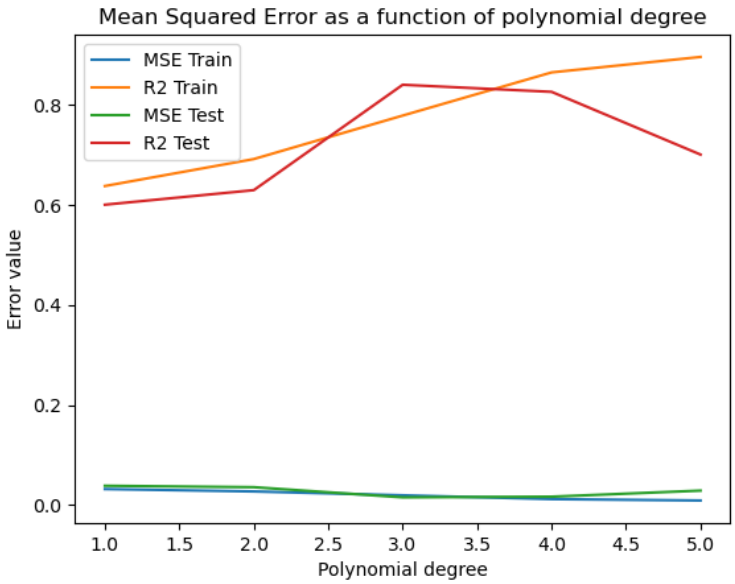


Figure 2 Mean Squared Error and R2 values for both training and test data as a function of polynomial degree. The plot illustrates how model complexity represented by the degree of polynomial, impacts the MSE and R2 scores. The current plot uses n=100

In addition, we plotted the parameters β as we increased the order of the polynomial.

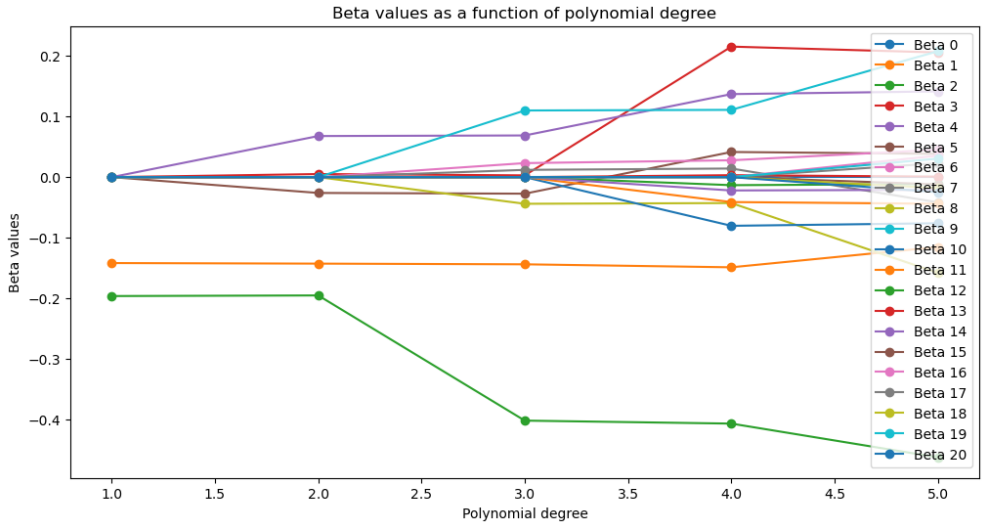


Figure 3 Beta coefficients as a function of polynomial degree. The plot displays how each beta value evolves with increasing polynomial complexity.

As the growth follows a binomial expansion, we get twenty-one different beta values because.

$$d = degree$$

$$Num\ Of\ Terms = \frac{(d+2)(d+1)}{2}$$

As the complexity of the model gets increases, we observe a corresponding rise in the number of coefficients. Specifically, as the polynomial degree increase, the magnitude of the beta coefficients also grows. Certain terms in the intermediate degrees exhibit particularly large beta values. The terms reflects the model's ability to capture more localized peaks and dips in data. Furthermore, we notice that the beta coefficients tend to cluster around origin which is due to the distribution of the x and y values. The alternating sign of the coefficients, a characteristic often observed in polynomial regression models, particularly of higher order, reflect the model's attempt to fit more complex patterns in the data.

Exercise b) Ridge

We can repeat the analysis from the previous exercise with small adjustments. We continue to draw one hundred randomly distributed samples between 0 and 1 and apply them to the Franke function with noise.

First, let us investigate how the MSE and R2-score of the Ridge regression changes as a function of different polynomial degrees for different values of lambda.

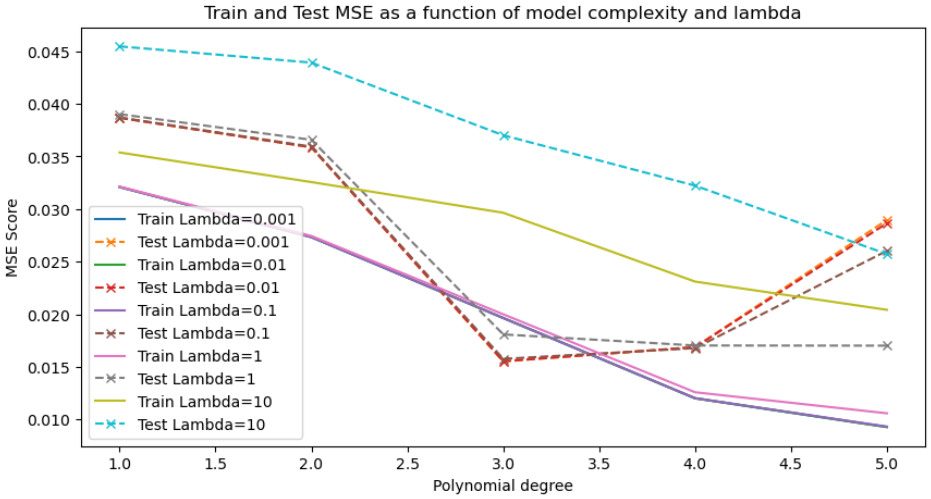


Figure 4 The MSE score for training and test data as a function of polynomial degree and regularization parameter in Ridge regression. The solid line represents the training data, and the dashed represent the test data.

We observe that from figure 4 that Ridge regression behaves similarly to OLS regression for the selected λ values, particularly up to polynomial degree five. The train MSE goes towards zero regardless of the lambda value and degree. On the other hand, test-MSE initially decreases towards zero but will eventually start to increase. This increase in test seems to be dependent on the value of λ, as lower values increase faster than higher values of λ. We observe that the test MSE for $\lambda = 10$ never seem to increase but $\lambda = 0.001$ increases dramatically after polynomial degree = 5.

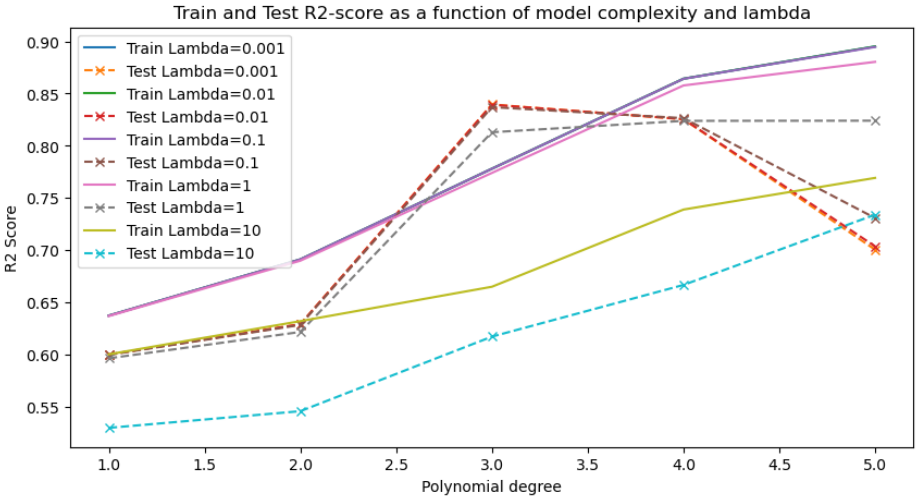


Figure 5 The R2 score for training and test data as a function of polynomial degree and regularization parameter in Ridge regression. The solid line represents the training data, and the dashed represent the test data.

Similarly, we observe the same behaviour for the R2-score. The train R2-score goes towards one regardless of polynomial degree, for all λ values, and the test R2-scores initially increases towards one, but will eventually decrease. We also notice that the decrease is dependent on the value of λ.

Exercise c) Lasso

Our second shrinkage method, called Lasso, will be analyzed like the first one. The MSE and R2 plots are illustrated in figure 6.

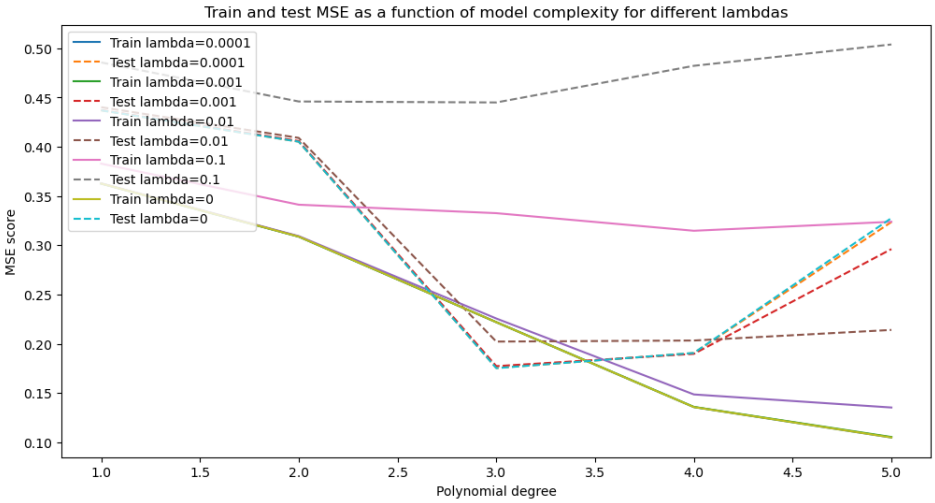


Figure 6 Visual representation of the MSE score for both training and test data using Lasso regression. The solid lines represent the training data, and the dashed line represents the test data for different lambda values, plotted as a function of polynomial degree.

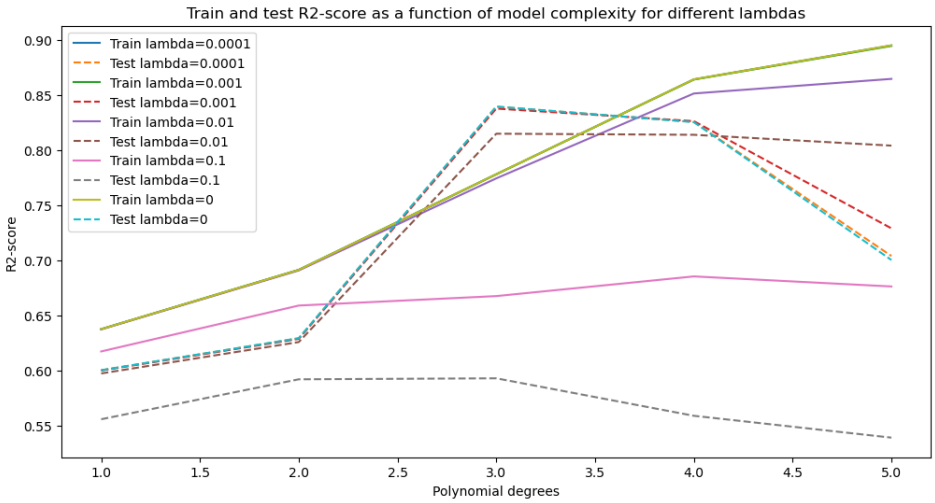


Figure 7 Visual representation of the R2 score for both training and test data using Lasso regression. The solid lines represent the training data, and the dashed line represents the test data for different lambda values, plotted as a function of polynomial degree.

Let us first discuss the MSE score shown in figure 6. For our train data we notice a decreasing trend in our MSE. This also seem dependent on the lambda value, where lambda values closer to zero has a better MSE score on the train data. For the test data the MSE score initially has a decreasing trend when the complexity rise but will eventually start to increase. Looking at the R2-score in figure 7 we notice the same behavior but instead of approaching zero, the model approaches one.

Through this observation we can argue that the regularization effect by Lasso diminishes as lambda approaches zero, meaning the model resembles an OLS regression. Additionally, higher order polynomials allow the models to become more flexible, enable it to fit the training data almost perfectly. This comes with a cost, the model is likely overfitting the training data, which is shown by the increase/decrease of MSE and R2 scores respectively. Furthermore,

Exercise e) Bias-Variance trade-off

This task evolves around the study of the bias-variance trade-off and the implementation of the bootstrap resampling technique. We would like the inform that we will only use the simpler ordinary least square for this exercise.

Before we continue with our analysis, we were asked to recreate the figure from Hastie, et al. This recreation is meant to illustrate how the bias-variance trade-off works. The comparison are in the appendix.

We will now take a closer look at the mean square error as a function of the complexity of our Franke Function to perform our analysis.

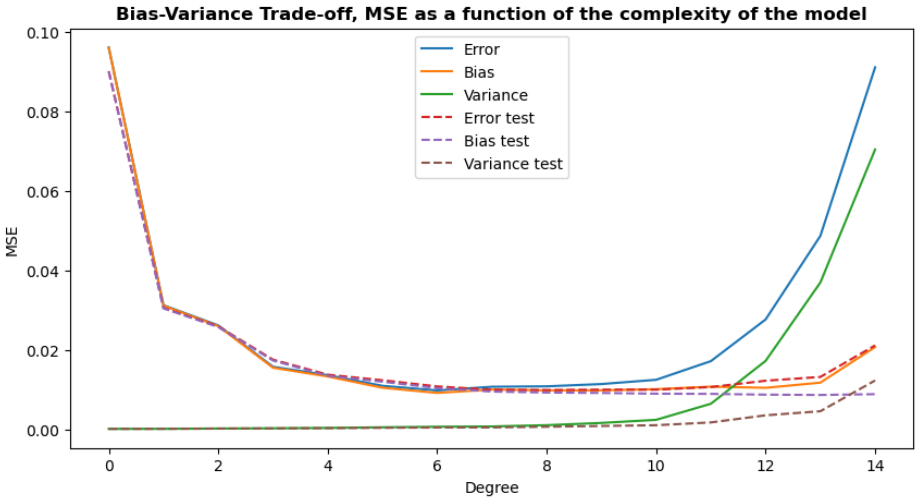


Figure 8 Bias-Variance trade-off for both training and test data, illustrated as a function of polynomial degree. The solid line represents the MSE, bias and variance for the training data, while the dotted lines represent the corresponding metrics for the test data.

From figure 8 we see that low complexity result in high bias, and with an increase in complexity the bias decrease. In contrast, low complexity result in low variance, and as we increase the model complexity the variance also increases. The error is as stated the Mean Square Error, but in this context, it can be understood as being comprised of the bias, variance and the irreducible error. The irreducible error is the random noise we added to our Franke function, and we notice that our error can be described as

$$MSE \approx Bias^2 + Variance$$

The bias-variance trade-off illustrates the delicate balance between model complexity and prediction accuracy. By leveraging bootstrap resampling, we gain a clearer understanding of this trade-off, facilitating better model selection and improved predictive performance.

Exercise f) Cross-validation

In exercise f, k-fold cross-validation is used, as opposed to bootstrapping. Furthermore, Ridge- and Lasso regression are also implemented in this exercise.

In the implementation of exercise f, the dataset is not split until the cross-validation starts. The design matrix is split into training and test data for each iteration of folds. Each regression method is tested on all polynomial degrees (OLS), and polynomial degrees and lambda values (ridge and lasso). This follows the implementation done in the code example for Cross-Validation and k-fold Cross-Validation from the weekly material from week 37 (https://compphysics.github.io/MachineLearning/doc/LectureNotes/_build/html/week37.html).

Furthermore, the code analyzes the MSE gathered from the cross-validation and gives us insights into the best polynomial degree and combination of polynomial degree and lambda value.

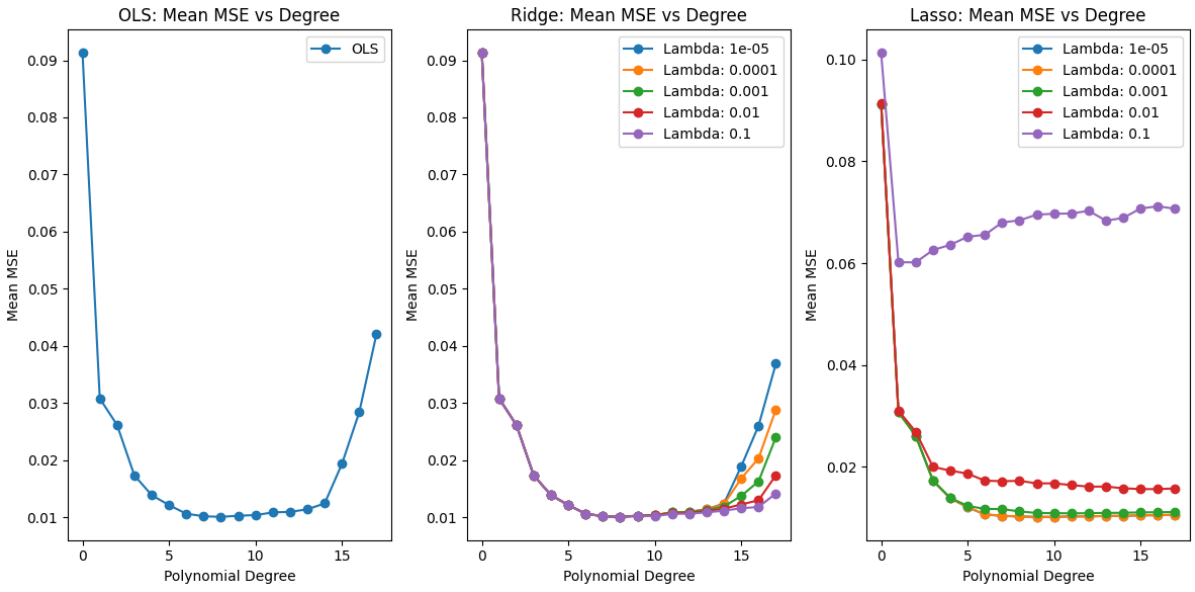


Figure 9 Figure 9 Illustrates the mean MSE over all k-folds as a function of polynomial degree, for different lambda values.

Table 1 Show the MSE for both train and test data for different polynomial degrees and lambda values.

Model	Poly. degree	Lambda	MSE: Train	MSE: Test
OLS	8		0.0091	0.0096
Ridge	8	0.1	0.0091	0.0097
Lasso	9	1e-05	0.0094	0.0100

The graph for OLS shows that when the model complexity is both low and high, the MSE is also high. As expected, the results are very much alike in exercise f. Due to utilizing resampling in both exercises, the results are more robust and reproducible. Furthermore, MSE from the train-, and test set are very close. This suggests that the model generalizes well opposed to overfitting, with the optimal polynomial degree.

As with OLS, the graph for ridge regression also shows that when the model complexity is both low and high, the MSE is also high. However, it's clear that lambda values, to some extent, prevent overfitting when model complexity becomes high, compared to OLS. In this case, it is also clear that the model generalizes well given the small difference between the different MSE values.

Lastly, the graph for lasso regression shows that when the model complexity is low, the MSE is high. In contrast to OLS and ridge regression, the graph does not show an increase in MSE with the higher model complexity. This suggests that lasso regression's lambda value prevents overfitting better than ridge regression in this case. Since lasso regression's regularization parameters might reduce some of the coefficients to zero and ridge's do not, the comparison between ridge- and lasso regression at higher polynomial degree, might suggest that some of the polynomial features are insignificant.

By further study of lasso regression's graph, it becomes clear that the higher lambda values perform worse. The higher lambda values reduce or remove significant features, resulting in higher MSE. Lambda value 0.1's graph shows this. As the polynomial degree increases, more significant features are reduced or removed, resulting in higher MSE.

Exercise g) Real data

In the last exercise of the project, the previous exercises were to be implemented on real data. As the exercise gave the option to choose a dataset freely, this exercise looks into red wine quality prediction. A dataset extracted from the UCI Machine learning repository (https://archive.ics.uci.edu/dataset/186/wine+quality), includes ten features and a quality estimate between 0-10 for red wines. With good predictions, this could give insights into whether a wine is good or not, but also which features are the biggest driving factors towards good red wines, by looking at the different beta values.

We started by splitting the data into features X and the value we want to predict (wine quality, between 0-10) Y. We scaled X using sklearn's *StandardScaler* mentioned above.

Furthermore, sklearn's *train_test_split* method splits the data into training and testing data. We chose to use cross-validation instead of bootstrapping for this exercise and subsequently, the rest of the implementation looks similar to exercise f, however there are some differences. In this implementation, the testing set created is left untouched until the final testing of the models. This means that in the cross-validation, the initial training data is used to create a new training set and a validation set. In this exercise the bias-variance trade-off is also studied.

Lastly, we can test the model prediction ability with the testing data, using the optimal parameters we get from analyzing the MSE values from the training.

Table 2 Shows the MSE for test and train data from the wine dataset for different polynomial degrees and lambda values.

Model	Poly. degree	Lambda	MSE: val	MSE: test	RMSE: test
OLS	1		0.4280	0.3896	0.624
Ridge	2	1.0	0.4195	0.3894	0.624
Lasso	2	0.001	0.4166	0.3635	0.602

The graph for OLS shows that when the model complexity is low, the MSE is low, whilst when the model complexity is high, the MSE is high, indicating overfitting. Based on OLS's RMSE, we can tell that the average prediction error is 0.624 points. Whether this is good or not depends on the domain. With little knowledge of wine quality scoring, we are guessing this is a relatively good score.

The graph for ridge regression also shows the same trend for MSE and model complexity, however in this case, the MSE becomes very large after polynomial degree 3 and suggests overfitting. For the optimal parameters, RMSE is also at 0.6240, something we suggested as a relatively good score earlier.

The graph for lasso regression also shows the same trend as ridge regression, but in this case, the MSE does not go as high. The RMSE is also very much alike.

The result from this exercise is somewhat contradictory to the theory as well as the results from the previous exercise. One would think that as the model becomes more complex, the shrinkage methods would perform better than OLS, given the regularization parameter.

Discussion

We applied three key linear regression techniques - OLS, Ridge, and Lasso - on both the Franke function and on our Wine dataset. Each method was used to explore how they handle model complexity, regularization and prediction accuracy.

OLS served as a baseline model, providing insight into how the model fits without any form of regularization. However, OLS is prone to overfitting when the model complexity increases, especially with high-degree polynomials.

Our two shrinkage methods, Ridge and Lasso, were introduced to mitigate overfitting by adding a penalty term. For Ridge, this penalty leads to a smaller and more stable parameter estimate. This method helps balance the bias-variance trade-off as it reduces variance at the cost of introducing a slight bias. Lasso on the other hand, drives some coefficients to zero, making it particularly useful when dealing with high-dimensional data.

To better understand the bias-variance trade-off, we conducted experiments using bootstrapping. This method was used to quantify the variability in model predictions and

assess how different levels of complexity influence the trade-off between the two metrics. Further, cross-validation helped evaluate the model's generalizability, confirming the effects of regularization and polynomial degree on both the Franke function and the wine dataset.

Conclusion and perspectives.

Conclusion

In our project, we applied three fundamental regression techniques, Ordinary Least Squares (OLS), Ridge, and Lasso regression, on the Franke function and real-world data to investigate their performance in terms of fitting models of different complexity. With the use of Mean squared error and the R^2-score we sought to understand how each method behave when the model complexity changes and when the lambda values changes, and how this influences prediction accuracy on both training and test datasets.

For our results, we observed that OLS tend to overfit the data as the model complexity increases, resulting in low training errors but rapidly increasing test errors. This reflects the model's susceptibility to high variance and overfitting in the absence of regularization. Ridge regression, with its L2 regularization, provided better control over the variance by penalizing large coefficients, thus showing improved generalization performance over OLS at higher complexities. Lasso regression, employing L1 regularization, offered the added benefit of feature selection by driving certain coefficients to zero. However, its performance on the Franke function depended on the selected hyperparameter values (lambda), making it more sensitive to tuning.

When studying the bias-variance trade-off, we found that simpler models tend to have high bias and low variance, while more complex models exhibit low bias but high variance. The use of Ridge and Lasso regression helped reduce the variance compared to OLS by introducing regularization, which in turn slightly increased bias but resulted in better generalization to on unseen data.

Future work

This is our first attempt performing a regression analysis and working with resampling methods and thus there are room for improvements. Our algorithms could include or explore other regression methods for comparison, and we might want to look at other performance metrics that suites our chosen method best. In addition, exploring alternative regularization techniques, like Elastic Net, could provide a more flexible approach.

Appendix

Wine quality dataset

The dataset used for exercise g can be found at
https://archive.ics.uci.edu/dataset/186/wine+quality

GitHub

The source code and results can be found at https://github.com/martin-og-ingar/FYS-STK4155-Project1

Task e) comparison.

Fig. 2.11 of Hastie, Tibshirani, and Friedman.

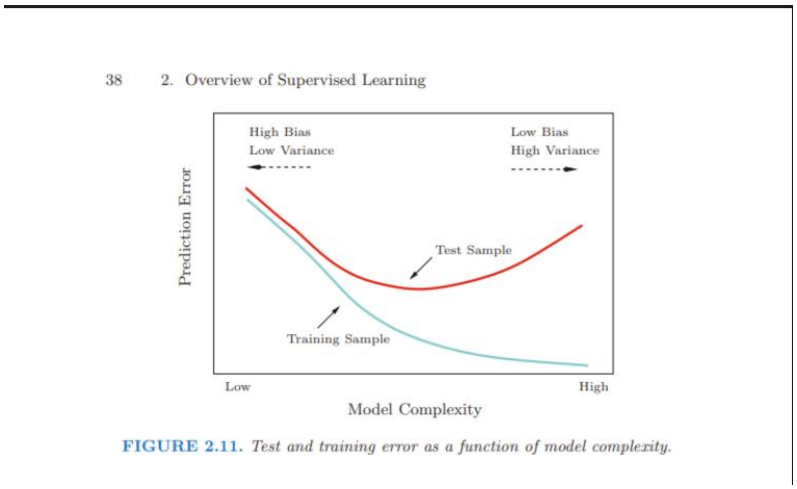
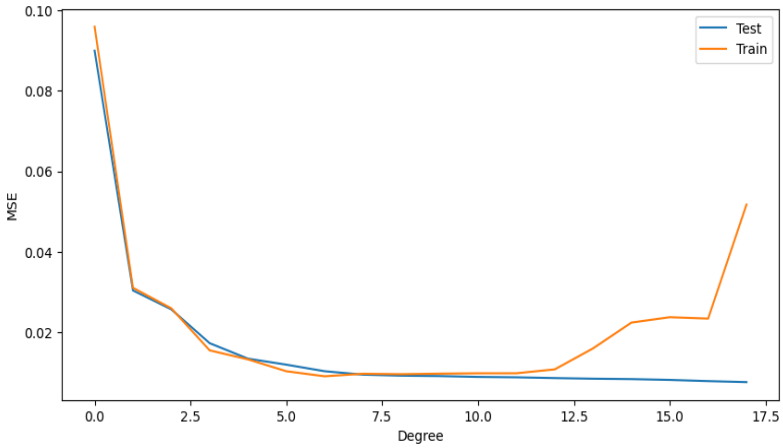


FIGURE 2.11. *Test and training error as a function of model complexity.*

Our recreation.



Task d, pen and paper.

Link to task d)

https://github.com/martin-og-ingar/FYS-STK4155-Project1/blob/main/results/penAndPaper/taskD.pdf

Task e, Bias-variance mathematics

Link to task e)

https://github.com/martin-og-ingar/FYS-STK4155-Project1/blob/main/results/penAndPaper/taskE_1.pdf

Bibliography.

Goodfellow, I., Bengio, Y., & Courville, A. (2016). *Deep learning*. The MIT Press.

Hastie, T., Tibshirani, R., & Friedman, J. H. (2017). *The elements of statistical learning: Data mining, inference, and prediction* (Second edition, corrected at 12th printing 2017). Springer.

Morten Hjorth-Jensen. (2024). *CompPhysics/MachineLearning* [Computer software]. CompPhysics. https://github.com/CompPhysics/MachineLearning (Original work published 2017)

Pedregosa, F., Varoquaux, G., Gramfort, A., Michel, V., Thirion, B., Grisel, O., Blondel, M., Prettenhofer, P., Weiss, R., Dubourg, V., Vanderplas, J., Passos, A., Cournapeau, D., Brucher, M., Perrot, M., & Duchesnay, É. (2011). Scikit-learn: Machine Learning in Python. *Journal of Machine Learning Research*, *12*(85), 2825–2830.

Ranstam, J., & Cook, J. A. (2018). LASSO regression. *British Journal of Surgery*, *105*(10), 1348–1348. https://doi.org/10.1002/bjs.10895

Raschka, S., Liu, Y., Mirjalili, V., & Dzhulgakov, D. (2022). *Machine learning with PyTorch and Scikit-Learn: Develop machine learning and deep learning models with Python*. Packt.