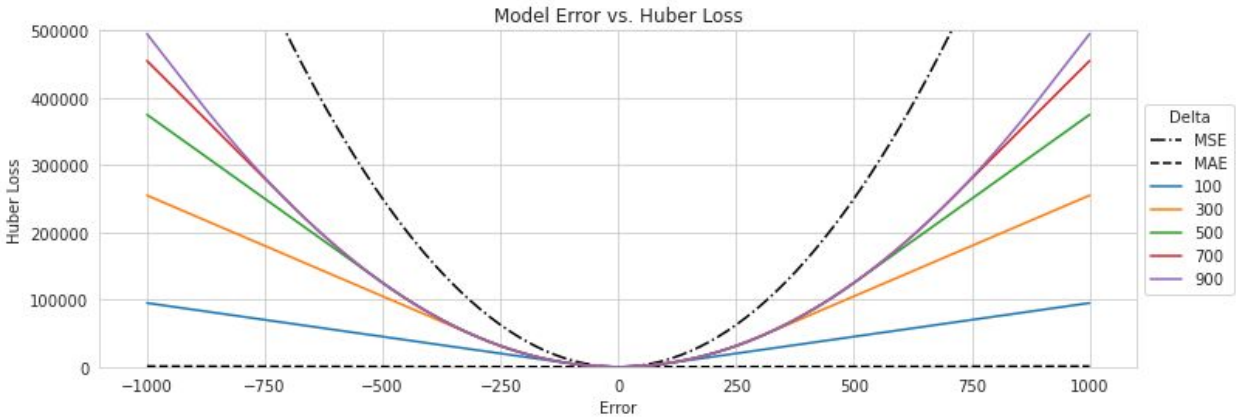
Capstone 2: In-Depth Analysis

The model optimization process, found in the XGB Regressor (Optimized Target) notebook, began with choosing the most appropriate functions to use to minimize the prediction errors, specific to the dataset. The chosen tree boosting library for this project, XGBoost, provided the ability to use a custom objective function for metric performance monitoring, during model training. The XGBoost training module required the objective function to return 2 values, the gradient and the hessian of the evaluation metric, also called the loss function. When given the model predictions and the true data points, this second function was used to calculate the prediction errors. These errors were used as evaluation metrics to assess when training should be halted, called early stopping, due to their lowest values remaining unchanged, after a specified number of additional boosting rounds. For this dataset, target values ranged in scale from under 10 dollars to nearly 10 billion dollars. Considering that many prediction errors were going to be very large, the mean absolute error (MAE) was chosen, as the evaluation metric, as opposed to the mean squared error (MSE). The MSE would have penalized models more than the MAE did, for those predictions that were large in scale. Because the target had been transformed using the natural log function, the loss function was written from scratch to get the prediction errors in dollar amounts. To accomplish this, it was necessary to transform both the predictions and the test labels, using the exponential function, before the loss function was applied to them.

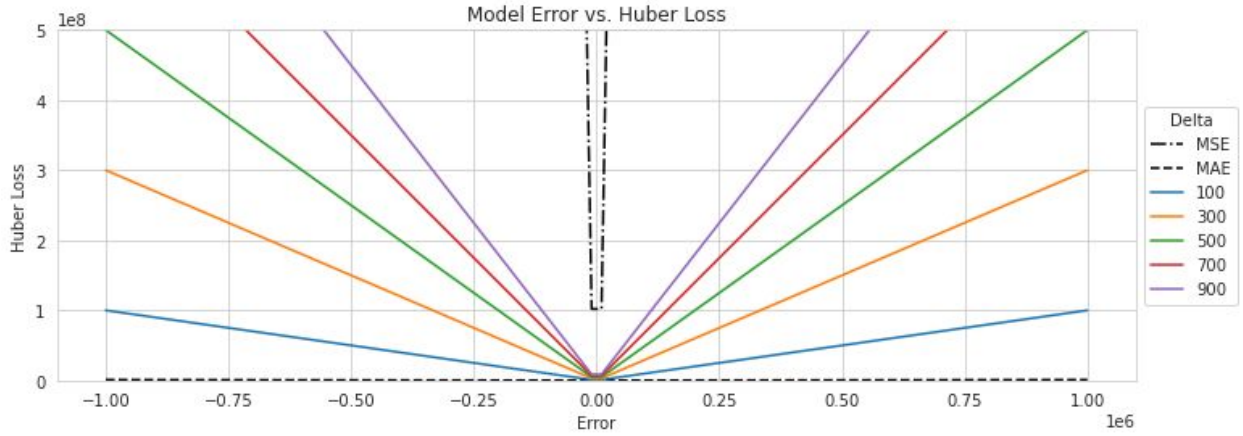
After the evaluation metric was chosen, the objective function needed to be selected. The Huber objective function had the ability to adapt to the wide range of prediction errors that were expected.

$$L_\delta(error) = \begin{cases} 1/2 * (error)^2, & \text{for } |\text{error}| \leq \delta \\ \delta(|\text{error} - 1/2 * \delta|), & \text{otherwise} \end{cases}$$

First, the value for the hyperparameter of this function, delta, needed to be set. The delta value would set an absolute error threshold, below which the behavior of the Huber function would act as a squared error (MSE).



For errors that were above the value of delta, the Huber function would behave like an absolute error (MAE).



Using this function would enable the models to make more accurate predictions, given the wide range of the target values.

Since the Huber function was not everywhere twice differentiable, and the hessian values required by the XGBoost training module were the results of taking the second derivative of the optimization function, the Pseudo-Huber function needed to be used.

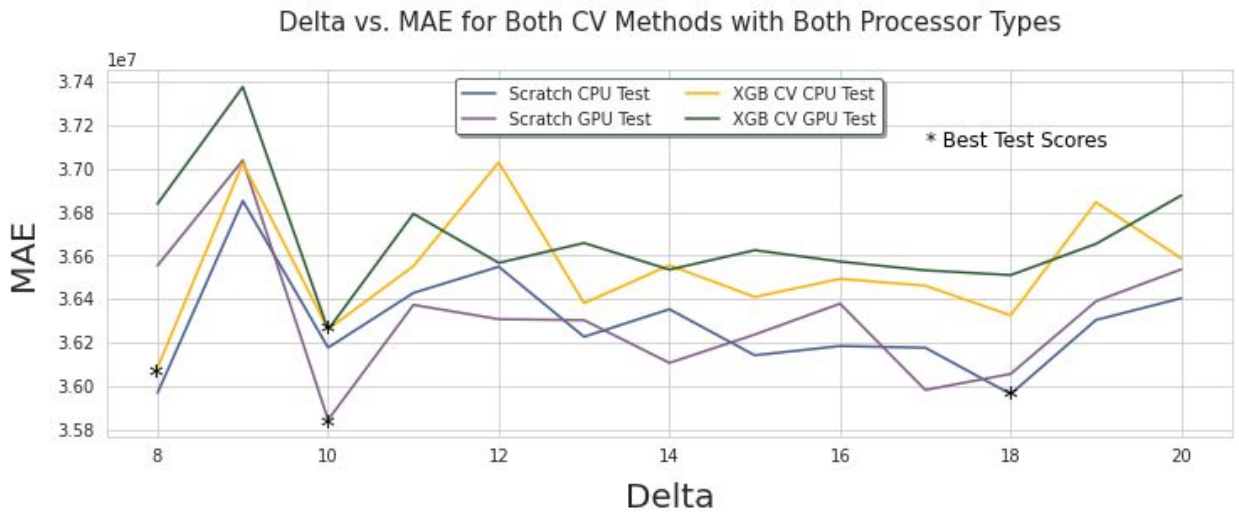
$$L_\delta(error) = \delta^2 * [(1 + (\text{error}/\delta)^2)^{1/2} - 1]$$

This function replicated the behavior of the Huber function used by the XGBoost training module to perform gradient descent, but was everywhere twice differentiable.

A 10 fold cross validation (CV) function was used to select for the best model, to give it better generalizability on new data. The built-in CV function from XGBoost performed differently than a custom CV function that was written from scratch.

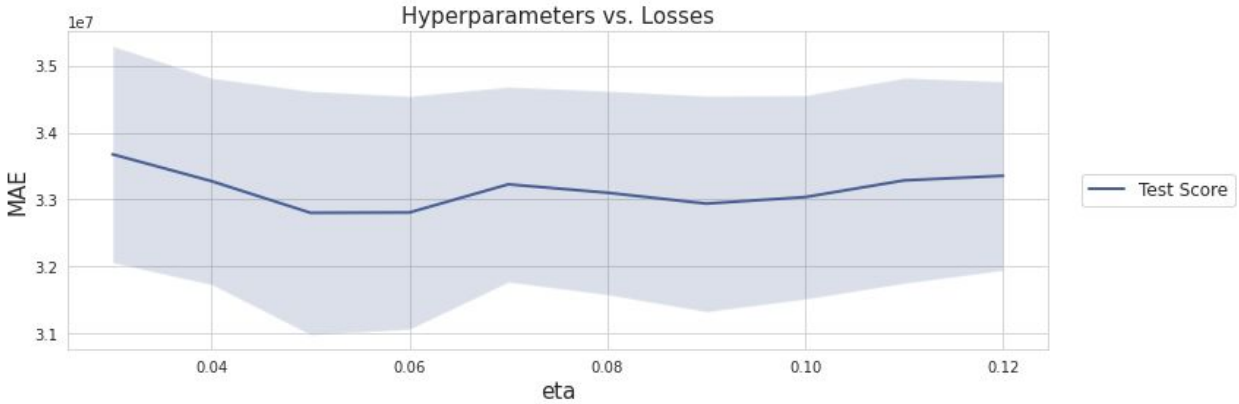
Because accuracy can vary, depending on the hardware doing the calculating, training was performed by using either CPU or GPU operations. The performance differences between these two options was evaluated during the choosing of the optimal delta value to use in the Pseudo-Huber function. Time complexity costs were considered, also, as run-times for the hyperparameter search for the boosted models could get quite long.

Training sessions were run over a range of delta values to find the one which optimized the evaluation errors best. Delta values between 8 and 20 were used , in conjunction with each hardware option and the two CV functions.



The best model found through this process was the result of using GPU calculations with the custom CV function and a delta value of 10. The average 10 fold CV test error, using default boosting hyperparameters, was lowered to $35,843,487, after these optimizations were used. The error had been reduced by an additional 10% from the baseline value. By using the GPU option, the training times were slightly reduced, as well.

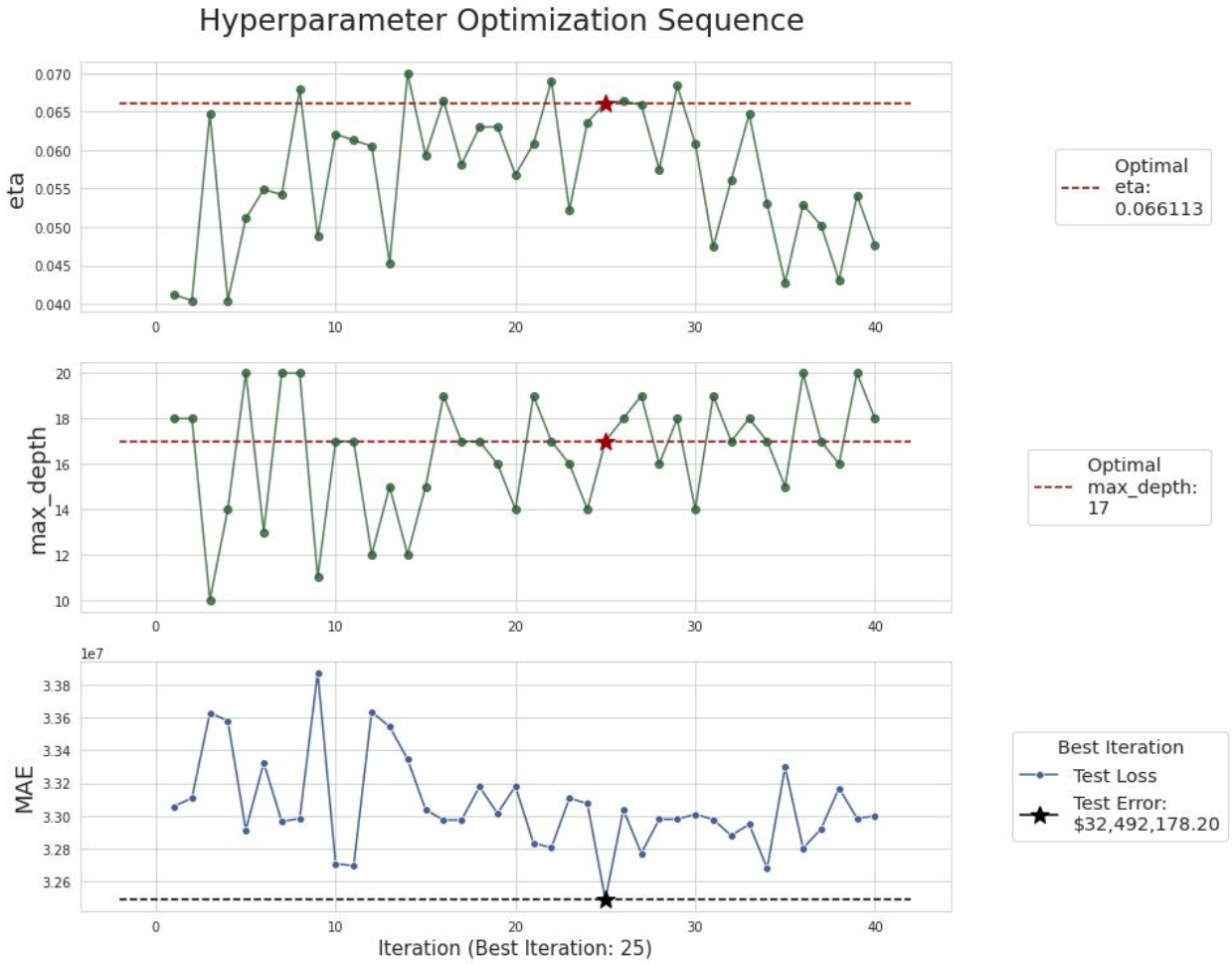
The final optimization phase involved a search for the best boosting hyperparameters. An informed (Bayesian) search technique was performed with the Hyperopt algorithm. The informed search method offered time cost benefits over a brute force grid search, which would have tried all combinations of hyperparameters throughout the search space. Instead, the choices of hyperparameters were based on the results of previously chosen models. Each round of tuning used a range of values, selecting one hyperparameter at a time. Each search round ended after a preselected number of tries did not produce a better model. This was another instance of early stopping. After one hyperparameter was selected, any others that were previously tuned were double checked for retuning. The average 10 fold CV test error was the metric used to evaluate which model had the best set of hyperparameters. A very limited grid search was performed to determine the best initial tuning range to set for each hyperparameter.



After the results were plotted, local minima were revealed. These were the values that made up the initial search spaces.

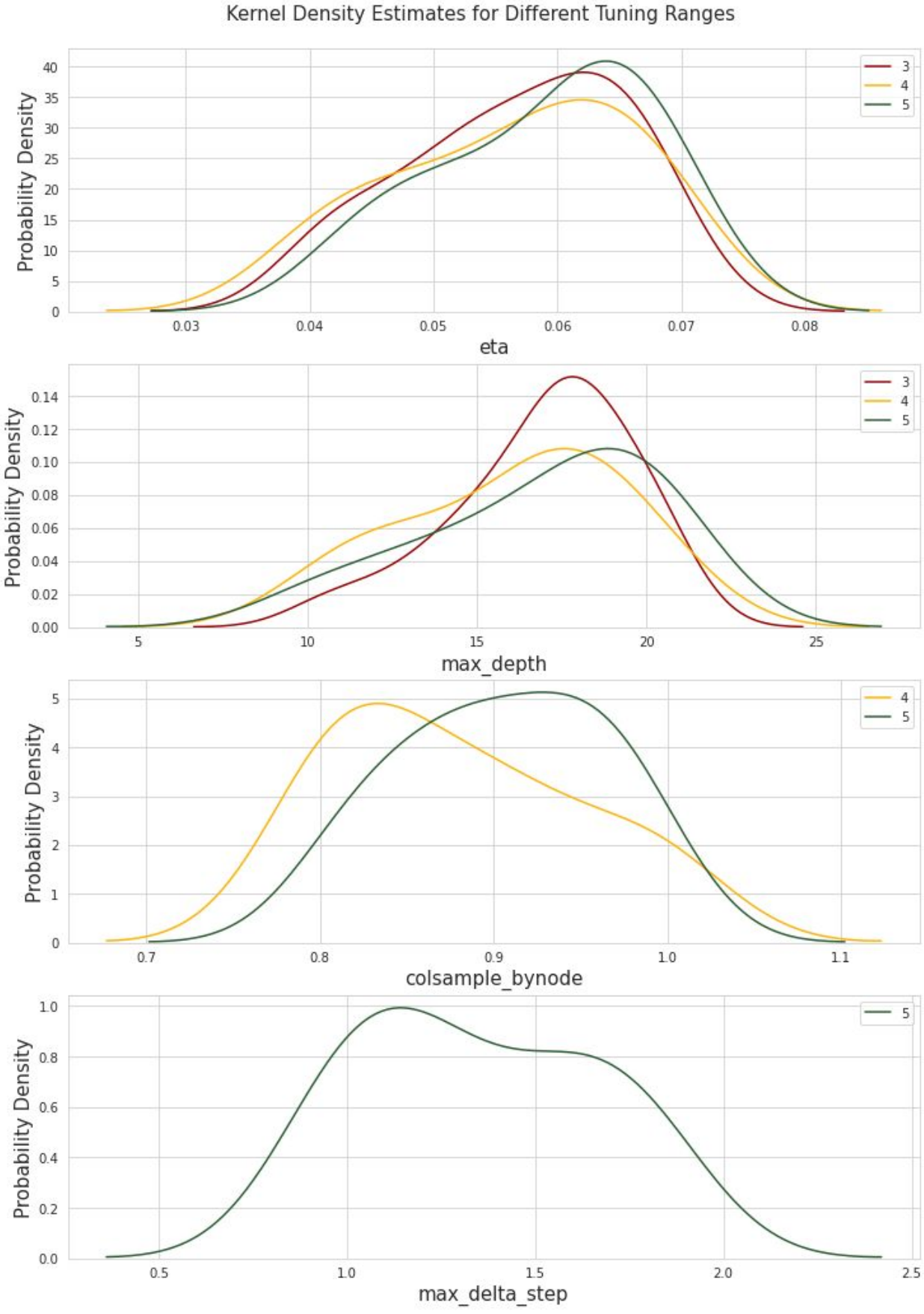
The XGBoost training module required that the maximum number of boosting rounds be predetermined. The initial value was set to be 500 rounds. With early stopping set to 100 rounds, this allowed the models to become quite complex. The tuning of regularization hyperparameters would balance this complexity.

For each round of tuning, the choices of hyperparameters selected by the informed search algorithm were plotted, along with their associated test errors.



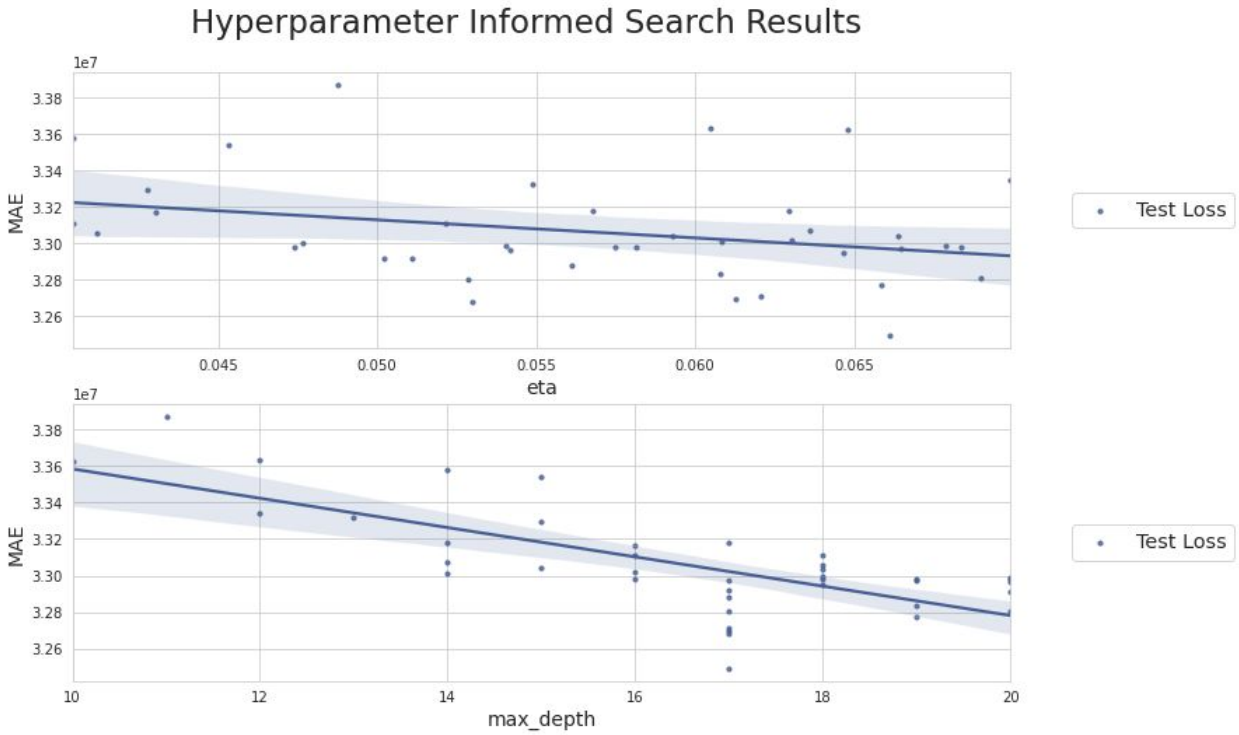
These plots were examined to determine if the search spaces needed to be expanded, as occasionally trial runs produced many models that preferred values on the cusps of their allowed ranges.

The kernel density estimates of the distributions of chosen hyperparameter values were plotted, as well.


Kernel Density Estimates for Different Tuning Ranges
Probability Density
eta
3
4
5
Probability Density
max_depth
3
4
5
Probability Density
colsample_bynode
4
5
Probability Density
max_delta_step
5

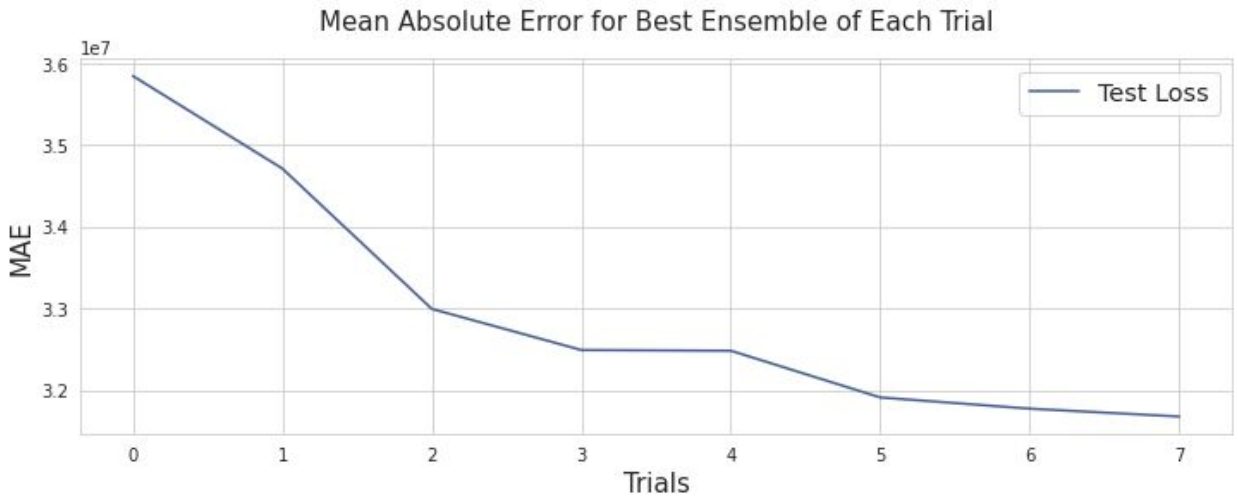
These plots complemented the search space visuals to confirm any needs for hyperparameter range adjustments.

Finally, regression lines were placed on scatterplots of the hyperparameter values vs. the test errors.



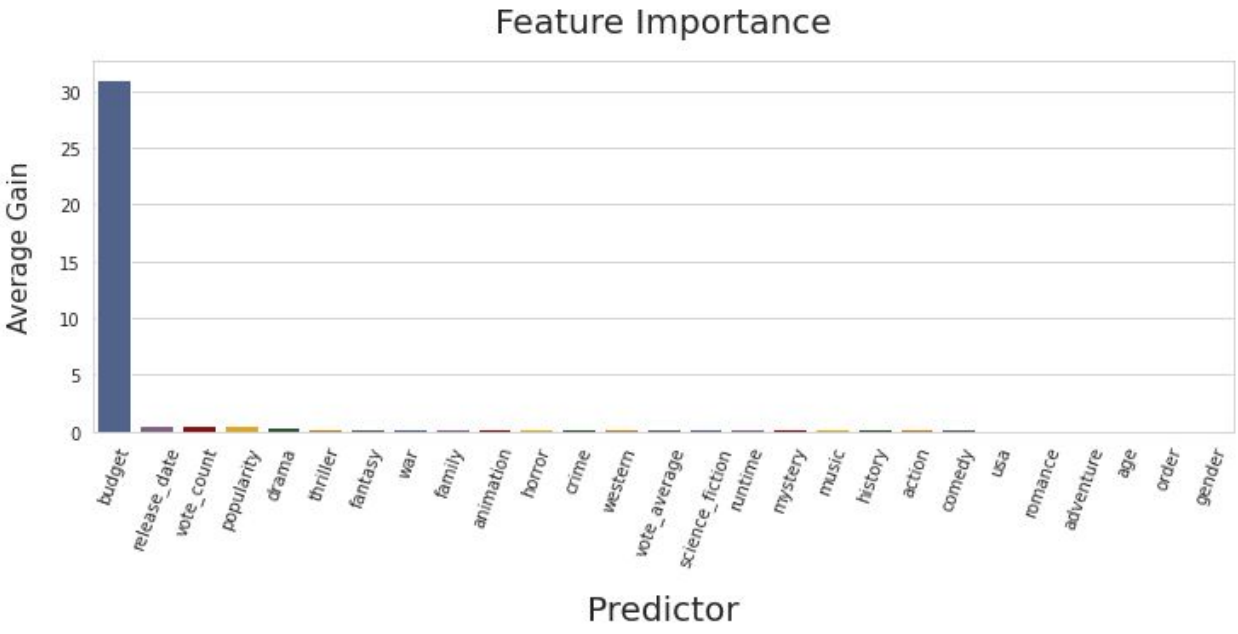
The vertical tilting of these lines would indicate the possibility of finding better performing models at hyperparameter values in that direction, possibly beyond the current range for that round.

This optimization algorithm was performed for 11 hyperparameters. Only 4 of them yielded improved models from the previous tuning round. Two of them, eta, also known as learning rate, and max delta step increased the regularization of the residual errors at the leaf nodes. One of them, colsample by node, adjusted a sampling method, which selected a subset of features to use at every leaf node. The last one, max depth, controlled how well the models learned the structure of the dataset, by allowing the individual trees to make more decisions, as they grew deeper. A hard cap of 20 layers was set on this hyperparameter, as deeper trees became extremely time cost inefficient. After the last hyperparameter was tuned, the limit was raised on the maximum number of boosting rounds to 1,000. This, set with an optimal number of early stopping rounds of 200 iterations, provided one last push to squeeze out any last bit of performance.



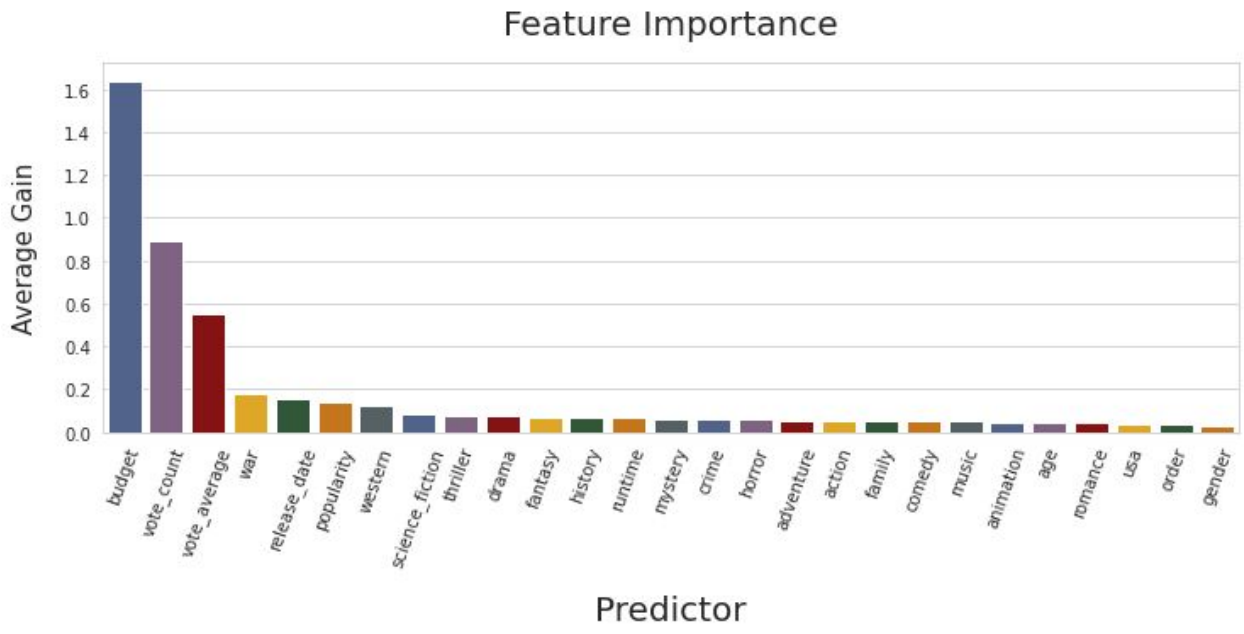
Tuning the hyperparameters gave an improvement of an additional 17%. The average test error was decreased to $31,678,760 for the final tuned model, using an optimized target variable and superior optimization function. The total reduction in the average 10 fold CV test MAE, from start to finish, was 44.2%.

The initial boosted tree ensemble, before the start of the hyperparameter tuning, was composed of many weak learners.



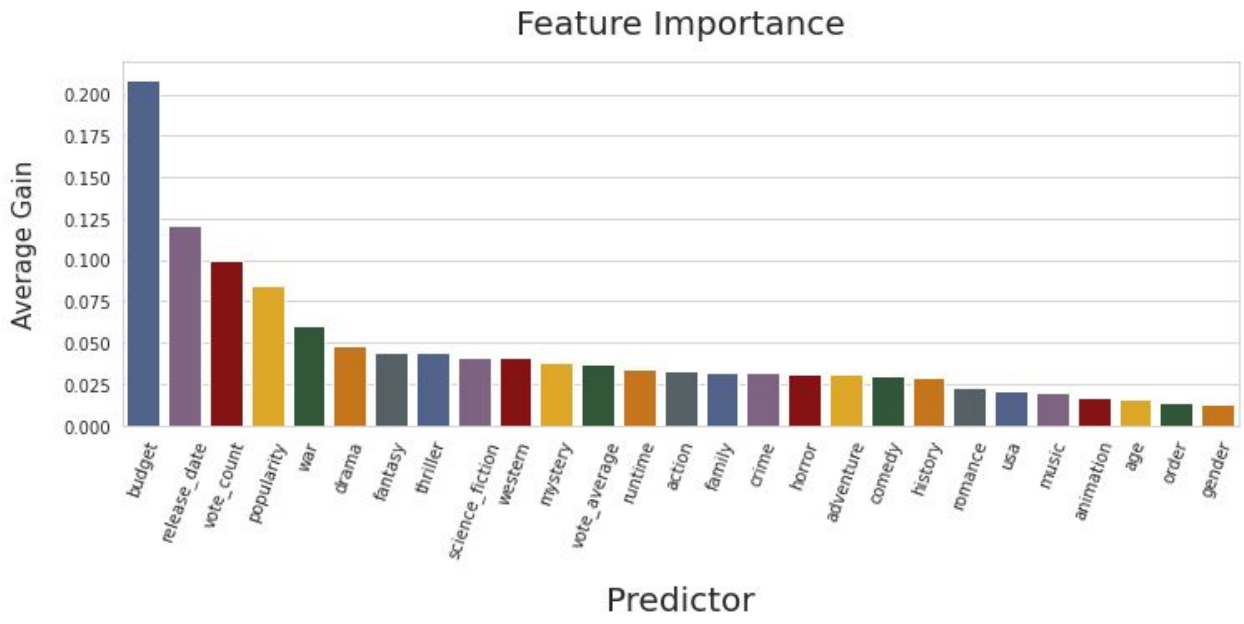
Nearly all of the decisions made by the untuned model were based on values of the budget feature.

As the hyperparameter selections progressed, the trees were able to find more structure in the information.



After tuning the first 3 hyperparameters, the best model was using a richer variety of the features to make its predictions.

By the end of the optimization algorithm, the best model was very complex.



Four of the features were being used extensively to predict, and they were all used to some significant degree.

After the final model was selected, it was used to make predictions on the validation data to observe the models performance on out-of-sample data. The average 10 fold CV validation error was $38,963,668. The final model was 77% as accurate as it was on the CV training data. This decrease was not surprising, as the size of the validation dataset was 25% of the size of the training set.