FYS-STK4155 - Project1

Gard, Are, David Andreas Bordvik

October 4, 2021

Motivation

In Project 1, we are tasked to study various regressions methods, such as Ordinary Least Squares, Ridge and Lasso. Our first area of study is how to fit polynomials to a specific two-dimensional function called Franke's Function. Our motivation behind fitting polynomials to Frank's function is to test the implementation of our regression algorithms, as well as studying various techniques such as bootstrapping and measurements such as the bias-variance tradeoff. Finally, we will move on to use real digital terrain data for our analysis.

The Franke Function is given on the form

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

with a 3-dimensional plot given in Figure **??**

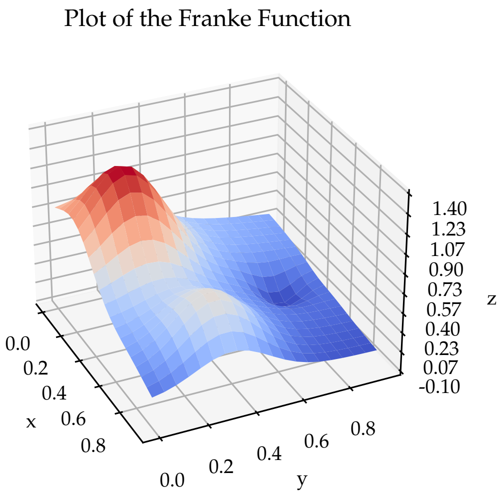

Figure 1: Plot of the Franke Function

Exercise 1

In Machine Learning, we are studying the problem of optimization, that is, finding the optimal parameter β such that $C(\boldsymbol{X}, \boldsymbol{\beta}) = \min_{\boldsymbol{\beta} \in \mathbb{R}^p} \frac{1}{n} \left\{ (\boldsymbol{y} - \boldsymbol{X}\boldsymbol{\beta})^T (\boldsymbol{y} - \boldsymbol{X}\boldsymbol{\beta}) \right\}$ our cost function is minimized. For the following exercise, where we will study Ordinary Least Squares regression, the previously stated cost function is the one we will minimize in order to fit the Franke Function, both without (as in Figure (**??**)) and with noise as in Figure (**??**).

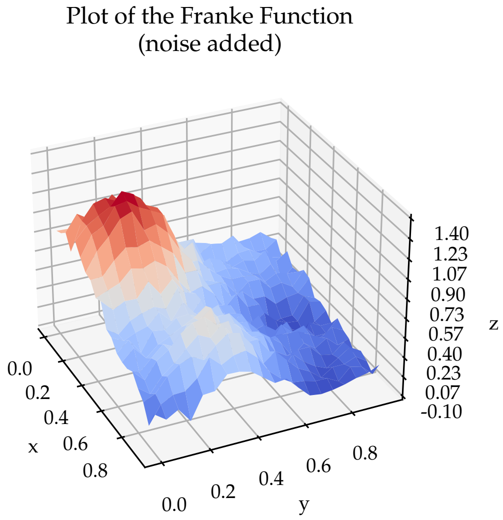

Figure 2: Plot of the Franke Function with added stochastic noise

When constructing our OLS model, we start off by minimizing the cost function with regards to β. That is, we take the derivative of $C(\boldsymbol{X}, \boldsymbol{\beta})$ and set it 0. The following derivation shows how we end up with the optimal parameters $\hat{\boldsymbol{\beta}}$, which in turn can be used to predict new values for the function which we are fitting.

$$\frac{\partial C(\boldsymbol{X}, \boldsymbol{\beta})}{\partial \beta} = 0$$

$$\frac{\partial C(\boldsymbol{X}, \boldsymbol{\beta})}{\partial \beta} = -\frac{2}{n} \boldsymbol{X}^{\mathrm{T}} (\boldsymbol{y} - \boldsymbol{X}\boldsymbol{\beta}) = 0$$

$$\boldsymbol{X}^{\mathrm{T}} (\boldsymbol{y} - \boldsymbol{X}\boldsymbol{\beta}) = 0$$

$$\boldsymbol{X}^{\mathrm{T}} \boldsymbol{X}\boldsymbol{\beta} = \boldsymbol{X}^{\mathrm{T}} \boldsymbol{y}$$

$$\hat{\boldsymbol{\beta}} = \left(\boldsymbol{X}^{\mathrm{T}} \boldsymbol{X}\right)^{-1} \boldsymbol{X}^{\mathrm{T}} \boldsymbol{y}$$

For consistency, it is noted that $\hat{\boldsymbol{\beta}} \in \mathbb{R}^p$, $\boldsymbol{y} \in \mathbb{R}^n$ and $\boldsymbol{X} \in \mathbb{R}^{n \times p}$.

With an expression for the predictors $\hat{\beta}$ derived, fitting a new value $\tilde{y}$ is simply $\tilde{y} = \boldsymbol{X}\hat{\boldsymbol{\beta}}$.

Our own implementation of Ordinary Least Square regression is implemented such that its use mimics that of SciKit-learn. [**?**] That is, we also include separate steps for initializing our model, fitting the model and predicting using the model using the just derived mathematical expression. For further inquiry about implementation, refer to the github repository linked in the **??**.

Confidence Intervals

Confidence intervals can be used to asses the uncertainty of a parameter. In our case, we will define confidence limits following the understanding of a Confidence Interval given in "Bootstrap Methods and their Application". That is, given a computed confidence region, any value inside the confidence region should be more likely than all values outside the confidence region. [?]

Furthermore, when computing the confidence interval for the parameters β, we first compute the variance $\boldsymbol{\sigma^2}(\beta_j) = \boldsymbol{\sigma^2}\left[(\boldsymbol{X}\boldsymbol{X}^{\mathrm{T}})^{-1}\right]_{\mathrm{jj}}$. Where $\boldsymbol{X}\boldsymbol{X}^{\mathrm{T}}$ is the Hessian matrix. Furthermore, it can be shown that the Hessian matrix can be given as the second derivative of the Cost function with respect to β. I.e.

$$\frac{\partial^2 C}{\partial \boldsymbol{\beta}^{\mathrm{T}} \partial \boldsymbol{\beta}} = \frac{2}{n}\boldsymbol{X}\boldsymbol{X}^{\mathrm{T}}$$

Mean Squared Error and R2 score

Two metrics that can be used to asses the quality of a model is its Mean Square Error (MSE) and R2 score. The MSE for any estimator is defined as

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

It will be shown later that the MSE can be broken into two components, namely the variance and the squared bias. As can be seen from the equation, the MSE would attain the value of 0 if $y_i = \tilde{y}_i$. Moreover, by rewriting the mean squared error as $\mathrm{MSE} = \frac{1}{m}\left\|\boldsymbol{y} - \tilde{\boldsymbol{y}}\right\|_2^2$, it can be seen that the error increases as the Euclidean distance between the prediction targets increase. [?]

The R2 score (coefficient of determination) is another metric that can be related to how the model covers its own variance. Defined as,

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

the closer the R2 score is to its maximum value 1, the more the variance of the model is explained by the model parameters. The R2 score gives a measure of model skill as a higher R2 generally indicates that the model is better at making new predictions. However, a perfect R2 score of 1 would result in the model covering it's entire variance, thus the model is overfitted and will not perform well in a general use case beyond the scope of it's initial training-data.

TODO Possibly more one implementation

Scaling the data

Our motivation for scaling the data arise in the context of fitting a design matrix of predictors with different units. To avoid having to disregard some predictors in favor of others based on their unit, not necessarily their contribution to the function fit, a scaling of the data is performed. By scaling the data using one of several scaling techniques, we ensure that all predictors lie in the same reference domain, resulting in a more accurate representation of the predictors. Furthermore, scaled data generally increases model skill.

SciKit-learn includes several different Scalers, such as the StandardScaler and MinMaxScaler. [?] For this discussion, the StandardScaler will be inspected.

The idea behind scaling the data with regards to the StandardScaler method, is to subtract the mean value and then divide by the variance. By performing these two operations, we ensure that the data have a mean value equal to 0 (i.e. are standardized/zero centered) and variance equal to 1.

Before scaling the dataset, an assessment of how to deal with the intercept has to be made. For reference, the intercept is defined as the first column of the design matrix, and for a polynomial fit would represent where the function intercepts with the y-axis when all other features are set to zero. Moreover, the intercept is a constant value (for our design matrices equal to 1), thus zero centering the intercept would result in a singular matrix, rendering the optimization problem unsolvable. Throughout this assignment, we are going to readd the intercept after scaling, essentially leaving the intercept untouched as to avoid any singular matrices. Moreover, for ordinary least squares regression, as there is no regularization of the predictors during the model fit, a model fitted on scaled contra unscaled data would attain the same mean squared error.

However, other regression methods such as Ridge and Lasso, which will be discussed in greater detail in further sections, have a dependance on the intercept through the hyperparamter λ when computing their regularization term. Not including the intercept when computing the regularization term would give rise to a divergence between the mean square error for the model scaled with compared to the model scaled without the intercept. Thus, by computing the regularization term with disregard to the intercept, the first β_j i.e. that of the intercept will be skipped in the computation. This will in most cases lead to a better mean square error for both Ridge and Lasso regression. Skipping the intercept when computing the regularization term also follows the definition of Ridge regression as in. [?]

Furthermore, while on the topic of Ridge and Lasso regression, a scaling of the data should always be performed. This is due to the regularized linear models such as Ridge and Lasso being sensitive to the scale of the input features. [?]

To determine whether scaling is apropriate for the current problem, that being fitting the Franke Function using oridnary least squares, an inspection of the generated data is made in light of the just discussion. For Exercise 1, the datapoints $x, y \in [0, 1]$. This would indicate that the data is already scaled to a unit reference system. Moreover, as we are training a model based on the ordinary least squares, there is no dependance on scale of the input features as for regularized linear models. However, for consistency with further models, the data will be scaled with respect to the training data. However, the target variables however will not be scaled.

Splitting the data

As we want our model to perform well in general cases, we split the data into a training and testing set to simulate model prediction using new data. This is achieved by the aforementioned split, since the training and test data are kept entirely separated. In practice, we fit the model using the training data, then perform a test of the model using the test data. The error rate for new cases predicted by the model using the test data, can be used to understand how the model will perform on new untrained data. [?] Moreover, by assessing the deviation between training error and test error, it can be seen whether the model is overfitting or not. It would be a case of overfitting if the training error is low, whereas the test error is high.

Throughout this assignment, we will split the data into a train and test set. The data could be split into an additional validation set, which is normally used for hyperparamter adjustment. However, as we don't see any practical use for a validation set for this assignment, we will skip out on splitting the data into an additional validation set. Though a validation set could be used for tuning the hyperparameter to select an optimal model, it could also lead to suboptimal model selection caused by imprecise prediction from a too small validation set. [?] Though our data consists of potentially unlimited data points, due to computational time constraints, we will resort to a relatively sparse dataset. Hence, we split into a training and test set to avoid sacrificing the training data in favor of a sufficient validation set.

Had we not omitted the validation set for hyperparameter tuning, the process of studying regularized models would have deviated somewhat from the study of the ordinary least squares regression. The process would then be that we split the data into a train-test split, as before. Moreover, the training data would have been split once more into a train and validation set, with an approximate ratio of 80 - 20 percent respectively. Furthermore, the hyperparameters are tuned with the MSE obtained from predicting using the validation set. However, as the validation set typically reports a lower error than the test set, the generalization error of the optimized model is studied using the test set. [?]

Comparing our OLS implementation to the one delivered by SciKit-learn

With our Ordinary Least Squares model implemented as described above, we start off by benchmarking our implementation to the LeastSquares method found in SciKit-learn. [?] We start off by setting up a uniform 2-dimensional grid and initialize a Franke Function with some added stochastic noise.

```
np.random.seed(4155)
n = 100 # The number of points in direction for the Franke Function
x = np.sort(np.random.uniform(0, 1, n))
y = np.sort(np.random.uniform(0, 1, n))
x, y = np.meshgrid(x,y)
z = FrankeFunction(x, y) + noise_factor(n,factor=0.05)
```

For this initial run, we are interested in studying the least squares fit of the Franke Function up to the fifth order.

By inspecting Figure (??), we can see that there are no visual differences between our implementation of OLS compared to the SciKit-learn implementation. Moreover, there is a reduction in MSE as model complexity increases. Thus it is clear that a higher order fit of the model results in a lower error when predicting new values using the test data.

Comparing Figure (??) and (??), where the first figure features half the amount of added noise compared to the second figure, there are two things of note. Firstly, Figure (??) attains an overall lower MSE, even as the order of the fitted function increases. Secondly, the initial difference between the train and test MSE is greater for Figure (??) than for Figure (??), though the difference converges to zero for higher order fits in both figures. As such, we make a preliminary conclusion that fitting functions with a higher amount of added Gaussian noise is more difficult than for smooth ”predictable” functions.

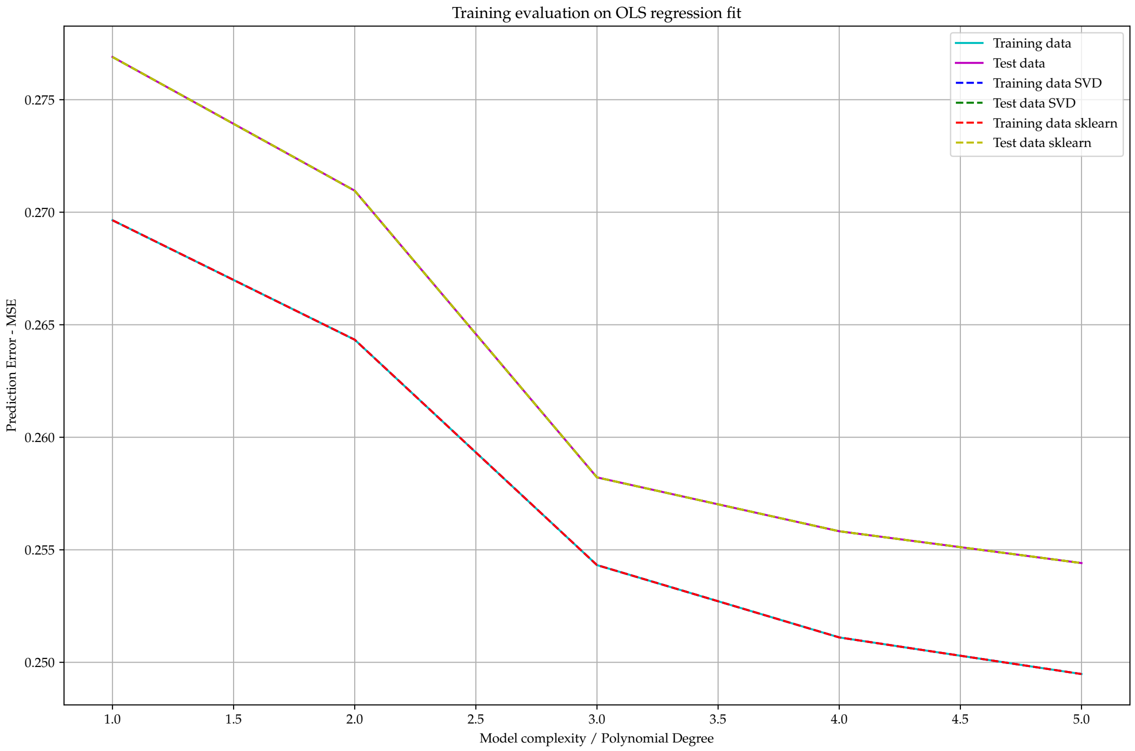

Figure 3: Benchmark run of the Franke Function fit with added Gaussian noise (factor of 0.05) with degree up to the fifth order of our OLS implementation compared to the similar LinearRegression() from SciKit-learn

figures/EX1_franke_function_OLS_evaluate_fit_dn.pdf

Figure 4: Benchmark run with doubled the amount of added Gaussian noise (factor 0f 0.1) of our OLS implementation to the similar LinearRegression() from SciKit-learn

The Confidence Interval for the predictors β_j is constructed for the fit using up to fifth order polynomials. As the predictors are based not only of the x and y parameters in isolation, but also their interaction terms, a fifth order fitted model includes 21 different predictors. The result of computing the Confidence Interval for the 21 different predictors can be seen in Figure **??**. Note that the predictors From Figure **??**, it can be seen that the lowermost and higher order terms have the smallest confidence intervals. Thus some of the predictors related to x and y of the third and fourth order pose the highest uncertainty.

Following the comparison of Figure (**??**) and (**??**), comparing the two confidence intervals reveals that as the added Gaussian noise is increased in size, the confidence intervals for the predictors increase as well. This can be seen especially for the predictors with some variability for the 0.05 noise factor case. Thus, as we increase the Gaussian noise added to the Franke Function, our predictors become less uncertain, which in turn could result in less precise model predictions.

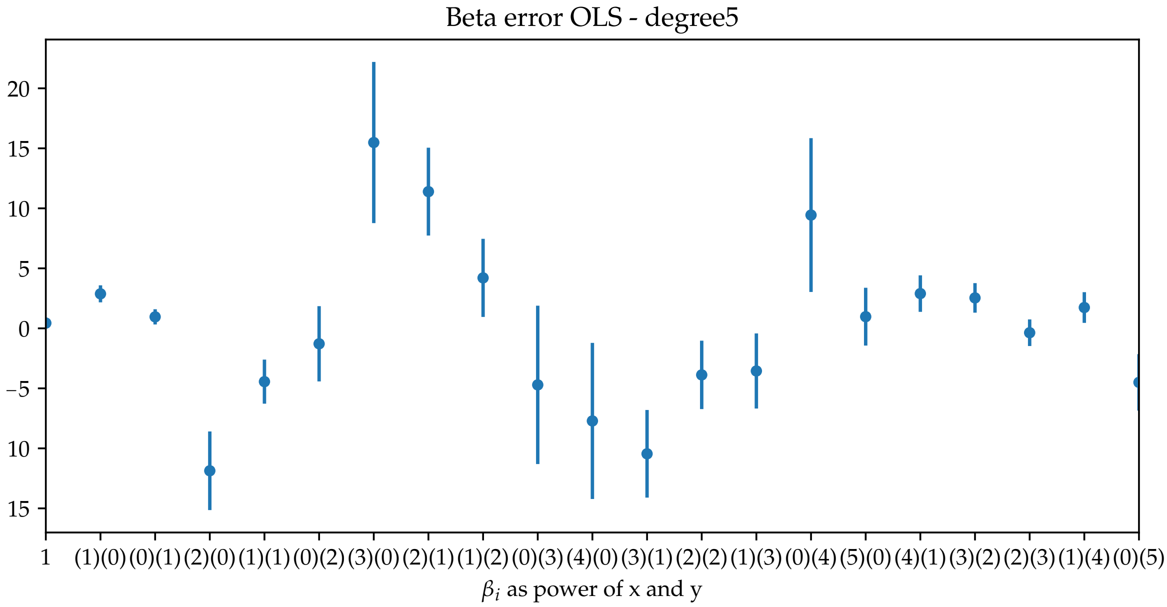

Figure 5: 95% Confidence intervals for the predictors of an OLS model with polynomials up to the fifth order and added Gaussian noise multiplied with a factor of 0.05.

figures/EX1_OLS_beta_error_degree5_dn.pdf

Figure 6: 95% Confidence intervals for the predictors of an OLS model with polynomials up to the fifth order and added Gaussian noise multiplied with a factor of 0.1.

Exercise 2

Some words on bootstrapping

Moving on to the bias-variance trade-off analysis, we start off by showing that

$$C(\boldsymbol{X}, \boldsymbol{\beta}) = \frac{1}{n}\sum_{i=0}^{n-1}(y_i - \tilde{y}_i)^2 = \mathbb{E}\left[(\boldsymbol{y} - \tilde{\boldsymbol{y}})^2\right]$$

Can be rewritten as

$$\mathbb{E}\left[(\boldsymbol{y} - \tilde{\boldsymbol{y}})^2\right] = \frac{1}{n}\sum_i (f_i - \mathbb{E}\left[\tilde{\boldsymbol{y}}\right])^2 + \frac{1}{n}\sum_i (\tilde{y}_i - \mathbb{E}\left[\tilde{\boldsymbol{y}}\right])^2 + \sigma^2.$$

where the terms are respectively (bias)^2, variance and noise. For simplicity, assume that we have a dataset where the data is generated from a noisy model

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

Furthermore, we will assume that the residuals ϵ are independant and normally distributed $\epsilon \sim \mathcal{N}(0, \sigma^2)$. Finally, $\tilde{\boldsymbol{y}} = \boldsymbol{X}\boldsymbol{\beta}$ is our approximation to the functions f. Start off by adding and subtracting $\mathbb{E}\left[\tilde{\boldsymbol{y}}\right]$ inside the expectation value.

$$\begin{aligned}\mathbb{E}\left[(\boldsymbol{y} - \tilde{y})^2\right] &= \mathbb{E}\left[(\boldsymbol{y} - \tilde{\boldsymbol{y}} + \mathbb{E}\left[\tilde{y}\right] - \mathbb{E}\left[\tilde{\boldsymbol{y}}\right])^2\right] \\ &= \mathbb{E}\left[((\boldsymbol{y} - \mathbb{E}\left[\tilde{y}\right]) - (\tilde{\boldsymbol{y}} - \mathbb{E}\left[\tilde{\boldsymbol{y}}\right]))^2\right]\end{aligned}$$

By using the fact that $\boldsymbol{y} = f(\boldsymbol{x}) + \boldsymbol{\epsilon}$ we can rewrite this as

$$= \mathbb{E}\left[((f(\boldsymbol{x}) - \mathbb{E}\left[\tilde{\boldsymbol{y}}\right]) - (\tilde{\boldsymbol{y}} - \mathbb{E}\left[\tilde{\boldsymbol{y}}\right]) + \epsilon)^2\right]$$

Computing the square inside the expectation value gives us

$$= \mathbb{E}\left[(f(\boldsymbol{x}) - \mathbb{E}\left[\tilde{\boldsymbol{y}}\right])^2 + (\tilde{\boldsymbol{y}} - \mathbb{E}\left[\tilde{\boldsymbol{y}}\right])^2 + \epsilon^2\right]$$

$$+2\left(\mathbb{E}\left[\epsilon(f(\boldsymbol{x}) - \mathbb{E}\left[\tilde{\boldsymbol{y}}\right])\right] - \mathbb{E}\left[\epsilon(\tilde{\boldsymbol{y}} - \mathbb{E}\left[\tilde{\boldsymbol{y}}\right])\right] - \mathbb{E}\left[(f(\boldsymbol{x}) - \mathbb{E}\left[\tilde{\boldsymbol{y}}\right])(\tilde{\boldsymbol{y}} - \mathbb{E}\left[\tilde{\boldsymbol{y}}\right])\right]\right)$$

Moreover, as ϵ are independant variables, the expectation value involving them as a product can be written as a product of expectation values. Knowing that $\mathbb{E}\left[\epsilon\right] = 0$, the third and second to last term is equal to zero. Also, knowing that $\mathbb{E}\left[\tilde{\boldsymbol{y}}\right] = \tilde{\boldsymbol{y}}$, the last t

$$= \mathbb{E}\left[(f(\boldsymbol{x}) - \mathbb{E}\left[\tilde{\boldsymbol{y}}\right])^2\right] + \mathbb{E}\left[(\tilde{\boldsymbol{y}} - \mathbb{E}\left[\tilde{\boldsymbol{y}}\right])^2\right] + \mathbb{E}\left[\epsilon^2\right] \tag{1}$$

The first term in (**??**) can be discretized as

$$\mathbb{E}\left[(f(\boldsymbol{x}) - \mathbb{E}\left[\tilde{\boldsymbol{y}}\right])^2\right] = \frac{1}{n}\sum_i (f_i - \mathbb{E}\left[\tilde{\boldsymbol{y}}\right])^2$$

Which is the bias squared as we were to show.

The second term in (**??**) is also discretized, yielding

$$\mathbb{E}\left[(\tilde{\boldsymbol{y}} - \mathbb{E}\left[\tilde{\boldsymbol{y}}\right])^2\right] = \frac{1}{n}\sum_i (\tilde{\boldsymbol{y}} - \mathbb{E}\left[\tilde{\boldsymbol{y}}\right])^2$$

Which takes form of the variance, as was set out to show.

Finally, it can be shown that $\mathrm{var}(\boldsymbol{y}) = \mathrm{var}(f+\epsilon) = \mathbb{E}\left[(f+\epsilon)^2\right] - (\mathbb{E}\left[(f+\epsilon)\right])^2 = \mathbb{E}\left[\epsilon^2\right]$ As such, we can use that

$$\mathrm{var}(y) = \sigma^2 = \mathbb{E}\left[\epsilon^2\right]$$

to see that the final term in (**??**) is equal to the noise. Thus we have shown that

$$\begin{aligned}\mathbb{E}\left[(\boldsymbol{y} - \tilde{\boldsymbol{y}})^2\right] &= \mathbb{E}\left[(f(\boldsymbol{x}) - \mathbb{E}\left[\tilde{\boldsymbol{y}}\right])^2\right] + \mathbb{E}\left[(\tilde{\boldsymbol{y}} - \mathbb{E}\left[\tilde{\boldsymbol{y}}\right])^2\right] + \mathbb{E}\left[\epsilon^2\right] \\ &= \frac{1}{n}\sum_i (f_i - \mathbb{E}\left[\tilde{\boldsymbol{y}}\right])^2 + \frac{1}{n}\sum_i (\tilde{y}_i - \mathbb{E}\left[\tilde{\boldsymbol{y}}\right])^2 + \sigma^2.\end{aligned}$$

The final expression consists of three terms. The first term is a sum of the squared bias, the second term is the variance and the final term is a constant noise term. The sum of squared bias and variance make up the mean square error of our model. [**?**]

The bias component of the mean square error measures the difference from the true mean to the desired regression model. As function of model complexity, the bias will decrease as complexity increases. The second term, the variance, gives a measurement of the variation of the model values around their average value. The variance will increase with model complexity. The constant noise term σ^2 is an irreducible error which can only be reduced by cleaning up the data beforehand. [**?**]. As the irreducible error is out of our control, it does not contribute when analyzing the bias-variance tradeoff.

The bias-variance tradeoff can be used as a method for model selection. As has been alluded to in the previous paragraph, the variance is inverse proportional to the bias. As such, there is a trade-off between bias and variance. As the bias-variance is directly related to the mean square error for both the training data but also test and production data used for making new predictions. When selecting a model we wish to balance and minimize both quantities, as that leads to the model with the best predicitve capabilities. [**?**]

Furthermore, the problem of overfitting can also be discussed in light of the bias variance tradeoff. Overfitting is proportional to the model variance, as such a high variance leads to an overfitted model. An overfitted model is one that has learned the noise of the training data, resulting in a perfect fit for the training data but a high mean square error when predicting using new data. Hence, a higher bias can be considered more useful to circumvent overfitting.

Lastly, it should be noted that the bias-variance tradeoff measuremnet is performed for a single dataset of limited size. As such, if there were more datapoints to train the model with, the model would attain a better overall fit. Moreover, a greater amount of training data would reduce the level of overfitting for a given model complexity. [**?**] Though there are some limitations to the measurement, the bias-variance tradeoff can be used to estimate where the trained model is general enough to both avoid under -and overfitting (i.e. too high bias or too high variance).

	MSE	R2-score
Training data	0.01636495753709963	0.8053964213340778
Test data	0.01636495753709963	0.8053964213340778

Exercise 4

Throughout Exercise 1 - 3, we have studied function fitting using the Ordinary Least Squares regression. When comparing models, we have plotted the test error as a function of the model complexity and chosen a model based on the bias-variance tradeoff. The bias-variance tradeoff has also been studied in light of resampling techniques such as the bootstrap and k-fold cross-validation. As we have seen when interpreting the result of the resampling, an increased degree of freedom for the model will make the model more prone to overfit.

Instead of reducing the degree of freedom of a model by limiting it's order, we will in this section look at a different approach for model selection through the introduction of a regularization parameter λ. Ridge regression is a regression model where the weights are constrained. [?] This is done by adding a regularization term to the cost function, such that the function we are to optimize becomes

$$C\left(\boldsymbol{X},\boldsymbol{\beta}\right) = \left\{\left(\boldsymbol{y}-\boldsymbol{X}\boldsymbol{\beta}\right)^{\mathrm{T}}\left(\boldsymbol{y}-\boldsymbol{X}\boldsymbol{\beta}\right)\right\} + \lambda\boldsymbol{\beta}^{\mathrm{T}}\boldsymbol{\beta}$$

Taking the derivatives of this function in terms of beta returns the closed form solution for the optimal $\boldsymbol{\beta}$

$$\hat{\boldsymbol{\beta}} = \left(\boldsymbol{X}^{\mathrm{T}}\boldsymbol{X} + \boldsymbol{\lambda I}\right)^{-1}\boldsymbol{X}^{\mathrm{T}}\boldsymbol{y}$$

Appendix

TODO her skal det ligge link til et github repo.