FYS-STK4155: Applied Data Analysis and Machine Learning

Study of Bias-Variance tradeoff for Regression Problems

Mohamad Mahmoud & Aleksandar Davidov
GitHub - click here -

December 23, 2021

Abstract

In this project, we have performed analysis of the bias-variance tradeoff for OLS, Ridge and LASSO, Neural Networks and Gradient Boost regression. Bootstrap was used for resampling. What we realised was that the models can achieve both low bias and variance but the method who did it most consistently was the Gradient Boosting algorithm.

1 Introduction

The study of bias-variance tradeoff is is especially important in statistics and Machine Learning where we want to find the best method to use for a specific problem given a set of data that will perform well beyond the training set of our data. High bias can cause the algorithm to miss relations between the inputs and the outputs. While high variance is associated with the modeling the random noise in the input data. In this project, we are going to study the bias-variance tradeoff for the OLS, Ridge and LASSO regression models, Feed Forward Neural Networks and Gradient Boosting. This is hence then a continuation of *[1]* where we explaned the theory behind on linear regression as well as bootstrap and bias-variance tradeoff. For the theory behind FeedForward Neural Networks, read *[2]*.

2 Theory

Gradient Boosting is a machine learning algorithm which gives a prediction based on the ensemble of an ensemble of weak classification/regression models. By weak classification models, we mean models which on their own, only predict slightly better than one would by random guessing. This is done via a method via a series of iterations. We start with the cost-function,

$$\mathcal{C} = \sum_{i=0}^{n-1} \mathcal{L}(y_i, f(x_i))^2$$

which we want to minimize. This is done by optimizing

$$\hat{f} = \underset{f}{\operatorname{argmin}}\, \mathcal{C}.$$

We can write

$$f_M(x) = \sum_{m=0}^{M} h_m(x)$$

where $h_m(x)$ is a real function that defines our final function $f_M(x)$. A common approach for he Gradient Boosting algorithm is steepest decent approach. In this approach, we approximate the

function $h_m(x) = -\rho_m g_m(x)$, where $\rho_m \in \mathbf{R}$ and $g_m(x)$ the gradient, which is defined as

$$g_m(x_i) = \left[\frac{\partial \mathcal{L}(y_i, f(x_i))}{\partial f(x_i)}\right]_{f(x_i)=f_{m-1}(x_i)}.$$

With the new gradient we can update our function $f_m(x) = f_{m-1}(x) - \rho_m g_m(x)$. Using the above squared-error function we see that the gradient is $g_m(x_i) = -2(y_i - f(x_i))$. Choosing $f_0(x) = 0$ we obtain $g_m(x) = -2y_i$ and inserting this into the minimization problem for the cost function we have

$$(\rho_1) = \underset{\rho}{\operatorname{argmin}} \sum_{i=0}^{n-1} (y_i + 2\rho y_i)^2.$$

However, steepest descent is not much used as a decent approach for Gradient Boosting, since it only optimizes the function f at a fixed set of n points. Therefore, we do not learn a function that can generalize. However, we can modify the algorithm by fitting a weak learner to approximate the negative gradient signal. Suppose we use the Mean Squared Error Cost Function

$$\mathcal{C} = \sum_{i=0}^{n-1} (y_i - f(x_i))^2,$$

then proceeding in an iterative fashion, our algorithm becomes

Algorithm 1 Gradient Boosting

procedure GradBoost
Initialize our estimate $f_0(x)$.
For $m = 1 : M$, we have
- a. compute the negative gradient vector $\boldsymbol{u}_m = -\partial C(\boldsymbol{y}, \boldsymbol{f})/\partial \boldsymbol{f}(\boldsymbol{x})$ at $f(\boldsymbol{x}) = f_{m-1}(x)$;
- b. fit the so-called base-learner to the negative gradient $h_m(u_m, x)$;
- c. update the estimate $f_m(x) = f_{m-1}(x) + h_m(u_m, x)$;

The final estimate is then $f_M(x) = \sum_{m=1}^{M} h_m(u_m, x)$.
end procedure

3 Results

As our dataset, we will use the Franke data from *[1]*. We will set $\mu = 0$ and $\sigma = 0.2$ for our noise. We will also work with a set precision of 0.05. We used the **scikit-learn** package for the implementation of our Machine Learning algorithms. All the code is available and can be found on GitHub.

Starting with the Ordinary Least Squares, we get the following

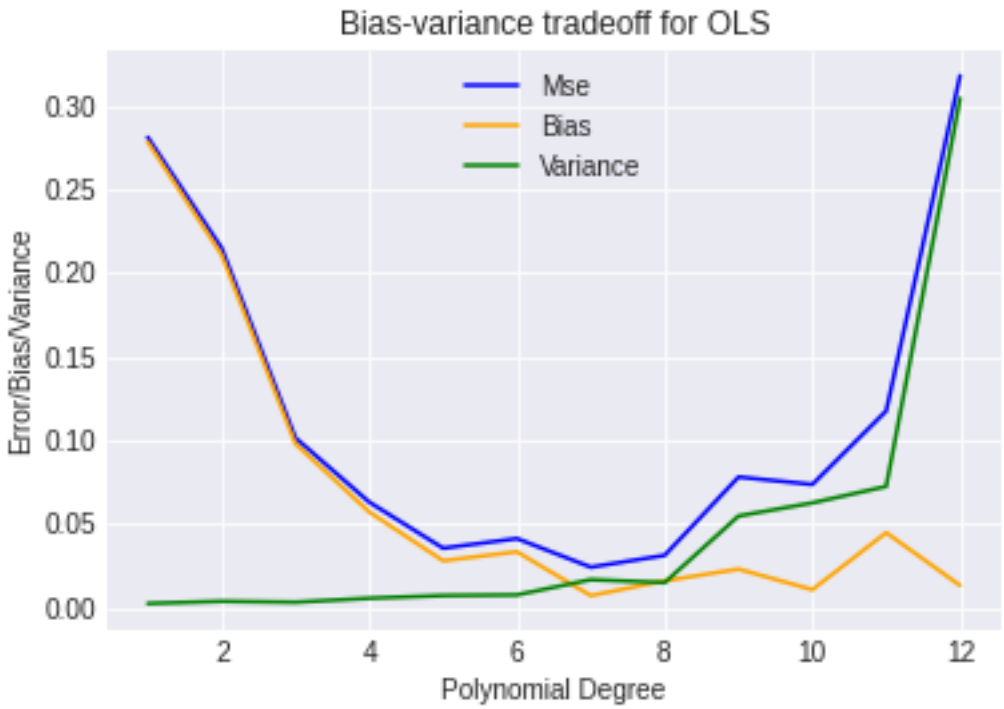

Figure 1: Bias-variance tradeoff for the OLS regression with a Polynomial Degree of 12.

Figure 1 shows the bias, variance and the Mean Square Error for OLS for the Franke data up to polynomial degree 12. The x-axis gives us the polynomial degree while the y-axis gives us the bias, variance and Mean Square Error. We can see that a polynomial degree in between 6 and 8 (7 gives the best) gives us the smallest error. Further, it seems that after degree 8, that the bias decreases slightly, while the variance increases drastically at this point. This means that overfitting is at a greater risk for OLS than underfitting.

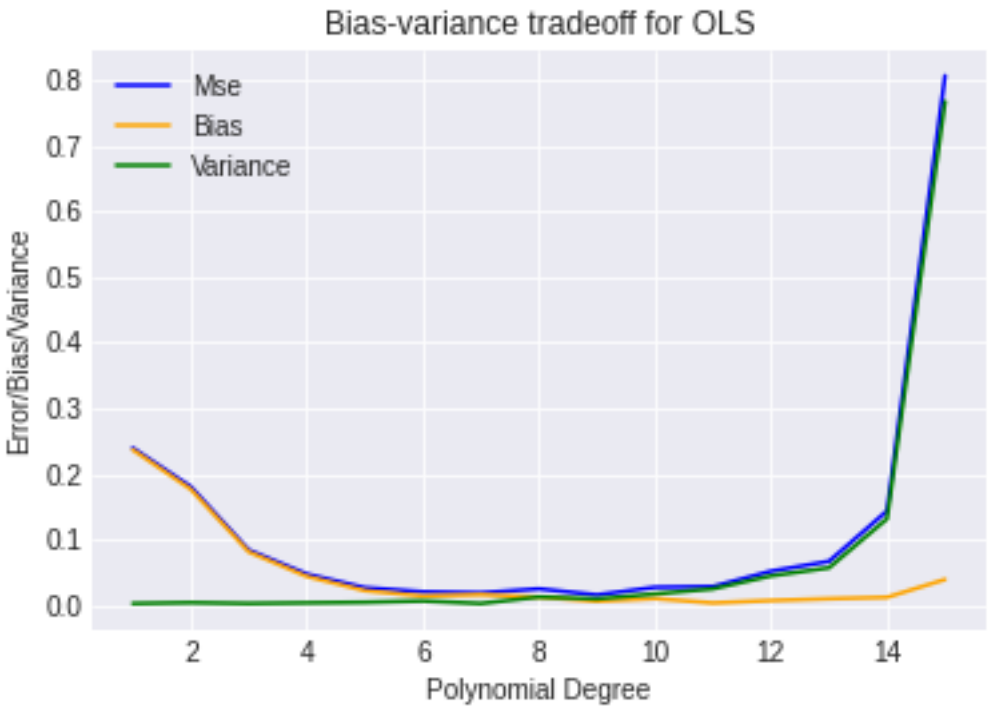

Figure 2: Bias-variance tradeoff for the OLS regression with a Polynomial Degree of 12.

Figure 2 shows the same type of plot as the one above but to polynomial degree 15. Here the drastic increase in variance is even more noticeable.

Continuing now to Ridge and LASSO regression, we get

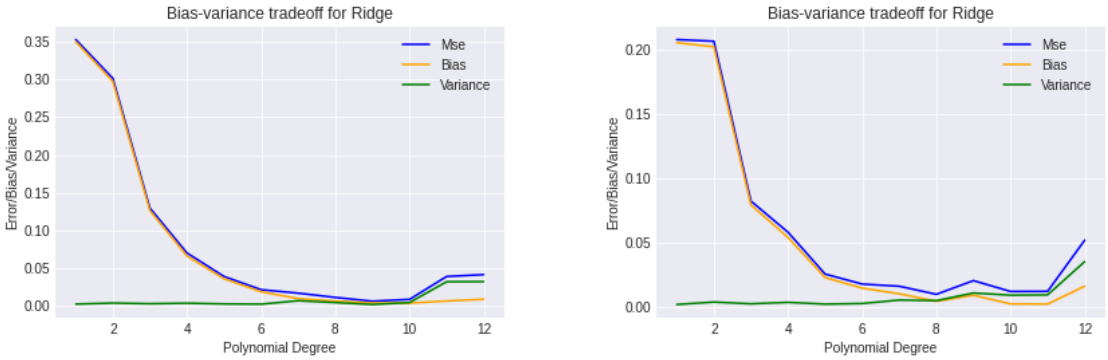

Figure 3: Bias-variance tradeoff for the Ridge regression with a Polynomial Degree of 12. Left plot: $\alpha = 0.1$, Right plot: $\alpha = 0.01$

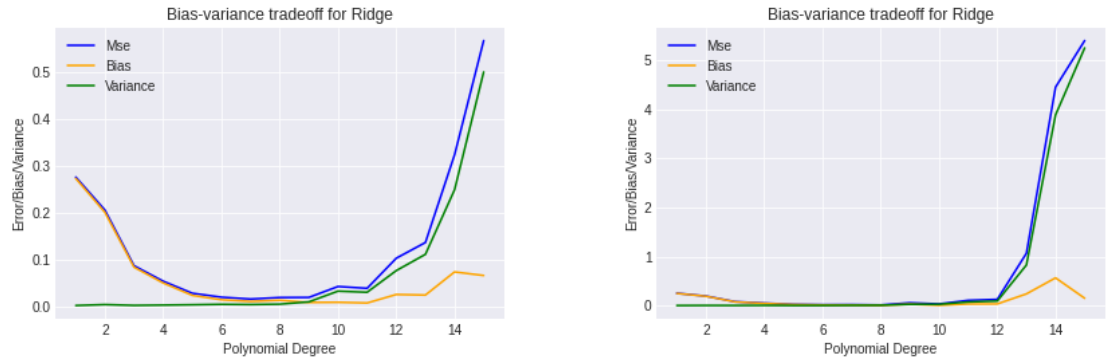

Figure 4: Bias-variance tradeoff for the Ridge regression with a Polynomial Degree of 15. Left plot: $\alpha = 1$, Right plot: $\alpha = 0.001$

Figure 3 shows the same plot as the ones above but for Ridge regression. Both go up to degree 12 but the plot on the left is with $\alpha = 0.1$ while the one on the right has $\alpha = 0.01$. We see that from both these plots, the optimal model is when the polynomial degree is somewhere in between 7 and 9. Figure 4 shows the bias-variance tradeoff for Ridge but up to polynomial degree 15 and the penalty $\alpha = 1$ and $\alpha = 0.001$. We still see that the optimal model is with a polynomial degree between $7 - 9$ but however here it is more obvious that a greater value of α gives a slower increase in variance. However, aside from this, the relationship between the bias and the variance are more or less the same for different values of α.

The final of the Linear Regression models, LASSO is shown below.

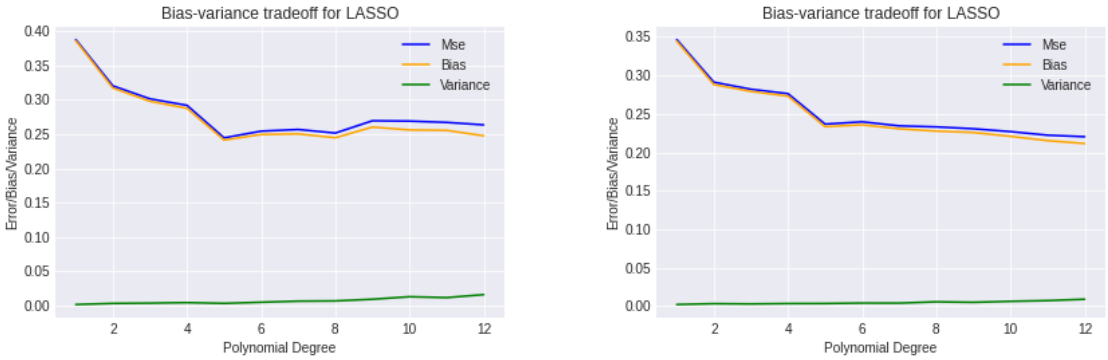

Figure 5: Bias-variance tradeoff for the LASSO regression with a Polynomial Degree of 12.
Left plot: $\alpha = 0.1$, Right plot: $\alpha = 0.01$

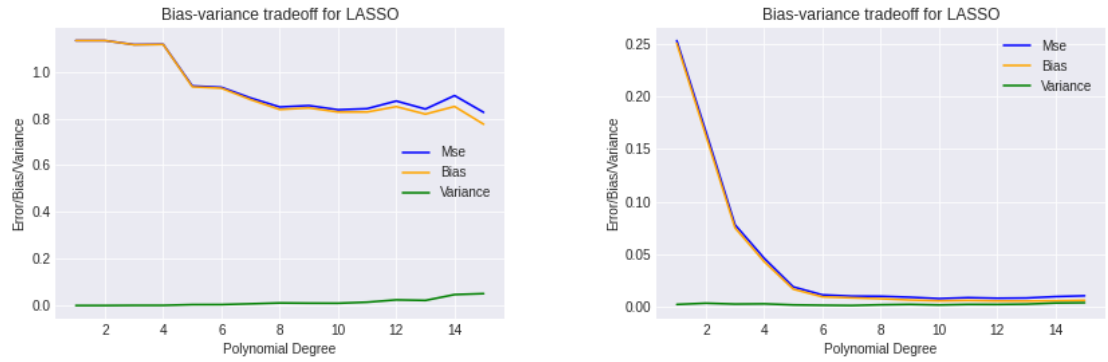

Figure 6: Bias-variance tradeoff for the LASSO regression with a Polynomial Degree of 15.
Left plot: $\alpha = 1$, Right plot: $\alpha = 0.001$

Figure 5 and 6 shows the plots for the final Linear Regression model, LASSO. They're the same as the previous ones in the sense that the polynomial degree goes to 12 on the first two, and 15 on the last two. The α values are also the same as for the Ridge plots. The first noticeable difference between LASSO and the previous Linear Regression models is that the variance has drastically reduced. We see that the variance increases slightly as the polynomial degrees increase. Therefore, the error here is mostly due to the bias, which in this case (LASSO) is significantly higher than what it was on the OLS and Ridge models. If we look at the plot where $\alpha = 1$, we see that it's the case with the highest error, while the case when $\alpha = 0.001$ gives smaller and smaller error as the polynomial degrees increase. For this case, as the degrees increase, we see that it reaches a point where both bias and variance, and as a consequence, the error, are low.

If we now go over to the Neural Network, which is implemented via the **MLPRegressor** in scikit, we see

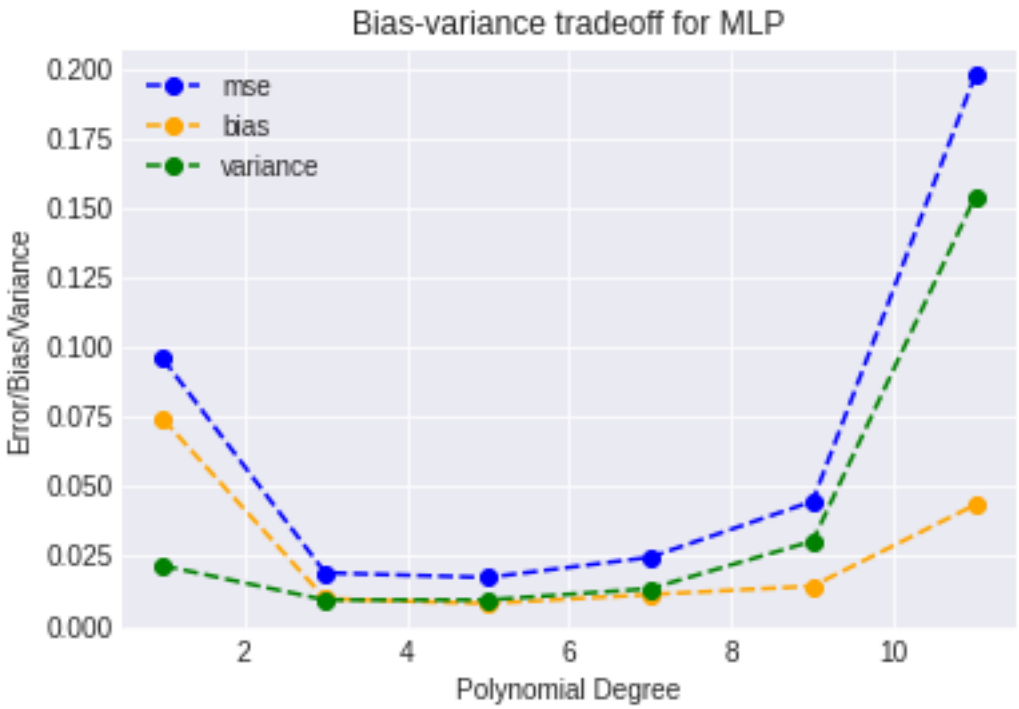

Figure 7: Bias-variance tradeoff for the MLP with different number of hidden layers and 10 number of nodes per layer.

Figure 7 shows the bias, variance and the Mean Square Error for a MLP trained on the Franke data with 10 nodes per hidden layer and different number of hidden layers. The number of hidden layers are given on the x-axis while the error, bias and variance are given on the y-axis. What we see from the plot is that as the complexity increases over 9 hidden layers, so does the variance significantly. One hidden layer seems to be too general, but as we increase to two, we see that the bias drops significantly.

The results here go hand in hand with what we know from Neural Network models. And that is that overfitting becomes more prominent as we increase the complexity, *fig. 11.6 p. 402 in [3]*. Since the variance is increasing faster than the bias, here also we're at a greater risk of overfitting.

Now if we go over to Gradient Boosting, we get the following plots

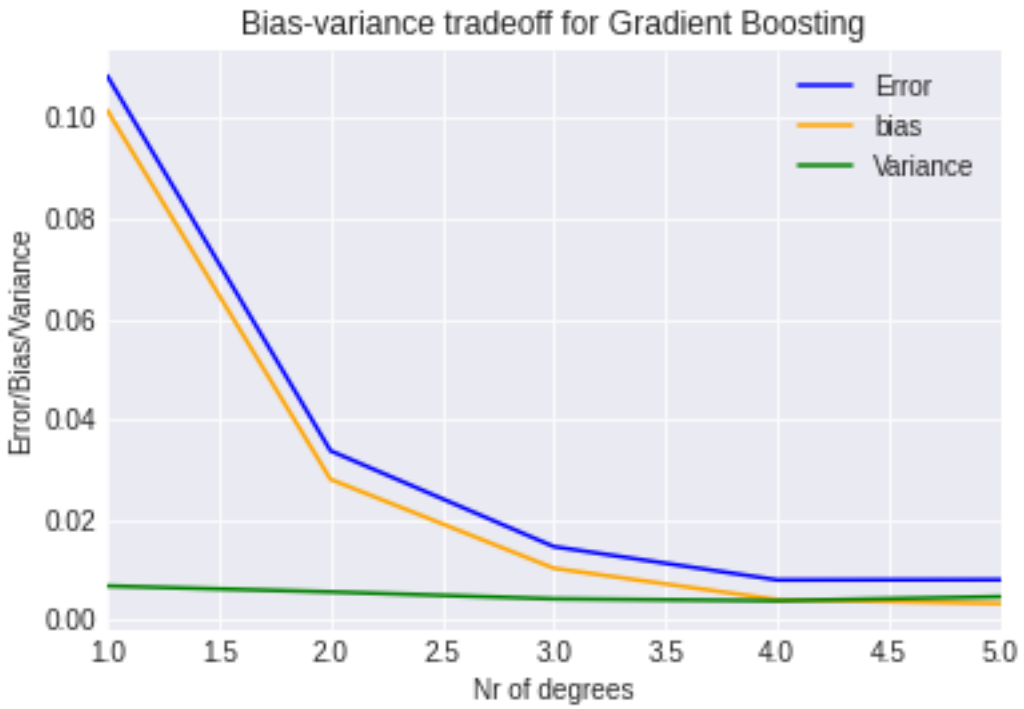

Figure 8: Bias-variance tradeoff for GradientBoosting learning rate equal to 1.0 and 100 boosting steps.

Figure 8 shows the bias, variance and the Mean Square Error for a Gradient Boosting algorithm trained on the Franke data with a set learning-rate of 1.0 and $n = 100$ boosting steps (which is the default). On the x-axis, we have the "number of degrees" which show the maximum depth of the individual regression estimators. The maximum depth limits the number of nodes in the tree.

From the plot we notice that the variance varies very little and is low. As mentioned, Boosting combines multiple "weak", high bias models in an ensemble that has lower bias than the individual models. This makes Gradient Boosting fairly robust to overfitting, we should be able to increase the boosting steps, n, without loss in performance.

Further, we see that the bias keeps decreasing all the way to the degree = 5.0 and at which, it is lower than the variance.

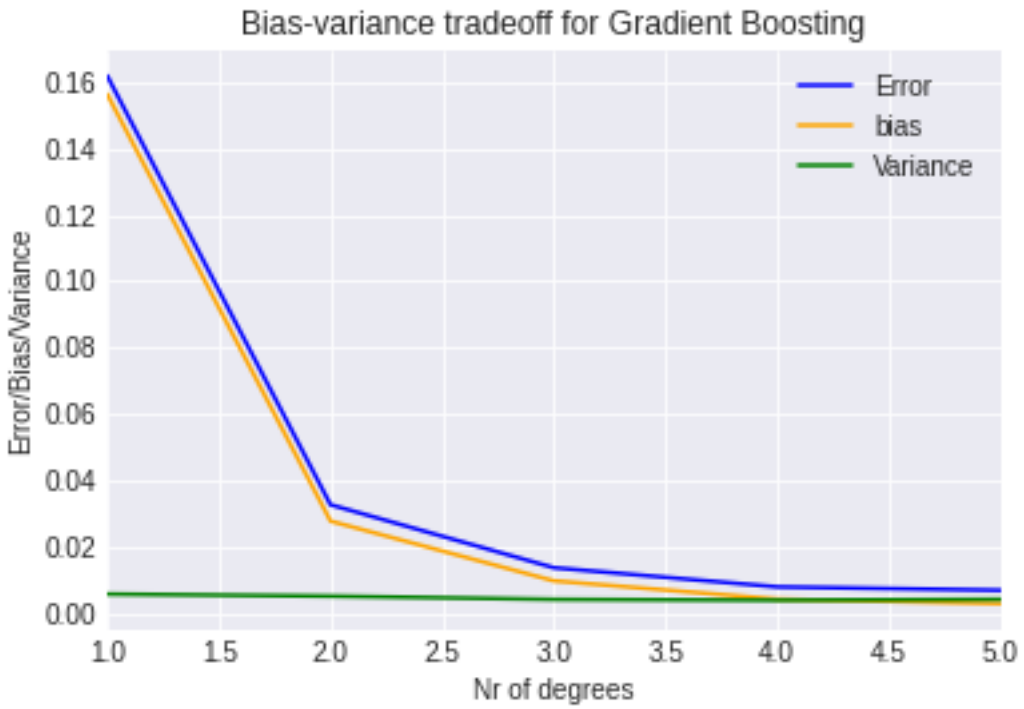

Figure 9: Bias-variance tradeoff for GradientBoosting learning rate equal to 0.1 and 100 boosting steps.

Firure 9 is similar to the one above but this time the learning-rate is is lowered to 0.1. We see that when the number of degrees is between 4.0 and 4.5, the bias and the variance have a similar value.

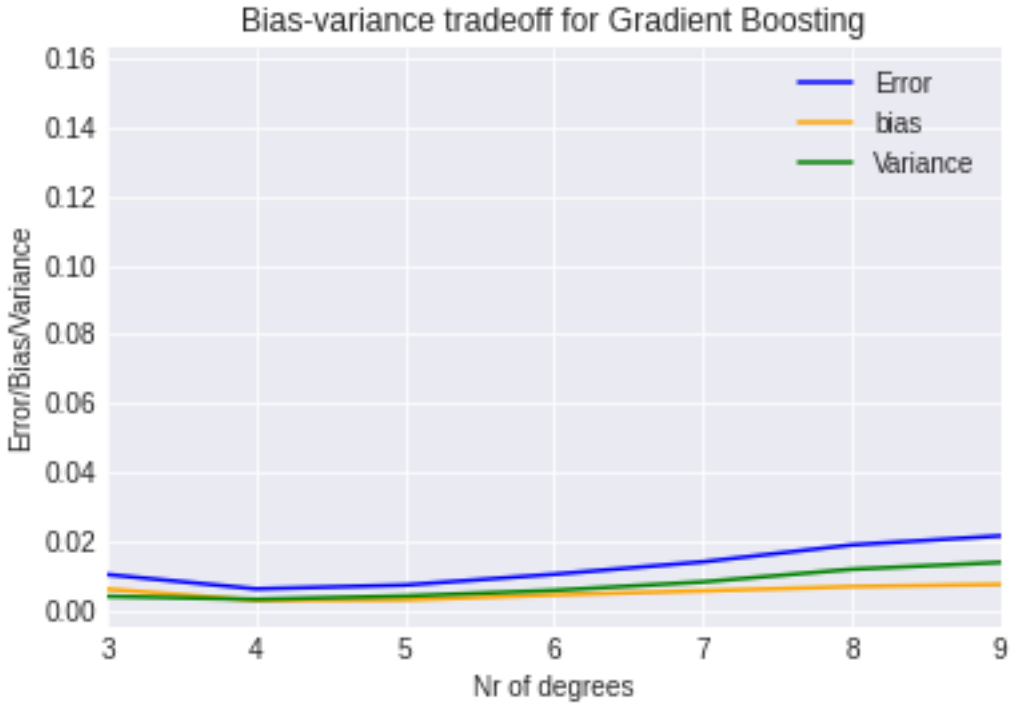

Figure 10: Bias-variance tradeoff for the MLP with different number of hidden layers and 10 number of nodes per layer.

Figure 10 shows the same learning-rate as the previous. But here the number of degrees has increased and $n = 200$. We again see that the bias and variance cross each other when max depth is 4.0 which is also gives the best model. After this, we see that the variance starts increasing. We also see that the model starts of with a higher bias and error than the previous one but it quickly drops the degrees increase, hence this doesn't become an issue.

4 Discussion and Conclusion

What was first noticed was that for all of these methods, the variance started of low, while the bias started of high. As the complexity increased, each method reached a level where both bias and variance were low. After this point the bias started to flatten out or decrees further while the variance started to increase. For the OLS and the Neural Network (MLP), the variance could quickly get big enough for overfitting to become an issue. The bias-variance tradeoff for the MLP behaves in some ways opposite to that one of the OLS. The decrease in bias is at first much greater than the decrees in variance for the MLP, while the variance stays low at first for the OLS.

We have seen that overfitting is at risk when working with OLS. The penalized regression models, Ridge and LASSO tackle this issue significantly better. We noticed that Ridge with a regularization of $\alpha = 0.1$ yields a much lower variance than OLS for the same polynomial degree. The LASSO procedure, with the same regularization keeps the variance at a minimum, however it starts of with much higher bias than the two other Linear Regression models. Contrary to them, LASSO achieves the optimal model for higher polynomial order. Therefore, we do not need to be as careful with overfitting when we are using the LASSO model.

Further, LASSO and Gradient Boosting are the two only methods we have looked at in which the variance remains low as we increase the complexity. Both models behaved in similar manner, in the sense that the bias and error decreased as the complexity increased. However, Gradient Boosting achieved a lower optimal error, bias and variance compared to LASSO. Even though LASSO is more computationally efficient, Gradient Boosting would still make the better choice based on its performance.

Lastly, when comparing the Neural Network metohd to Gradient Boosting, we notice that both methods started of with low variance, but Gradient Boosting also started with much higher bias than the Neural Network. However, the optimal Neural Network model yielded low values for the bias and variance. However, the optimal error for Neural Networks was significantly higher than it was for Gradient Boosting. Also, considering that the Gradient Boosting method reached the optimal results without increasing the default boosting steps ($n = 100$), it made Gradient Boosting all together more efficient method, even in terms of speed.

Therefore, although not the fastest model, the Gradient Boosting algorithm showed to be the optimal model in terms of bias-variance tradeoff and error. The model, even at times had the bias reach a lower value than the variance. All of the methods studied could reach a point with low bias and variance. But only the LASSO, Gradient Boosting could reach this consistently. As well as Neural Networks, provided you use the optimal number of hidden layers. In terms of speed and execution time, all models performed well for our dataset.

References

[1] M. Mahmoud and A. Davidov. Regression Analysis and Resampling Methodd. PDF, (2021)

[2] M. Mahmoud and A. Davidov. Classification, Regression and FFNN. PDF, (2021)

[3] T. Hastie, R. Tibshirani and J. Friedman. *The Elements of Statistical Learning.* Springer, New York, 2009.