FYS-STK4155 - Applied Data Analysis and Machine Learning

Bias-Variance Tradeoff Analysis

Jouval Somer, Stig Patey, Celine Marie Solberg & Ole Petter Maugsten

December 2022

Abstract

In this report, the bias and variance tradeoff in three different machine learning methods was studied. The methods studied were linear regression, neural networks and decision trees. The data we were trying to fit was a noisy Gaussian curve. The best linear regression fit had a MSE of 0.009862, and the best neural network had a MSE of 0.01001. Both errors were about 0.01 which was the exact value of the variance of the irreducible noise in the data. This meant that the two models performed well. The best decision tree from the random forest gave an MSE of 0.01493. The model performed okay, but it did not beat the two other machine learning methods. Even though the linear regression model and the neural network performed equally well, the former had a significantly lower complexity than the latter.

1 Introduction and Problem Statement

When choosing a machine learning method to solve a problem, there are several parameters that needs to be predefined. The architecture or complexity of the model is one of the first things one usually decide upon. For some problems, a simple model might provide the best solution. For other problems, a more complex model might be necessary to catch non-linearities in the data. So how do we find the best model for our specific problem? We need a way of telling whether the model is a good fit or a bad fit to the dataset at hand.

For regression problems, the mean squared error is a common estimate of the model's performance. One desires the lowest mean squared error possible. It can be shown that this error can be broken into a sum of the model's bias, the model's variance and the data's variance. In order to minimise the error, i.e. find the best suited model, one can study the trade off between the model's bias and variance. In this text, we will study the bias and variance trade off in terms of model complexity of three machine learning methods. These methods are linear regression, neural networks and decision trees.

2 Theory

Given a dataset $\{x_i, y_i\}, i = 1, 2, ..., n$ with n datapoints, and a machine learning model $\{\tilde{y}_i, x_i\}$, where $\tilde{y}_i$ is the predicted value based on the input value x_i, we define the *mean squared error* (MSE) of the model as

$$\text{MSE} = \frac{1}{n}\sum_{i=1}^{n}(y_i - \tilde{y}_i)^2. \tag{1}$$

This error can be broken into a sum of the model's *bias*, the model's *variance* and the data's variance σ^2

$$\text{MSE} = \text{Bias}^2 + \text{Variance} + \sigma^2. \tag{2}$$

The equation is derived in the appendix.

The σ-error comes from the noise in the dataset and is therefore inherently irreducible. The bias and variance are defined as

$$\text{Bias} = (y - E[\tilde{y}]) \tag{3}$$

$$\text{Variance} = E[(E[\tilde{y}] - \tilde{y})^2]. \tag{4}$$

E denotes the expectation value of some variable. The bias tells us how biased a model is to certain values. If the bias is high, we say the model is *underfitted*. The variance tells us how much the model fluctuates. If the variance is high, we say the model is *overfitted*. The goal is to find a balance between the bias and the variance.

3 Method

To study the machine learning methods, we define a dataset $\{x_i, y_i\}, i = 1, 2, ..., n$ with $n = 200$ datapoints and $x_i \in [0, 1]$. The x-axis is divided into n equally spaced points between 0 and 1. The corresponding y-values are given by the bell curve

$$\{y_i\} = e^{\left(-\frac{(x_i-0.5)^2}{0.1^2}\right)} + \epsilon_i \tag{5}$$

where ϵ is gaussian noise with mean 0 and standard deviation $\sigma = 0.1$. The generated dataset is displayed in figure 1.
The machine learning methods we study in this text are linear regression, neural networks and decision trees.

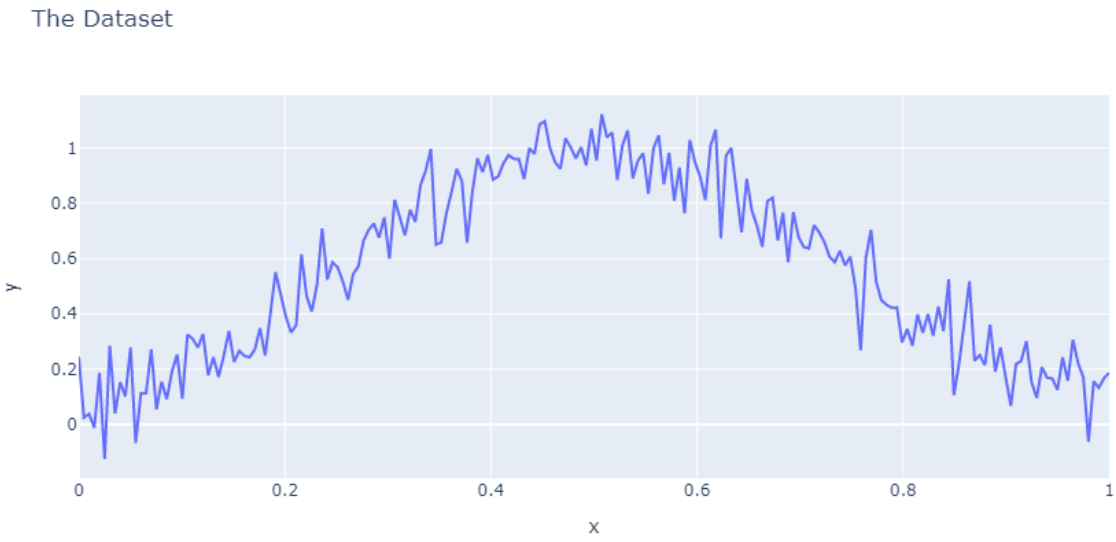

Figure 1: The figure shows the dataset which we will try to model. It is a gaussian curve with random noise.

For the *linear regression*, we made design matrices X with different polynomial terms $[x, x^2, x^3, ..., x^n]$. Since the data is not centered, we let scikit learn calculate the intercept for us by setting the parameter *fit intercept* = *True* in scikit learn's linear regression method. The model was then made by finding the optimal weights for each term through scikit learn's ordinary least squares (OLS) method. Polynomials of order 1 to 18 was tried. The training data was bootstrapped 100 times. The MSE, bias and variance was calculated for each design matrix, i.e. each polynomial fit. It allowed us to see how the error changed with the model's complexity.

For the *neural networks*, different architectures were tried. All networks had one input layer and one output layer, in addition to three hidden layers. The complexity of the network was altered by varying the number of nodes. For each neural network, the hidden layers had the same amount of nodes. The learning rate was fixed to 0.01 which is a common choice [Brownlee 2022]. The activation function was set to the ReLU function $max(0, x)$ which should well illustrate both under- and over-fitting. All networks used ADAM for optimisation. The MSE, bias and variance was calculated for each neural network.

For the *decision tree*-method, we grew random forests with 100 trees by bootstrapping the train dataset for each tree. The splits were made to maximally reduce the MSE. For each forest we varied the depth of the trees from 1 to 8. From each forest, the tree with the lowest MSE was chosen as the best fit. The MSE, bias and variance was calculated for each of these trees.

4 Results

4.1 Linear Regression

Polynomials of varying order were fitted to the dataset. For each model, the MSE, bias and variance were calculated. The results are shown in figure 2.

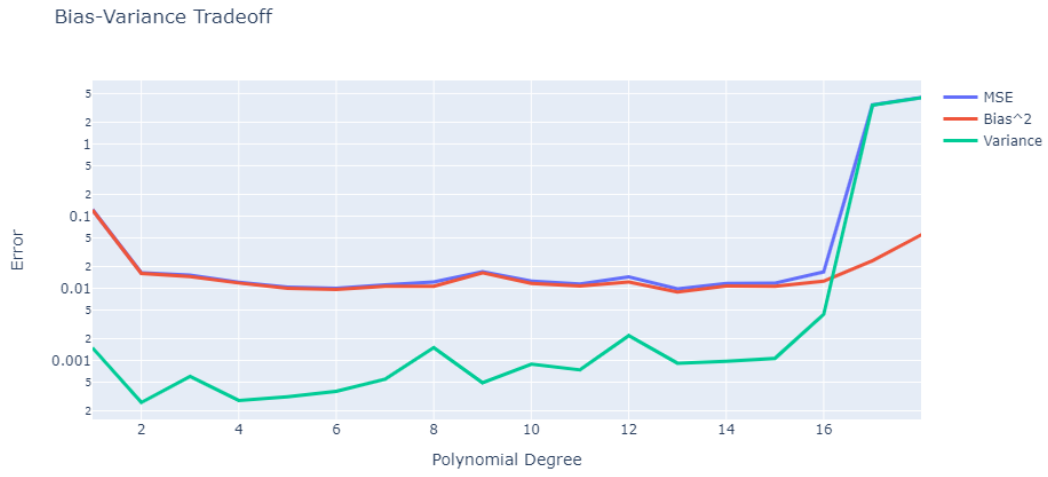

Figure 2: The figure shows the MSE, bias and variance of fitted polynomials. The polynomial of degree 5 had the lowest MSE of 0.009862

The polynomial of degree 5 had the lowest MSE of 0.009862 and is shown in figure 3.

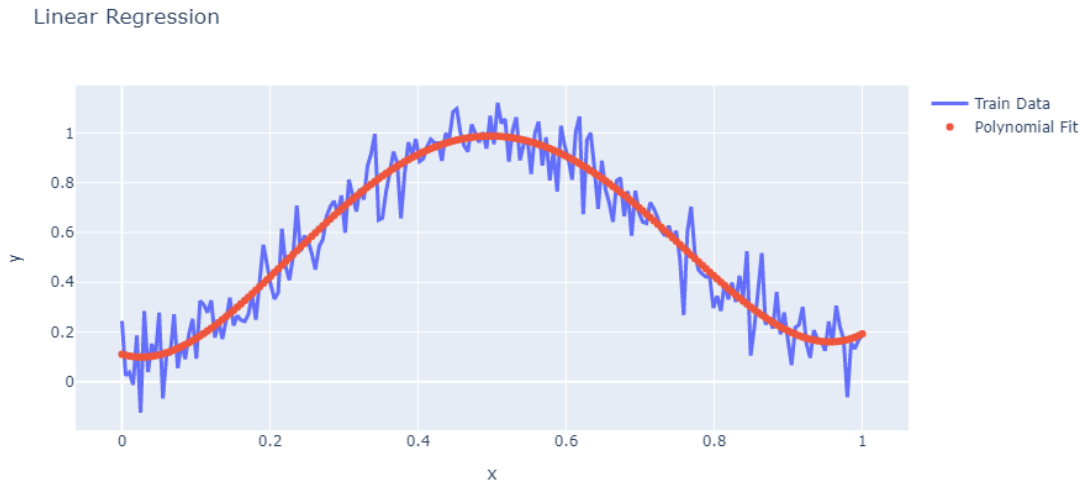

Figure 3: The figure shows the polynomial fit of degree 5.

The result is a polynomial fit that looks like a Gaussian curve, just like we desired.

4.2 Neural Networks

Neural networks with varying number of nodes were trained to the dataset. Three hidden layers were used for every network. For each model, the MSE, bias and variance was calculated. The results are shown in figure 4.

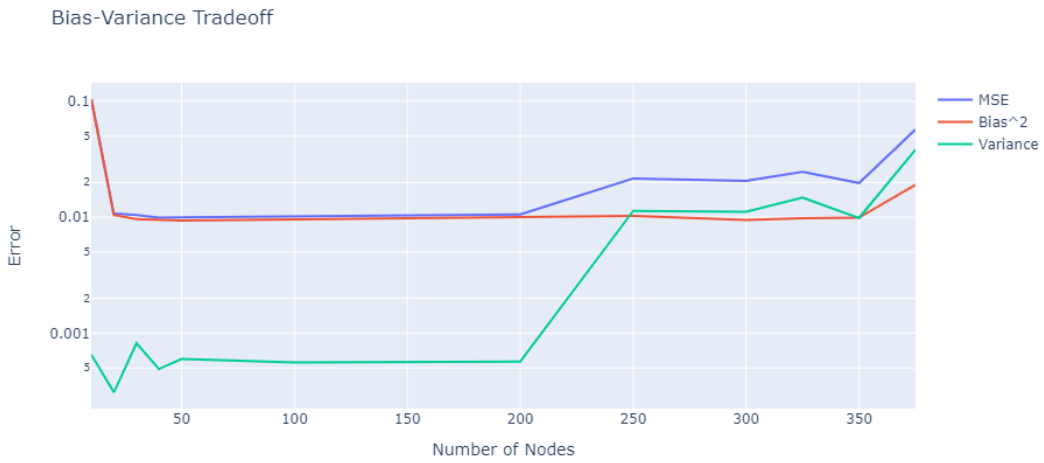

Figure 4: The figure shows the MSE, bias and variance of trained neural networks. The neural network with 40 nodes had the lowest MSE of 0.01001

Figure 4 shows that the bias decreases and the variance increases with the model's complexity.
The neural network with 40 nodes in the hidden layers had the lowest MSE of 0.1001 and is shown in figure 5.

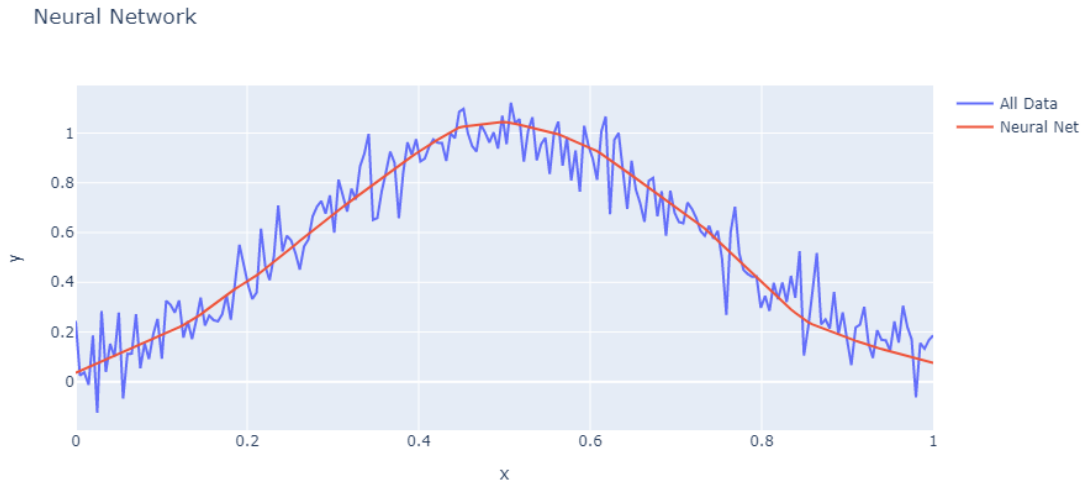

Figure 5: The figure shows the prediction of the neural network with 40 nodes in the hidden layers.

Figure 5 shows that the neural network's guess on a model is close to a Gaussian curve.

4.3 Decision Trees

Decision trees from random forests with varying depth were fitted to the dataset. For each model, the MSE, bias and variance was calculated. The results are shown in figure 6.

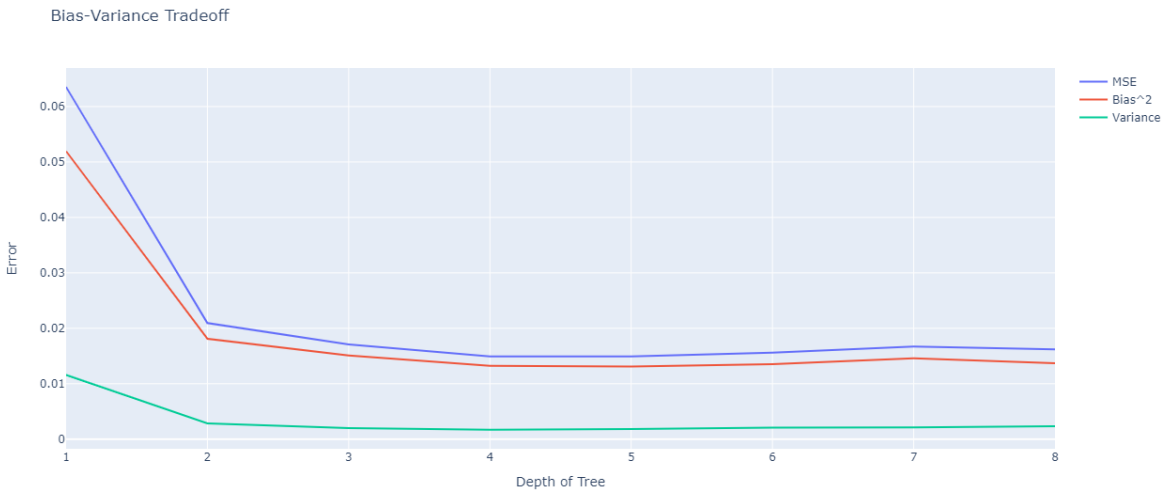

Figure 6: The figure shows the MSE, bias and variance of fitted decision trees from random forests. The tree with depth 5 had the lowest MSE of 0.01493.

Figure 6 shows that the model was never overfitted. The decision tree with depth 5 had the lowest MSE of 0.01493 and is shown in figure 7.

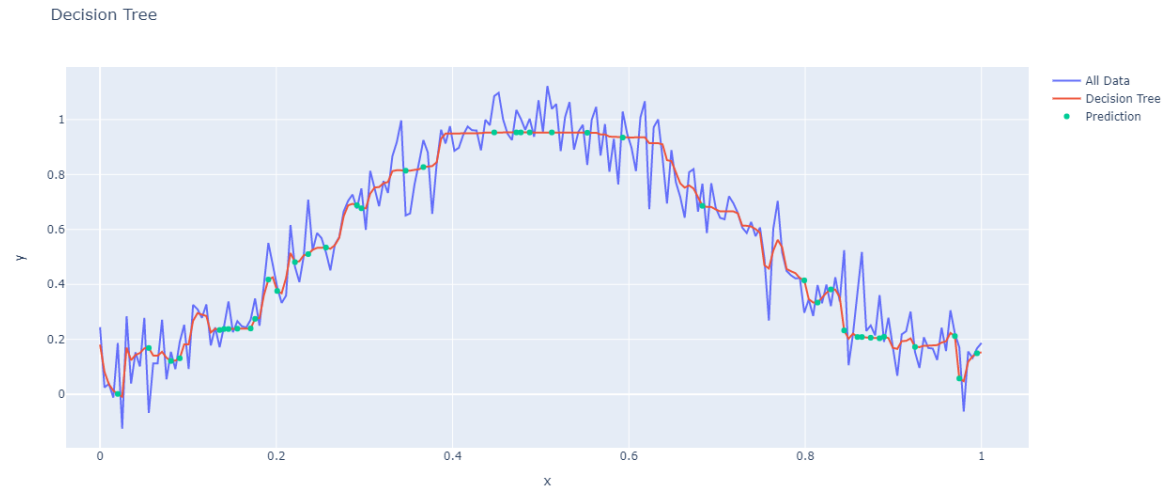

Figure 7: The figure shows the decision tree model of depth 5 along with all the data. It also shows the predictions from the test set.

Figure 7 shows that the decision tree model looks somewhat like a quantified Gaussian curve. The model looks okay, but is not nearly as good as the linear regression or neural network model.

The architecture of the decision tree of depth 5 is shown in figure 8.

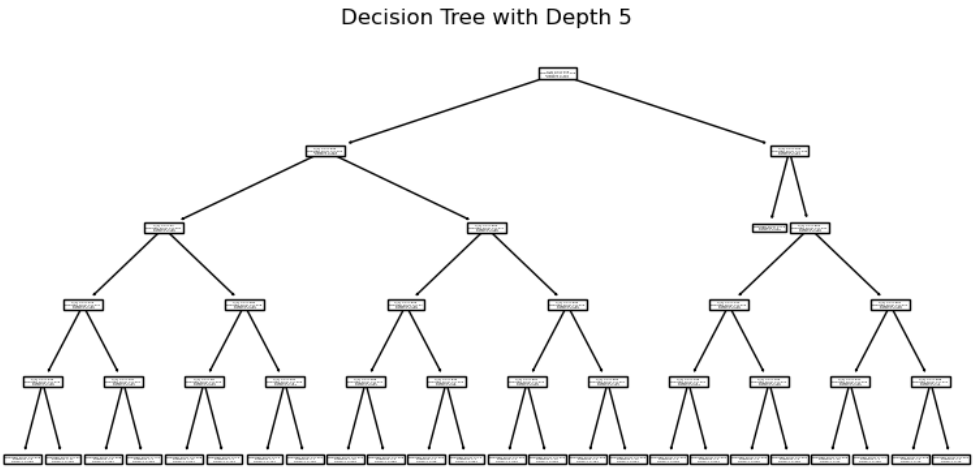

Figure 8: The figure shows the architecture of the decision tree model of depth 5.

5 Discussion

The first models that were made were polynomial fits through the ordinary least squares method. From figure 2, we see that the bias rapidly decreases from polynomials of degree 1 to 2. This is because we move from fitting a line to fitting a curve which better catches the non-linearity of the data. From here on out, as we increase the order of the fitted polynomial, the variance goes up. This is because higher order polynomials are able to bend more than those of lower order. For polynomials of degree higher than 16, we see that the variance skyrockets bringing the MSE with it. If the polynomial model is of high order, it might try to interpolate certain datapoints. This is a clear indication that the model is overfitted. The model with the lowest MSE was a polynomial of order 5. The MSE was 0.009862. We remind ourselves that the variance of the noise was $\sigma^2 = 0.01$. We therefore do not expect to find a model with lower MSE than that value. The best polynomial fit had approximately the same error and is considered a good model.

The second type of modelling we did was with neural networks. From figure 4, we see that when the number of nodes is too low, the network is not able to catch the non-linear relation between x and y. The use of another non-linear activation function could have helped with this problem. Networks with a number of nodes between 30 and 200 did an excellent job of modelling the dataset as they returned an MSE of about 0.01. Networks with a higher number of nodes performed rather bad as the variance of the models increased dramatically bringing the MSE with them. It is important to note here that the networks converged to a solution and are not just under-trained. The high variance suggests that these neural networks were overfitted to the data. The best neural network had 40 nodes in each hidden layer and returned an MSE of 0.01001. This is about the same error as the best polynomial fit had. They are both considered good models. Even though the two models performed equally well, the neural network has a higher complexity and in general acts more like a black box compared to the easy-to-implement linear regression model. The latter would therefore be of preference for similar regression problems.

The third method of modelling we did was with decision trees. We see from figure 6, that the bias dropped a lot in value as the depth of the tree went from 1 to 2. This is because the tree allows predictions on more levels as the depth increases. As the depth continued to increase, the variance never skyrocketed the same way the polynomial fits did. Sklearn's random forest regressor makes splits either at random or accordingly to a criterion like maximally decreasing the MSE. This will most likely split the data into subsets of unequal size. The tree is then destined to have fewer leaves than 2 to the power of the depth. This will keep the variance down. Another factor that dampens the variance is that the decision tree models do not take extreme values between the datapoints of the training set the same way polynomial fits might do. For all the trees we made, none of them managed to model the data nearly as good as the polynomial models or the neural networks did. The best tree had a depth of 5 and returned a MSE of 0.01493. It is not a terrible model, but it is worse than the other methods in this text.

6 Conclusion

The lowest MSE was equal to 0.009862 and was obtained by a linear regression model with a polynomial fit of degree 5. The value was very close to the irreducible variance of the noise $\sigma^2 = 0.01$. A neural network consisting of three layers with 40 nodes in each layer gave an MSE of 0.01001 which was again approximately the same value as the natural variance of the data. Both models were considered good models as they gave a low MSE and an output that looked quite similar to the function we wanted to fit. The lowest MSE obtained by a decision tree from a random forest was 0.01493. The model's MSE was an indication that the model worked, but the performance was poor compared to the other two methods.

For all the methods we implemented, we saw a tradeoff between the bias and the variance of the models. We also saw that by balancing the bias and variance correctly, we found the best models.

Bibliography

Brownlee, Jason (2022). *How to Configure the Learning Rate When Training Deep Learning Neural Networks*. URL: https://machinelearningmastery.com/learning-rate-for-deep-learning-neural-networks/ (visited on Dec. 18, 2022).

Appendix

Bias-Variance

The mean squared error $E[(z-\tilde{z})^2]$ can be decomposed into the model's squared bias $(E[\tilde{z}]-f)^2$, variance $E[(\tilde{z}-E[\tilde{z}])^2]$, and the variance σ^2 of the noise.

$$\begin{aligned}
E[(z-\tilde{z})^2] &= E[(f+\epsilon+\tilde{z})^2] \\
&= E[f^2+\epsilon^2+\tilde{z}^2+2f\epsilon-2f\tilde{z}-2\epsilon\tilde{z}] \\
&= E[(f-\tilde{z})^2+\epsilon^2+2\epsilon(f-\tilde{z})] \\
&= E[(f-\tilde{z})^2]+E[\epsilon^2]+2E[\epsilon]E[(f-\tilde{z})] \\
&= E[(f-\tilde{z})^2]+\sigma^2 \\
&= E[((f-E[\tilde{z}])-(\tilde{z}-E[\tilde{z}]))^2]+\sigma^2 \\
&= E[(E[\tilde{z}]-f)^2+(\tilde{z}-E[\tilde{z}])^2-2(f-E[\tilde{z}])(\tilde{z}-E[\tilde{z}])]+\sigma^2 \\
&= (E[\tilde{z}]-f)^2+E[(\tilde{z}-E[\tilde{z}])^2]-2(f-E[\tilde{z}])(E[\tilde{z}]-E[\tilde{z}])+\sigma^2 \\
&= (E[\tilde{z}]-f)^2+E[(\tilde{z}-E[\tilde{z}])^2]+\sigma^2 \\
&= (Bias[\tilde{z}])^2+Var[\tilde{z}]+\sigma^2
\end{aligned}$$