The Bias-variance-tradeoff

Filip Severin von der Lippe

December 14, 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 tradeoff 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.

Bias-variance tradeoff

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 $\boldsymbol{X} = (\boldsymbol{X}_j, y_j)$ 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 $\boldsymbol{y} = f + \epsilon$ we rewrite the costfunction:

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

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

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

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

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

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 $\boldsymbol{y} = f + \boldsymbol{\epsilon}$.

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

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

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

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.