The Bias-variance-trade-off

Filip Severin von der Lippe

December 18, 2022

GitHub repository containing code and further instructions on how to reproduce the results of this report: https://github.com/Fslippe/FYS-STK4155/tree/main/project3/extra

1 Introduction

In this report we will perform a bias-variance trade-off analysis of different methods. We will look at a regression problem using the well known Boston housing dataset which contains various parameters of the neighbourhood influencing the median value of owner-occupied homes. We will look into decision trees, starting with a single tree and move over to a random forest. A Feed forward Neural Network will also be studied. We will start off by introducing the theory behind the bias-variance trade-off and the methods used in the analysis before presenting and discuss the results of the different methods. In the end we will summarize our findings and propose possible further analysis.

2 Method

Bootstrap

To study the bias-variance trade-off we first implement the bootstrap resampling technique. This is done by resampling our train data n_B times with replacement. That means we gain n_B new samples that may be unique but still only contain values from the original sample. This means that we can draw a sample from the original [1,2,3] to be for example [2,1,2]. For significant large amounts of training data, we are left with a minimal chance of drawing the same sample multiple times. From this we get by following the central limit theorem under the assumption that all our n_B subsets are drawn from the same distribution, a normal distribution describing our data with original length n with mean $\frac{1}{n}\sum_i y_i$ and standard deviation $\sigma/\sqrt{n_b - 1}$ for n_b new data subsets and the original standard deviation of the distribution σ.

Bias-variance trade-off

We define a cost function given by the mean squared error for a prediction vector $\tilde{\boldsymbol{y}} = \boldsymbol{X}\boldsymbol{\beta}$ and the true data vector $\boldsymbol{y}$.

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

Here we have defined our dataset on the form $\mathcal{D}_j = (\boldsymbol{x}_j, y_j)$ for $j = 0, 1, ..., n-1$. We continue by defining the input target data to be given by some function and noise $\epsilon \sim N(0, \sigma^2)$

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

By the definition of our data $\boldsymbol{y} = f + \boldsymbol{\epsilon}$ we rewrite the costfunction:

$$\begin{aligned}
\mathbb{E}[(\boldsymbol{y} - \tilde{\boldsymbol{y}})^2] &= \mathbb{E}[(f + \boldsymbol{\epsilon} - \tilde{\boldsymbol{y}} + \mathbb{E}[\tilde{\boldsymbol{y}}] - \mathbb{E}[\tilde{\boldsymbol{y}}])^2] \\
&= \mathbb{E}[f^2 + \boldsymbol{\epsilon}^2 + \tilde{\boldsymbol{y}}^2 + 2\,\mathbb{E}[\tilde{\boldsymbol{y}}]^2 \\
&+ 2f\boldsymbol{\epsilon} + 2f\tilde{\boldsymbol{y}} + 2f\,\mathbb{E}[\tilde{\boldsymbol{y}}] - 2f\,\mathbb{E}[\tilde{\boldsymbol{y}}] \\
&+ 2\boldsymbol{\epsilon}\tilde{\boldsymbol{y}} + 2\boldsymbol{\epsilon}\,\mathbb{E}[\tilde{\boldsymbol{y}}] - 2\boldsymbol{\epsilon}\,\mathbb{E}[\tilde{\boldsymbol{y}}] \\
&+ 2\tilde{\boldsymbol{y}}\,\mathbb{E}[\tilde{\boldsymbol{y}}] - 2\tilde{\boldsymbol{y}}\,\mathbb{E}[\tilde{\boldsymbol{y}}] - 2\,\mathbb{E}[\tilde{\boldsymbol{y}}]^2] \\
&= \mathbb{E}[(f - \mathbb{E}[\tilde{\boldsymbol{y}}])^2] + \mathbb{E}[\boldsymbol{\epsilon}^2] + \mathbb{E}[(\mathbb{E}[\tilde{\boldsymbol{y}}] - \tilde{\boldsymbol{y}})^2] \\
&+ 2(f - \mathbb{E}[\tilde{\boldsymbol{y}}])\,\mathbb{E}[\boldsymbol{\epsilon}] + 2\,\mathbb{E}[\boldsymbol{\epsilon}]\,\mathbb{E}[(\mathbb{E}[\tilde{\boldsymbol{y}}] - \tilde{\boldsymbol{y}})] \\
&+ 2(f - \mathbb{E}[\tilde{\boldsymbol{y}}])\,\mathbb{E}[(\mathbb{E}[\tilde{\boldsymbol{y}}] - \tilde{\boldsymbol{y}})].
\end{aligned}$$

By using that $\boldsymbol{\epsilon} \sim N(0, 1)$ we have $\mathbb{E}[\boldsymbol{\epsilon}] = 0$ and $\mathrm{Var}[\boldsymbol{\epsilon}] = \mathbb{E}[\boldsymbol{\epsilon}^2] - \mathbb{E}[\boldsymbol{\epsilon}]^2 = \mathbb{E}[\boldsymbol{\epsilon}^2] = \sigma^2$ from this we get:

$$\begin{aligned}
\mathbb{E}[(\boldsymbol{y} - \tilde{\boldsymbol{y}})^2] &= \mathbb{E}[(f - \mathbb{E}[\tilde{\boldsymbol{y}}])^2] + \mathbb{E}[(\tilde{\boldsymbol{y}} - \mathbb{E}[\tilde{\boldsymbol{y}}])^2] + \sigma^2 + 2(f - \mathbb{E}[\tilde{\boldsymbol{y}}])\,\mathbb{E}[\mathbb{E}[\tilde{\boldsymbol{y}}] - \tilde{\boldsymbol{y}}] \\
&= \mathbb{E}[(f - \mathbb{E}[\tilde{\boldsymbol{y}}])^2] + \mathbb{E}[(\tilde{\boldsymbol{y}} - \mathbb{E}[\tilde{\boldsymbol{y}}])^2] + \sigma^2.
\end{aligned}$$

Writing the expectation values in terms of the mean function $\mathbb{E}(\boldsymbol{x}) = \frac{1}{n}\sum_i x_i$, we get:

$$\begin{aligned}
\mathbb{E}[(\boldsymbol{y} - \tilde{\boldsymbol{y}})^2] &= \frac{1}{n}\sum_i (f_i - \mathbb{E}[\tilde{\boldsymbol{y}}])^2 + \frac{1}{n}\sum_i (\tilde{\boldsymbol{y}}_i - \mathbb{E}[\tilde{\boldsymbol{y}}])^2 + \sigma^2 \\
&= \quad \mathrm{Bias}^2 \quad + \quad \mathrm{Variance} \quad + \quad \mathrm{Noise}.
\end{aligned}$$

We face a problem in the calculation of the bias because of the unknown f_i. To get around this we rewrite the bias term with help from the definition $\boldsymbol{y} = f + \boldsymbol{\epsilon}$.

$$\begin{aligned}
\mathbb{E}[y - \mathbb{E}[\tilde{\boldsymbol{y}}]^2] &= \mathbb{E}[(f + \boldsymbol{\epsilon} - \mathbb{E}[\tilde{\boldsymbol{y}}])^2] = \mathbb{E}[\boldsymbol{\epsilon}^2] + \mathbb{E}[(f - \mathbb{E}[\tilde{\boldsymbol{y}}])^2] + \mathbb{E}[-2\boldsymbol{\epsilon}(f - \mathbb{E}[\tilde{y}])] \\
&= \sigma^2 + \mathbb{E}[(f - \mathbb{E}[\tilde{\boldsymbol{y}}])^2] + \mathbb{E}[-2\boldsymbol{\epsilon}]\,\mathbb{E}[f - \mathbb{E}[\tilde{\boldsymbol{y}}]] + \mathrm{Cov}(-2\boldsymbol{\epsilon}, (f - \mathbb{E}[\tilde{\boldsymbol{y}}])^2) \\
&= \sigma^2 + \mathrm{Bias}^2.
\end{aligned}$$

In practice, this means that our calculated bias using $\boldsymbol{y}$ instead of ϵ has an error of σ^2. By assuming this error is very small will the new way of calculating the bias be very accurate.

Data

The data we will use is the Boston housing set which can be loaded into python by scikit learn. The dataset contains 506 values for 13 features and one target feature as seen below in table 1.

Table 1: The Boston housing dataset and its features.

Feature	**Desctiption**
CRIM	per capita crime rate by town
ZN	proportion of residential land zoned for lots over 25,000 sq.ft.
INDUS	proportion of non-retail business acres per town
CHAS	Charles River dummy variable (= 1 if tract bounds river; 0 otherwise)
NOX	nitric oxides concentration (parts per 10 million)
RM	average number of rooms per dwelling
AGE	proportion of owner-occupied units built prior to 1940
DIS	weighted distances to five Boston employment centres
RAD	index of accessibility to radial highways
TAX	full-value property-tax rate per 10000$
PTRATIO	pupil-teacher ratio by town
B	$1000(B_k - 0.63)^2$ where B_k is the proportion of blacks by town
LSTAT	lower status of the population
MEDV	**Target data** - Median value of owner-occupied homes in 1000$

Since the bias-variance analysis can be computationally heavy, we sample only 202 random values from the original data. For this smaller dataset we perform a 80-20% train-test split before standard scaling the data as

$$y = \frac{y - \bar{y}_{train}}{\sqrt{\text{Var}(y_{train})}} \quad \text{and} \quad \frac{\boldsymbol{X} - \bar{\boldsymbol{X}}_{train}}{\sqrt{\text{Var}(\boldsymbol{X}_{train})}}.$$

Regression methods

Decision trees

For a regression case a decision tree starts by looking at all possible values $x_1, x_2, ..., x_p$ and making J non overlapping regions around these. This means that we for new input data falling into a region R_j will predict the value that initially made that region. The construction of the J different regions are often not made exactly as described above, but made with a top-down approach. This means that we begin at the top of the tree and add to new branches corresponding to two new regions. We describe this in terms of a cutpoint s splitting two regions R_1 and R_2 by

$$\{X|x_j < s\} \quad \text{and} \quad \{X|x_j \geq s\}.$$

Here we choose the feature j and the cutpoint s for the two regions based on the minimization of the MSE for the input design matrix X given by

$$\sum_{i:x_i \in R_1} (y_i - \overline{y}_{R_1})^2 + \sum_{i:x_i \in R_2} (y_i - \overline{y}_{R_2})^2.$$

This is done for every feature p and every one of the N data points, giving a total of $p \times N$ splits performed. The splitting process continues until all data has been classified, or until some stopping criteria such as a maximum depth of the tree or a minimum number of samples left in a leaf node. We will use the maximum depth of the tree as an adjustable parameter when we study the bias-variance trade-off. To generate the decision tree for our regression problem we use the DecisionTreeRegressor method from scikit learn.

Random Forest

A big problem often found with a single decision tree is overfitting. This can be explained by how the tree can have almost an infinite number of branches, perfectly splitting up the training data to classify all the training samples. This can of course be solved by restricting the tree with for example a maximum depth, but also be solved by initializing several trees. We now have a forest of trees where each of them are trained on different subsets of data. These data subsets are made through bootstrapping of the training data. The branch splitting is done differently in a random forest. Here we select $m \leq p$ random variables from the p predictors to consider for each split. This leaves us with trees that may look quite different from each other, because of the strongest predictors not always being among the m variables, leaving us with these predictors not always being in the upper branches. The average of every tree's prediction is what leaves us with the final prediction. The averaging of several uncorrelated trees as in the random forest will greatly reduce the variance in comparison to an averaging over correlated trees. The random forest we will also be implemented through scikit learn with the method RandomForestRegressor, and we will look at how the tree depth and the number of trees influence the bias-variance trade-off.

Feed forward Neural Network

A full description of a Feed Forward Neural Network (FFNN) can be found in project 2.

A FFNN is built up by layers of neurons each connected to every neuron of the neighbouring layers. Each neuron have its corresponding bias, and each connection have its own corresponding weight. These are respectively added and multiplied to the neurons input, and through backpropagation they are optimized to minimize some cost function to make better predictions. We will implement the FFNN with tensorflow sequential and look at both how different number of layers and neurons influence the bias-variance trade-off.

3 Results and discussion

Decision Tree

For the decission tree we perform the bias-variance trade-off for 100 bootstrap samples and different number of tree depths up to a max depth of 20 as seen in figure 1.

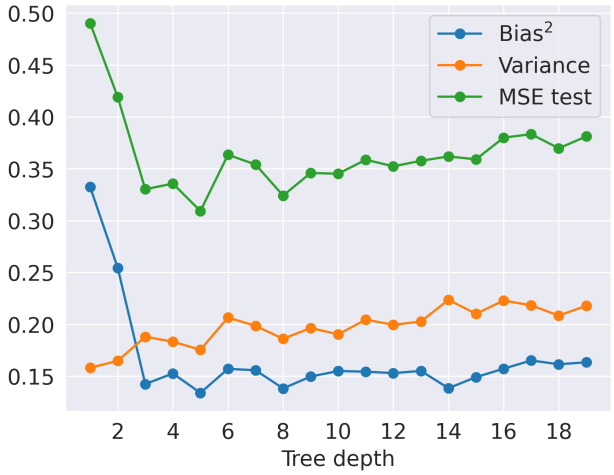

Figure 1: Bias-variance trade-off as a function of maximum tree depth for a decision tree using 100 bootstrap samples.

In figure 1 we see that an increasing tree depth results in a decrease in the bias at the same time as an increase in the variance. Both of these contributions results in the lowest mean squared error for a tree depth of 5 of 0.3. We see that also the bias starts increasing after this depth, indicating that we will not find a better prediction for a larger tree depth. This can be explained by overfitting as discussed earlier. If not the training is restricted it will perfectly classify the training data, removing the model's generalization and possibility to predict other data from the same source.

Random Forest

Below in figure 2 we see the bias-variance trade-off as a function of tree depth. We see 4 plots corresponding to different number of trees. We have again used 100 bootstrap samples.

There are small variations in the trade-off, and we mainly see a big drop in the MSE and bias when increasing the tree depth from 1 to 3. We notice that the bias is the main contributor to the MSE, also for larger tree depths. The variance stays small with only a small decrease by increasing the tree depth. We generally find the lower values of MSE from a decrease in bias for tree depths of 35 and above. This indicates that we may find even lower MSE by increasing the tree depth further. We notice almost no difference between different number of trees, but the overall lowest MSE is found by using 10 trees and a tree depth of 35. Compared to a single decision tree we see a much lower MSE of 0.15, which mainly comes from the much lower variance (0.05) since the biases have similar values (0.15). This is as expected from the averaging of several uncorrelated trees,

which as described earlier can greatly reduce the variance. This is together with the bootstrapping of data to each tree which following the central limit would give a smaller variance of the final averaged prediction

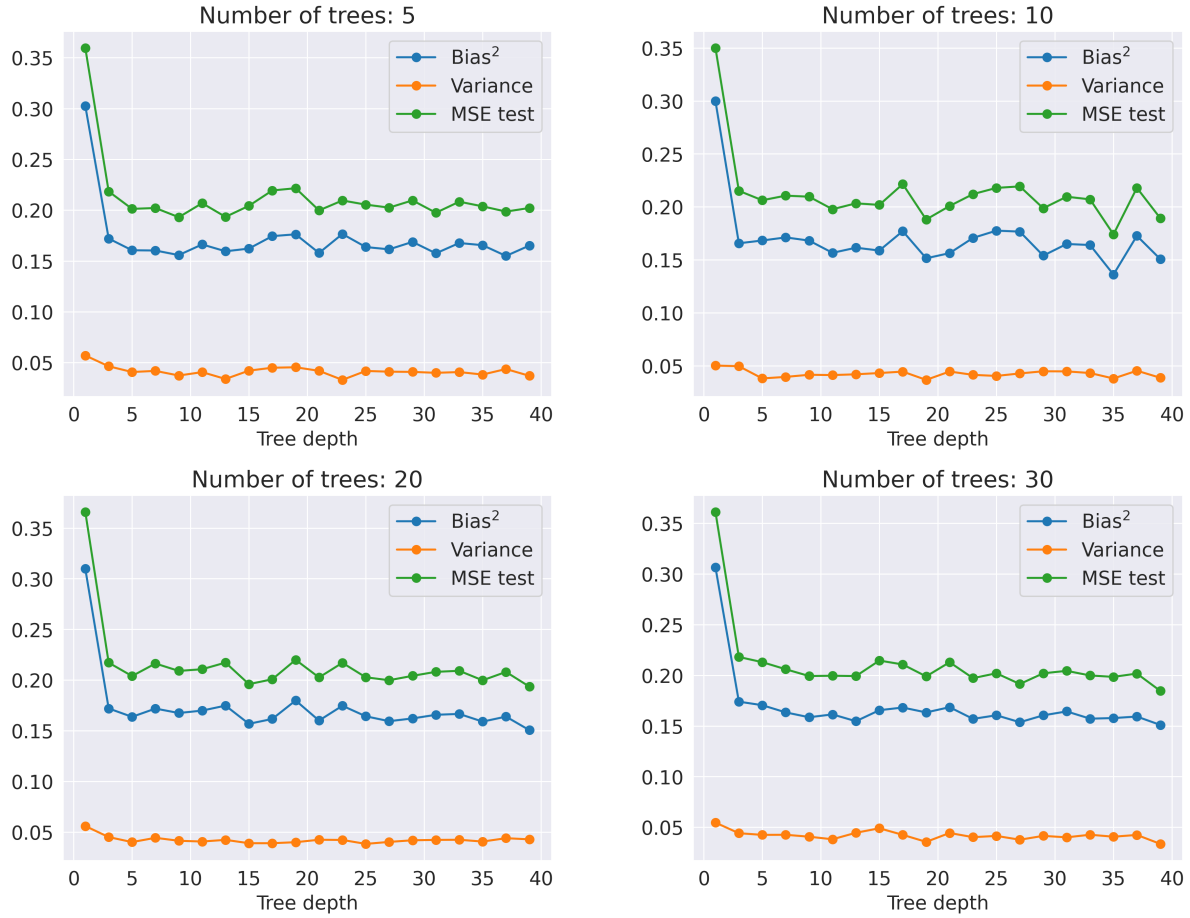

Figure 2: The bias-variance trade-off using 100 bootstrap samples as a function of tree depth up to a depth of 40 where every second tree depth is skipped. We see 4 plots corresponding to different number of trees in the forest.

Feed Forward Neural Network

In figure 3 we see the bias-variance trade-off using 100 bootstrap samples for different number of neurons per layer. We see 4 plots corresponding to 1, 2, 3 and 4 layers in the neural network.

The 4 plots in figure 3 all show quite varying results for different number of neurons, but we still notice a general trend. For all plots we see small variances between 0 and 1. We also see an increasing variance until around 10 neurons where a further increase generally equals a lower variance going towards 0. This shows that the bias is the main contributor to the MSE especially for a higher number of neurons. For 4 layers we see an indication of overfitting with a slight increase in the variance and MSE for higher number of neurons. In the other cases we see decreasing MSE for more neurons indicating that a further neuron increase could equal an even lower MSE. We see that the Neural Network have smaller MSE for certain number of neurons than both the decision

tree and the random forest. We see MSE at or below 0.15 for all the tested number of layers, and even MSE below 0.1 for 2 layers and 58 neurons, and 4 layers and 21 neurons.

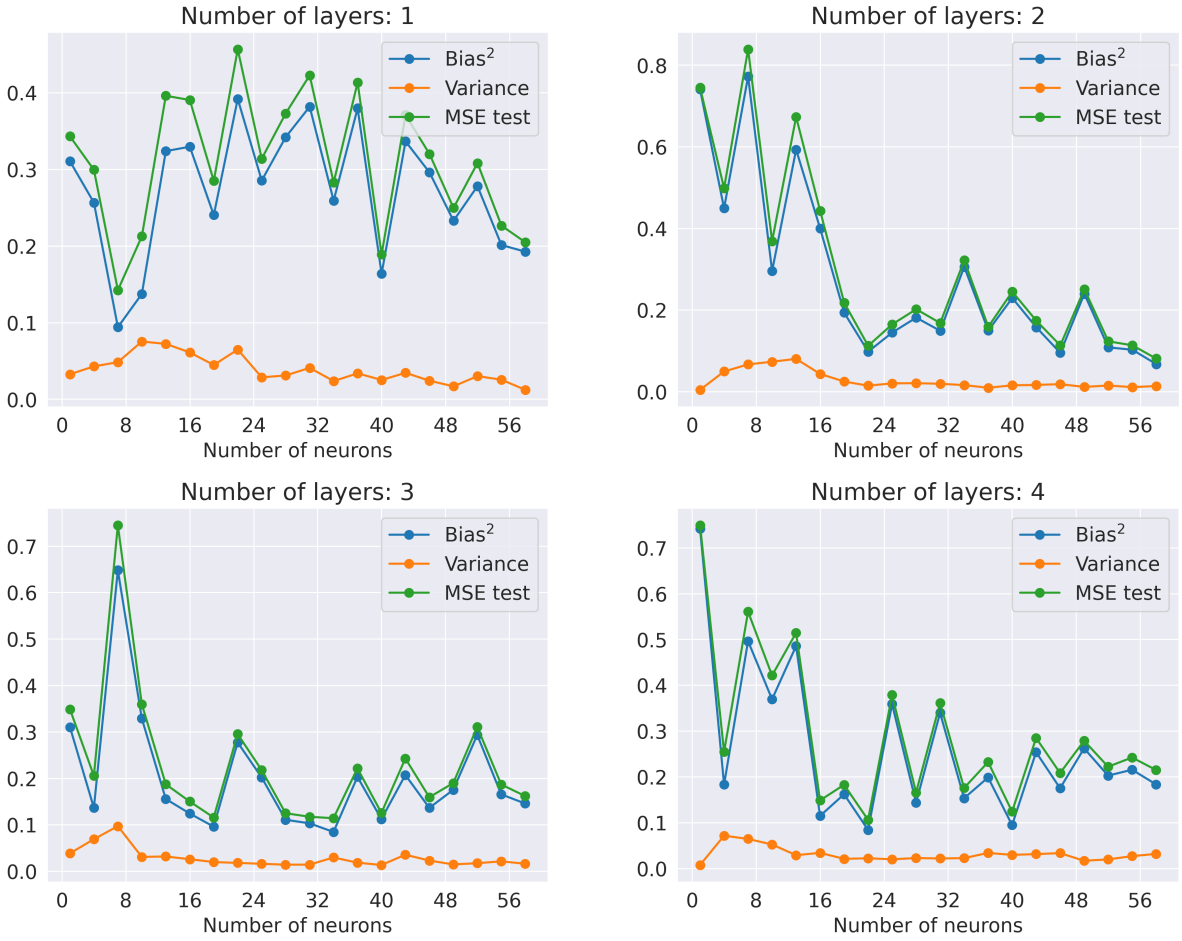

Figure 3: The bias-variance trade-off using 100 bootstrap samples as a function of neurons in each layer up to a total of 58 neurons where every third neuron is skipped. We see 4 plots corresponding to different number of layers in the neural network.

4 Conclusion

We have found that different methods and their different parameters equals very different bias-variance trade-offs. A single decision tree has shown the worst overall performance with a minimum MSE of 0.3. It had a more typical trade-off than the other methods with an increasing variance and decreasing bias for higher complexities. The random forest gave us very stable trade-offs with a small to no dependency on the number of trees. We saw no indication of overfitting. The bias was the main contributor to the MSE and its minimum of 0.15, and the variance stayed generally low. For the neural network we saw a quite varying trade-off with the bias being the main contributor to the MSE. We saw again no sign of overfitting except by using 4 layers with a slight increase in variance for higher number of neurons.

Our results may have been influenced by our dataset, which makes it together with the lower optimal complexities of our models interesting to make a further analysis of the trade-off for a larger and more advanced dataset.