International Journal of All Research Education and Scientific Methods (IJARESM),
ISSN: 2455-6211, Volume 12, Issue 10, October-2024, Available online at: www.ijaresm.com

Exploring Key XGBoost Hyperparameters: A Study on Optimal Search Spaces and Practical Recommendations for Regression and Classification

Vibhu Verma

Principal Data Scientist, GWU, Capital One NY, USA

----------**********----------

ABSTRACT

This paper presents an in-depth exploration of four key hyperparameters—min_child_weight, n_estimators, max_depth, and learning_rate—in XGBoost (eXtreme Gradient Boosting), a machine learning algorithm recognized for its efficacy in diverse regression and classification tasks. By evaluating these hyperparameters across six datasets, we assess their impact on model performance in both regression and classification contexts. Given the computational demands of hyperparameter tuning, especially with large datasets, we propose an initial search space for each parameter to balance performance gains with resource efficiency. This approach offers practical guidance for optimizing XGBoost while mitigating the high computational costs associated with extensive searches. The findings provide insights into the role of each parameter in enhancing model accuracy and generalization, supporting practitioners in making effective tuning decisions for real-world applications.

Keywords— XGBoost, hyperparameters, min child weight, n_estimators, max depth, learning rate, regression, classification

INTRODUCTION

XGBoost, short for eXtreme Gradient Boosting, is a high-performance machine learning algorithm based on the gradient boosting framework and designed to handle large datasets and complex model requirements efficiently. Developed with scalability and speed in mind, XGBoost has become a popular choice across data science applications, especially where predictive accuracy and computational efficiency are paramount. The algorithm has consistently demonstrated strong performance in various fields, from finance and healthcare to e-commerce and image recognition, making it a staple in both industry and academia. In essence, XGBoost leverages the principles of ensemble learning, where a series of weak learners (in this case, decision trees) are iteratively added to minimize prediction errors, with each successive model focusing on the residuals of its predecessors. This iterative refinement process enables XGBoost to produce robust, highly accurate models, capable of handling complex interactions and non-linear patterns.

One of the standout features of XGBoost is its support for parallelized computation, making it particularly suitable for large-scale data applications. Additionally, XGBoost offers several algorithmic optimizations that differentiate it from standard gradient boosting. These include a novel, regularized objective function to prevent overfitting, weighted quantile sketching to handle sparse data, and a sparsity-aware algorithm for data handling. XGBoost also supports customizable loss functions and advanced regularization terms (L1 and L2), giving it the flexibility to perform well on both classification and regression tasks, even with noisy data or datasets that are prone to overfitting. These improvements are not only practical in real-world applications but also contribute to its popularity in machine learning competitions, where time and computational constraints are often decisive factors.

Given its capabilities, XGBoost allows practitioners to tune a wide range of hyperparameters, which control various aspects of the model's structure, learning process, and regularization. Effective tuning of these hyperparameters is essential to achieve optimal performance, as improper settings can lead to overfitting, underfitting, or unnecessary computational overhead. However, hyperparameter tuning can be a costly and time-consuming endeavor, especially when working with large datasets, and often requires significant computational resources. For this reason, it is important to identify a balanced approach to tuning, where a practical yet effective initial search space is selected to avoid exhaustive searches that yield minimal improvements.

This study focuses on four key XGBoost hyperparameters—min_child_weight, n_estimators, max_depth, and learning_rate—which have a major impact on model performance and generalization ability. These parameters are not only among the most influential but also tend to be common across various applications, making them critical to understand and optimize for both regression and classification tasks. We will test these hyperparameters across six

datasets to assess their effects on model accuracy and resource efficiency. Additionally, we propose an initial search space for each parameter, providing practical guidance for practitioners aiming to strike a balance between performance and computational cost. The findings aim to inform efficient tuning strategies for XGBoost, helping practitioners navigate the trade-offs associated with extensive hyperparameter optimization in resource-constrained settings.

In addition to min_child_weight, n_estimators, max_depth, and learning_rate, hyperparameters like subsample and colsample_bytree can also be tuned to enhance model performance. These parameters control data sampling and feature selection, respectively, allowing practitioners to optimize for bias-variance trade-offs and improve generalization across diverse datasets.

Key Concepts

In XGBoost, tuning the right hyperparameters is crucial for optimizing model performance and generalization capabilities. The four hyperparameters we focus on—min_child_weight, n_estimators, max_depth, and learning_rate—are not only influential but also commonly adjusted to improve outcomes in regression and classification tasks. Each of these hyperparameters plays a distinct role in controlling the complexity, robustness, and efficiency of the XGBoost model.

Learning Rate

Learning rate, sometimes referred to as the "shrinkage" parameter, controls the weight given to each new tree added to the ensemble. By scaling the contribution of each tree, learning_rate slows down the learning process, allowing the model to learn more gradually and potentially reduce the risk of overfitting. Smaller values of learning_rate make the model less likely to overfit but require a higher number of trees (n_estimators) to reach optimal performance, which can increase training time. Conversely, higher learning rates can accelerate training but may result in a model that fails to generalize well. Properly tuning learning_rate is therefore essential to balance the learning speed and accuracy, ensuring the model gradually converges to a solution that generalizes well on unseen data.

Max Depth

Max depth is a parameter that limits the depth of each tree in the ensemble, controlling how much detail the model captures. Trees with a higher maximum depth can model more complex interactions among features, which may improve accuracy on the training set. However, a high max_depth can lead to overfitting, as the model starts capturing noise along with the signal in the data. Setting max depth to a smaller value makes the model more generalized and interpretable, which can be especially beneficial in applications where interpretability is critical. This parameter thus serves as a key control over the model's capacity, with smaller depths often resulting in more generalized models and larger depths capturing complex, nonlinear patterns.

Number of Estimators

Number of estimators controls the number of trees in the XGBoost model, directly impacting the learning process and the final model complexity. Each tree added to the ensemble works to correct the residual errors of the previous trees, enhancing the model's predictive accuracy. While more trees generally improve model accuracy by capturing finer patterns in the data, an excessively large number of trees can lead to overfitting and increased computational time, especially on large datasets. Striking the right balance is essential: too few trees may result in an underfit model, whereas too many can make the model unnecessarily complex. Thus, determining an appropriate range for n_estimators is critical in managing both model performance and training efficiency

Min Child Weight

Min child weight is a regularization parameter that specifies the minimum sum of instance weights required in a child node. This parameter plays an essential role in controlling the complexity of the model by determining whether or not a node should be further split based on its instance weight sum. A higher min_child_weight value results in fewer splits, effectively pruning the model and making it less likely to overfit, especially in cases where the dataset is noisy or contains a significant number of outliers. Lower values, on the other hand, make the model more complex by allowing nodes with lower instance weights to be split, which can capture more intricate patterns but may also increase the risk of overfitting.

Experiment Design

The objective of this experimental design is to systematically vary four critical hyperparameters in XGBoost—min_child_weight, n_estimators, max_depth, and learning_rate—to investigate their impact on model performance and convergence speed. This study aims to establish a clear understanding of how these parameters interact and affect the model's ability to generalize from the training data to unseen instances.

TABLE I. DATASET DETAILS

Data Set Name	Dataset Definitions			
	# Instances	# Features	Train Test Split	Problem Type
Air Quality UCI	9357	11	80:30	Regression
All State Claims Severity	188318	131	80:20	Regression
Parkinsons Telemonitoring	5875	22	80:30	Regression
Credit-Card Fraud Data	284807	29	80:20	Classification

For this experiment, we focus exclusively on the following hyperparameters, using the specific ranges you've defined:

min_child_weight: Ranging from 1 to 20 in increments of 0.5. Lower values allow the model to capture complex patterns in the data, but may increase the risk of overfitting, especially in noisy datasets. Higher values help to prune the tree structure and promote generalization by requiring a larger minimum weight sum for node splitting.

n_estimators: This parameter defines the number of trees in the ensemble and is set in this experiment to [1, 5, 10, 20, 30, 40] followed by increments from 50 to 500 in steps of 50. By varying this parameter, we can observe how the inclusion of additional trees affects model performance and convergence. While more trees generally enhance accuracy by capturing more details, too many trees may lead to overfitting and increased computational cost.

max_depth: The maximum depth of each tree ranges from 1 to 50 in increments of 5, with an additional option of None (unrestricted depth). This parameter plays a critical role in balancing model complexity and generalization. A higher maximum depth allows the model to learn intricate relationships but can increase the risk of overfitting. Conversely, lower depths promote simpler models that may not fully capture underlying data patterns.

learning_rate: This parameter controls the contribution of each tree to the final model and ranges from 0.01 to 1, with 100 equally spaced values. A lower learning rate leads to a more gradual learning process, often requiring more trees (n_estimators) to achieve optimal performance. This parameter significantly influences how quickly the model converges to an effective solution.

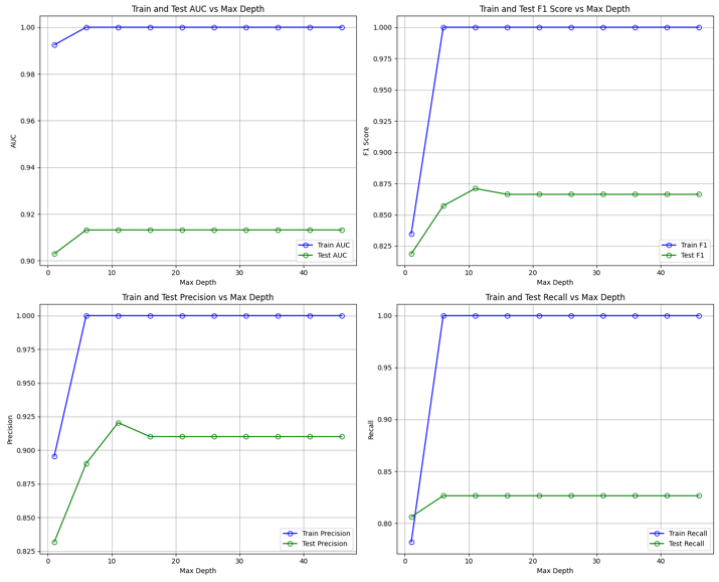

Fig. 1. Credit Card Fraud and Model Performance with Max Depth

International Journal of All Research Education and Scientific Methods (IJARESM),
ISSN: 2455-6211, Volume 12, Issue 10, October-2024, Available online at: www.ijaresm.com

EXPERIMENT SETUP AND EXECUTION

The experimental setup consists of several key components:

Datasets: We will utilize a diverse set of datasets that represent different domains and characteristics to ensure the results are generalizable. These datasets will vary in size, complexity, and data types.

Performance Metrics: Define metrics for model evaluation, including Mean Squared Error (MSE) for regression tasks and accuracy or F1-score for classification tasks. These metrics will help quantify the performance of each model configuration.

Cross-Validation: Implement a simple train-test split approach, where the model will be trained on one dataset and evaluated on a separate test set. This method ensures that performance estimates are unbiased and reflective of how the model will perform on unseen data.

For each hyperparameter, we will assess its impact on model performance. For example, we will vary min_child_weight from **1 to 10** and observe the resulting changes in performance metrics. We will follow a similar process for n_estimators, max_depth, and learning_rate, systematically recording the performance outcomes for each configuration.

After executing the experiments, we will collect and analyze the data related to convergence of gain and performance metrics. We will measure the number of boosting rounds taken for each configuration to reach optimal performance.

This analysis will help us identify the configurations that yield the highest performance across datasets.

RESULTS

For the classification task on the credit card data, we observed that using a learning rate of 0.3 or higher leads to diminishing returns. Specifically, the test AUC begin to plateau and eventually decline, indicating overfitting on the training data and underperformance on the test data.

Analyzing the gain for all splits reveals that lower learning rates yield more efficient splits, resulting in higher gains, especially when set below 0.4 which shows that the model converges more efficiently for smaller gains.

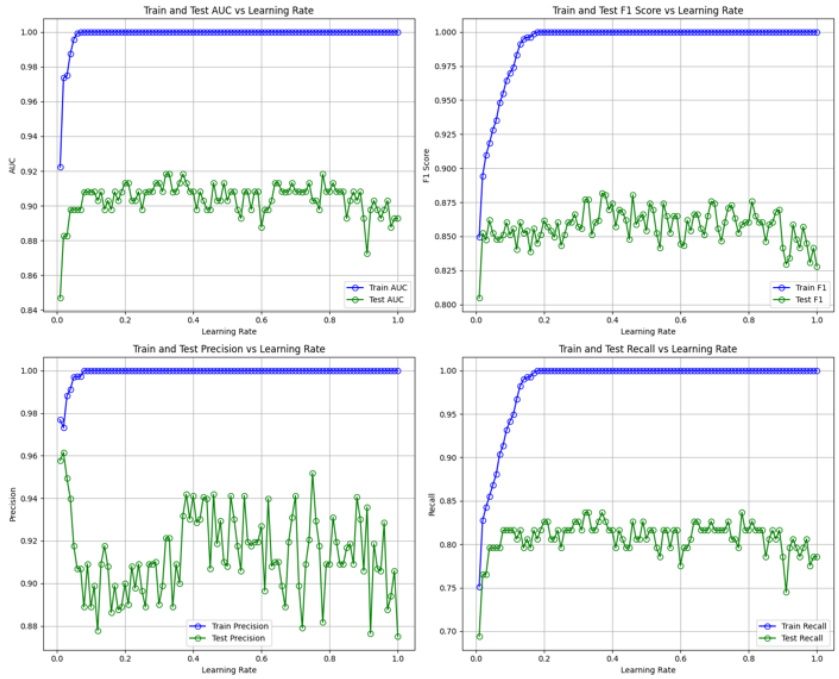

Fig. 2. Credit Card Fraud and Model Performance with Learning Rate

International Journal of All Research Education and Scientific Methods (IJARESM),
ISSN: 2455-6211, Volume 12, Issue 10, October-2024, Available online at: www.ijaresm.com

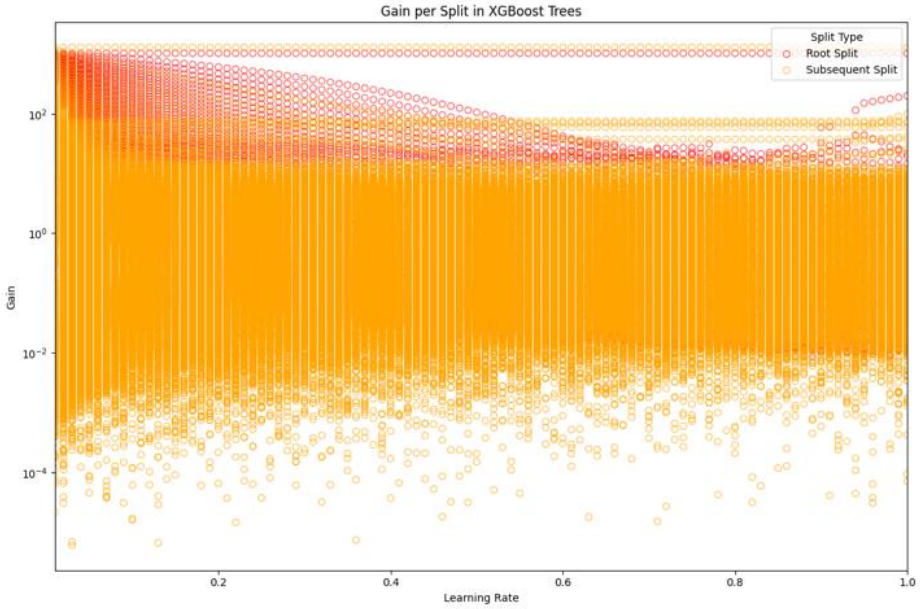

Fig. 3. Credit Card Fraud and Model Gain Efficiency with Learning Rate

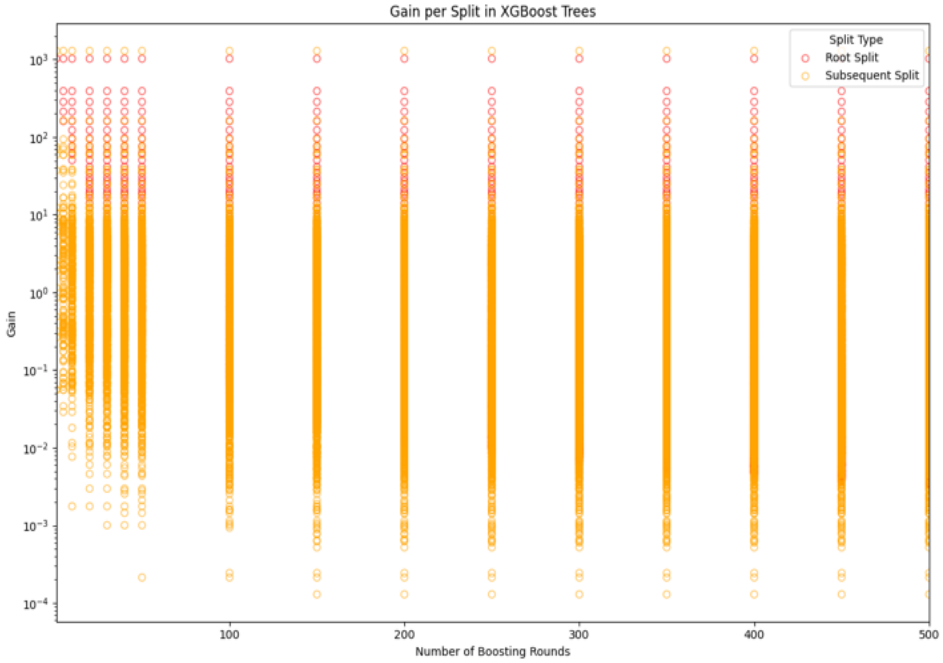

Fig. 4. Credit Card Fraud and Model Gain Efficiency with Num Estimators/ Number of Boosting Rounds

Additionally, we found that decision trees with a maximum depth of 10 or greater perform poorly on the test data. The training performance for trees with a maximum depth exceeding 10 also plateaus, suggesting we should limit our maximum depth to 10 or below. The gain analysis further supports this, as we see more efficient gains from trees with shallower depths, indicating more effective splits.

Regarding the number of estimators, our observations show that there are no significant performance improvements when the number exceeds 100 for both the training and testing sets. Therefore, we recommend keeping the number of estimators between 20 and 120 to identify the optimal point.

The gain plot indicates that fewer estimators correspond to a higher density of greater gains, highlighting that more efficient splits occur with a smaller number of boosting rounds.

International Journal of All Research Education and Scientific Methods (IJARESM),
ISSN: 2455-6211, Volume 12, Issue 10, October-2024, Available online at: www.ijaresm.com

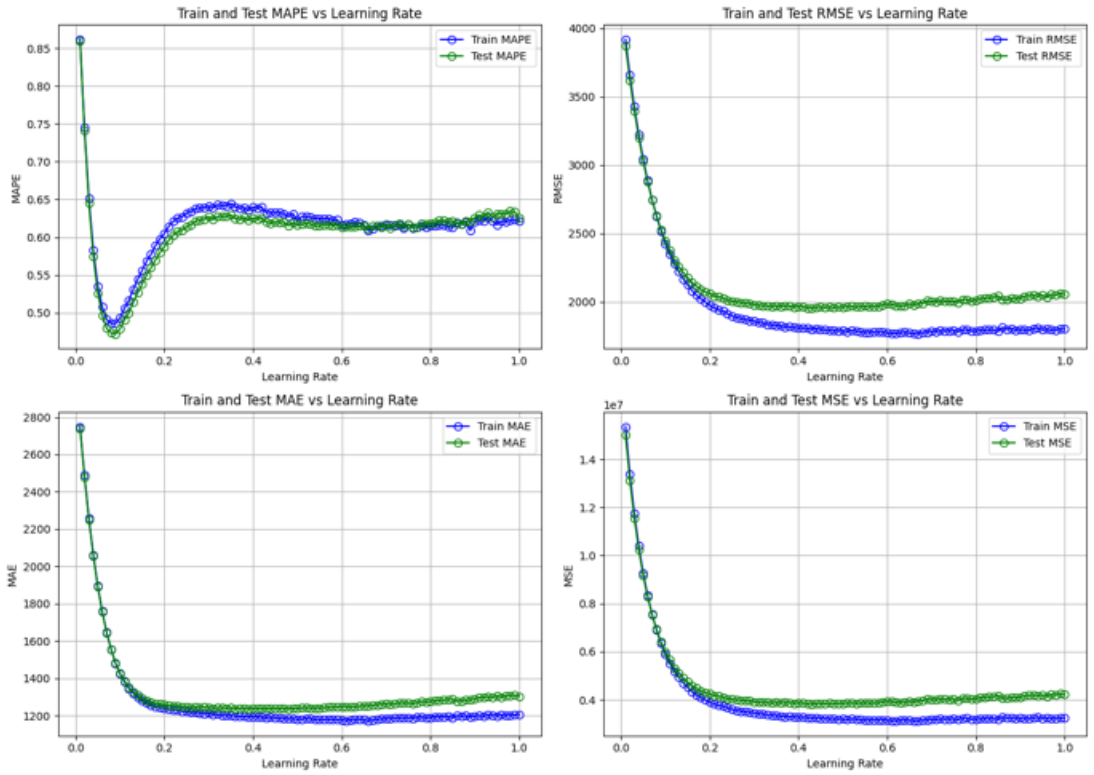

Fig. 5. All State Data and Model Performance with Learning Rate

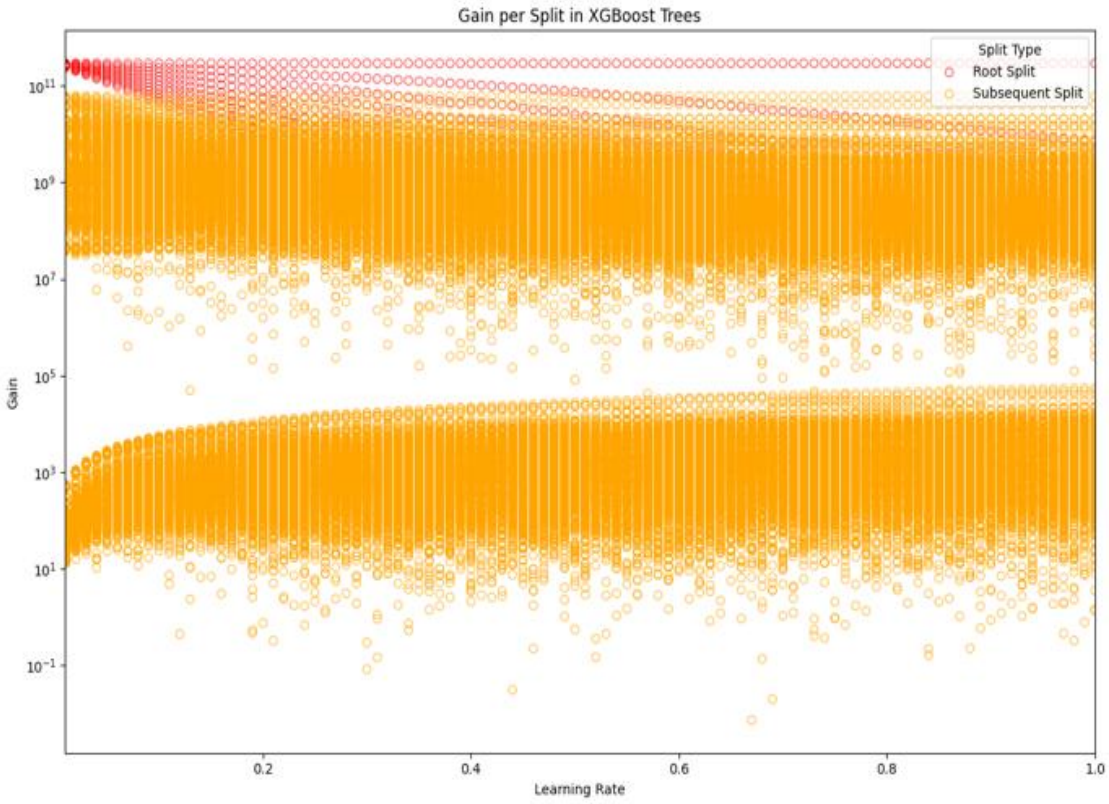

Fig. 6. All State Data and Model Gain Efficiency with Learning Rate

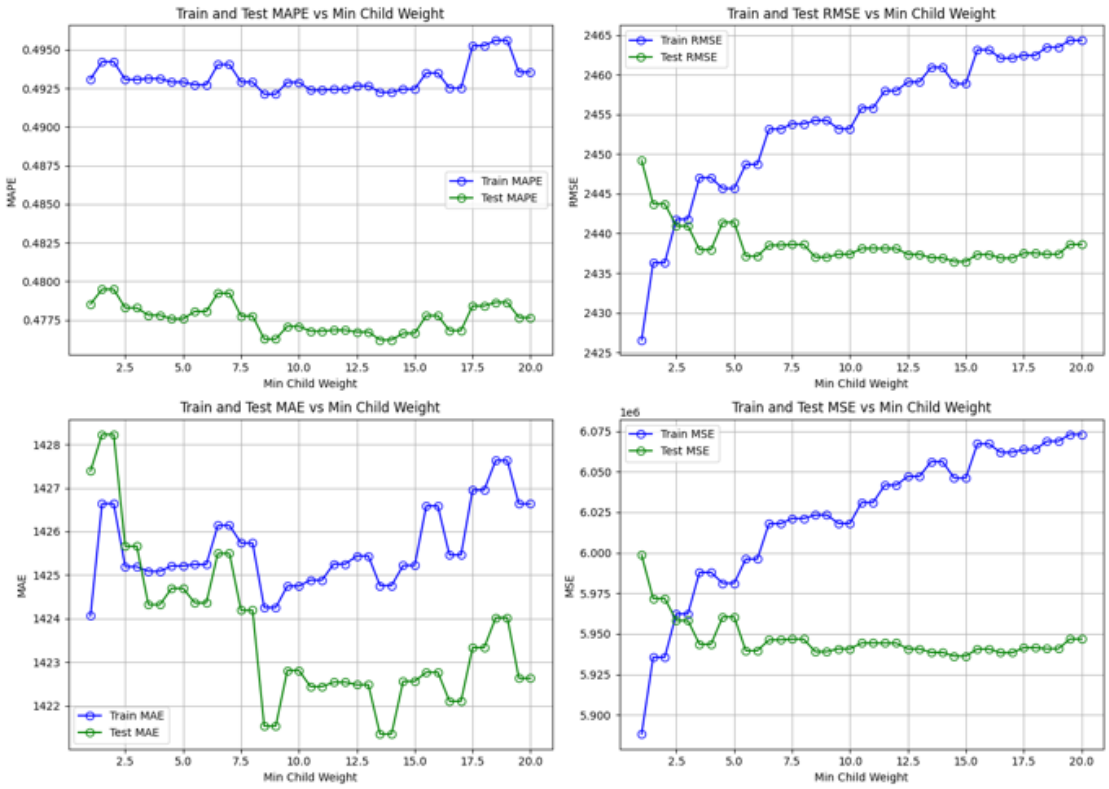

Fig. 7. All State Data and Model Performance with Min Child Weight

Lastly, our analysis of the minimum child weight revealed that values between 1 and 10 are most effective for test performance.

For the air quality regression task, we observed that learning rates greater than 0.3 resulted in higher values for root mean square error (RMSE), mean absolute percentage error (MAPE), and mean absolute error (MAE). This suggests that a larger learning rate may hinder convergence to an optimal solution. Analyzing the gains reveals a significant density of larger gains at smaller learning rates, reinforcing the effectiveness of using lower rates.

Regarding maximum tree depth, we noted that depths greater than 10 lead to diminishing returns and plateauing performance on the test set. Therefore, we recommend limiting the maximum depth to under 10. Additionally, the number of estimators should be kept below 100 to maximize model performance on the test set.

For the Allstate claims data, a learning rate around 0.2 to 0.3 produced optimal results across various metrics on the test set. A maximum depth of approximately 10 yielded the highest scores, while depths exceeding this threshold indicated signs of overfitting on the training set.

We also observed diminishing returns with an increasing number of estimators, suggesting that a more complex model may lead to overfitting, ultimately resulting in underperformance on the test set. Furthermore, a minimum child weight greater than 10 was associated with poor performance on the training set, also showing diminishing returns on the test set.

The same trends were observed in the Parkinson's telemonitoring prediction data, aligning with our findings from the air quality and Allstate claims datasets.

In the classification task, optimal hyperparameter ranges were identified as a learning rate below 0.4, a maximum depth of 10 or lower, between 20 to 120 estimators, and a minimum child weight between 1 and 10. For the regression task, the best performance was observed with a learning rate around 0.2, a maximum depth under 10, fewer than 100 estimators, and a minimum child weight above 1.5 and less than 10.

These findings establish a robust foundation for hyperparameter optimization (HPO) as we prepare to delve deeper into the tuning process. By pinpointing effective ranges for key parameters, we can streamline our search for optimal model configurations. To enhance our efficiency further, we can utilize Bayesian optimization techniques with libraries that are designed to intelligently explore these hyperparameter search spaces. This approach enables us to achieve greater

performance gains while avoiding the computational expense of exhaustively searching the entire grid of values, ultimately leading to more effective model optimization.

REFERENCES

[1] Citation: T. Chen and C. Guestrin, "XGBoost: A Scalable Tree Boosting System," in Proceedings of the 22nd ACM SIGKDD International Conference on Knowledge Discovery and Data Mining, San Francisco, CA, USA, 2016, pp. 785–794. doi: 10.1145/2939672.2939785.
[2] Citation: V. Gupta, P. Kumar, and A. Kumar, "A Comprehensive Review on XGBoost: Application and Performance Evaluation," Journal of King Saud University - Computer and Information Sciences, 2023. doi: 10.1016/j.jksuci.2023.06.017.
[3] Citation: S. L. Tang and H. J. Huang, "XGBoost for Regression and Classification: A Tutorial," Journal of Data Science, vol. 18, no. 4, pp. 539-550, 2020. doi: 10.6339/JDS.2020.18.4.539.
[4] Citation: N. A. Hasan and M. H. Zainudin, "An Empirical Comparison of Supervised Learning Algorithms for Predicting Student Performance," International Journal of Electrical and Computer Engineering, vol. 11, no. 3, pp. 2601-2608, 2021. doi: 10.11591/ijece.v11i3.6803
[5] Citation: R. Dias, L. L. J. Ribeiro, and A. F. da Silva, "Hyperparameter Optimization of XGBoost for Predictive Maintenance Using Bayesian Optimization," Expert Systems with Applications, vol. 221, pp. 119484, 2023. doi: 10.1016/j.eswa.2023.119484.
[6] Citation: T. Akiba et al., "Optuna: A Next-generation Hyperparameter Optimization Framework," Proceedings of the 25th ACM SIGKDD International Conference on Knowledge Discovery & Data Mining, pp. 2623-2631, 2019. doi: 10.1145/3292500.3330581.
[7] Citation: J. S. Lim and S. D. Lee, "Understanding the Performance of XGBoost Algorithm in Predictive Modeling," Applied Sciences, vol. 11, no. 18, pp. 8382, 2021. doi: 10.3390/app11188382.