A Comparison of AutoML Hyperparameter Optimization Tools for Tabular Data

by

Prativa Pokhrel

Submitted in Partial Fulfillment of the Requirements

for the Degree of

Master

of

Computing and Information Systems

YOUNGSTOWN STATE UNIVERSITY

May, 2023

A Comparison of AutoML Hyperparameter Optimization Tools for Tabular Data

Prativa Pokhrel

Signature:

Prativa Pokhrel, Student Date

Approvals:

Alina Lazar, Thesis Advisor Date

Dr. Feng Yu, Committee Member Date

Dr. John R. Sullins, Committee Member Date

Dr. Salvatore A. Sanders, Dean of Graduate Studies Date

ABSTRACT

The performance of machine learning (ML) methods, including deep learning, for classification and regression tasks applied to tabular datasets is sensitive to hyperparameters values. Therefore, finding the optimal values of these hyperparameters is integral to improving the prediction accuracy of a machine learning algorithm and the model selection. However, manually searching for the best configuration is a tedious task, and many AutoML (automated machine learning) frameworks have been proposed recently to help practitioners solve this problem. Hyperparameters are the values or configurations used to control the algorithm's behavior while building the model. Hyperparameter optimization is the guided process of finding the best combination of hyperparameters that delivers the best performance on the data and task at hand in a reasonable amount of time. In this work, the performance of two frequently used AutoML hyperparameter optimization frameworks, Optuna and HyperOpt, are compared on popular OpenML tabular datasets to identify the best framework for tabular data. The results of the experiments show that the performance score of Optuna is better than that of HyperOpt, while HyperOpt is the fastest for hyperparameter optimization.

Acknowledgements

I would like to express my sincere gratitude and appreciation to my thesis advisor and my supervisor, Dr.Alina Lazar of the Department of Computer Science and Information Systems at Youngstown State University for her unwavering support and guidance. Her invaluable expertise, insightful feedback, and unwavering dedication have been instrumental in helping me to complete this thesis.

I would also like to thank the committee members, Dr. Feng Yu and Dr. John R. Sullins for their precious time and guidance during my thesis process. Their expertise and insightful feedback have significantly contributed to the quality of this thesis.

I would like to extend my sincere appreciation to the Department of Computer Science and Information Systems and the College of Graduate Studies for the financial support they provided during my graduate studies.

Finally, I must express my heartfelt thanks to my family and to my friends for their unconditional love and support throughout my years of study, which has been essential in helping me to achieve my academic goals. This accomplishment would not have been possible without their encouragement and support. Thank you.

Table of Contents

List of Figures

List of Tables

1 Introduction

Machine learning (ML) is a rapidly growing field that has the potential to revolutionize industries. Its ability to learn and process large amounts of data quickly and accurately and make predictions has made it a powerful tool for solving a wide range of problems. ML has been rapidly applied in various real-world scenarios, including customer service, fraud detection, and medical diagnosis. Studies have demonstrated that an optimal set of hyperparameters enhances the model's performance. [2] [3]

However, if an ML model's performance is subpar, it can result in significant negative impacts on people's lives, particularly in areas like medical diagnosis or fraud detection. The performance of ML for classification and regression tasks applied to tabular datasets is sensitive to hyperparameter values. Hyperparameters are used to control the behavior of the algorithms while building the model. And hyperparameter optimization aims to determine the optimal combination of hyperparameters to deliver optimal performance on the data in a reasonable amount of time. Finding the optimal values of these hyperparameters is integral to improving the prediction accuracy of an ML algorithm and the model selection process. However, manually searching for the best configuration is a tedious task, and many AutoML (automated machine learning) frameworks have been proposed recently to help practitioners solve this problem.

The comparative study of hyperparameter optimization frameworks is crucial for society and the world, as it has far-reaching implications for various industries and applications. Improved ML models can lead to better and quicker decision-making and outcomes in areas such as healthcare, finance, and transportation. For example, a well-tuned ML model can help diagnose diseases more accurately, leading to better patient outcomes. However, optimizing the performance of machine learning algorithms by searching for the best hyperparameter sets takes time and computing resources. Performing this process by hand is convoluted and time-consuming. By automating the parameter search, data scientists can focus on feature engineering and interpreting the results. For users who lack ML expertise, finding the best combination of hyperparameters requires trial-and-error methods, which is a very tedious task. The idea of automated machine learning hyperparameter optimization has evolved in recent years due to efforts to automate hyperparameter optimization steps in the ML workflow due to the surge in the number of non-specialists using ML. This problem is solved by AutoML, which speeds up the process of determining the best values for the model's hyperparameters and enables data scientists to focus on algorithm development and performance enhancement.

1.1 Definition of Hyperparameters

In machine learning, "hyperparameters" are settings or variables that can be tweaked and adjusted to enhance the model's efficiency. They are predefined or set by the user before training, rather than being learned when a model is being trained. Because model values are fixed during training, hyperparameters are said to be "external to the model." Nonetheless, individuals can choose the values of the model hyperparameters, which can affect how a learning algorithm learns. The size of the training batch, the number of hidden layers in a neural network, the learning rate of an optimization technique, and other training-related factors are all regulated by these hyperparameters. During the training of the learning algorithm, several hyperparameters are utilized; however, these are not incorporated into the resultant model. After the learning process is complete, the trained model parameters, also known as models, are acquired. The hyperparameters used during training are not included in this model. Some of the common hyperparameters are learning rate, batch size, k in k-nearest neighbors, number of iterations, etc.

On the other hand, parameters are values that are iteratively changed after the model learns them during the training phase in order to lower the loss function. The weights and biases of each neuron in a neural network, for example, are parameters that are learned during training.

The key difference between hyperparameters and parameters is that hyperparameters are set by the user and remain constant throughout the training process, while parameters are learned by the model and change during training. The choice of hyperparameters can significantly influence the model's output, and finding good hyperparameters is often a crucial aspect of the machine-learning process.

1.2 Types of Hyperparameters

Based on their representation, hyperparameters can be categorized into two main types: ordered and categorical. Additionally, the ordered hyperparameters can be further divided into two subcategories: discrete, which is denoted by integers, and continuous, which is denoted by real numbers. Although it is possible to have ordinal variables that do not correspond directly to integers, they can usually be converted into integers using their index without losing any significant amount of information.

The type of hyperparameter can significantly impact the performance of hyperparameter optimization algorithms. Some hyperparameter optimization algorithms can natively handle all types of variables, whereas others mandate that all variables be converted to real numbers before execution. Performing transformations such as converting categorical variables into real numbers can lead to some loss of information, and it is essential to execute an appropriate transformation to obtain optimal results. When multiple categorical variables are present, algorithms that do not handle categorical variables may demonstrate relatively inferior performance. As a result, it is vital to consider this factor when comparing algorithms, both theoretically and experimentally.

1.3 Definition of Hyperparameter Optimization

Hyperparameter optimization is the process of selecting the best hyperparameters for a machine learning model to maximize its performance. Optimizing the hyperparameters of a machine learning algorithm is a critical part of the model-building procedure that is commonly accepted as standard practice [4]. Finding the best hyperparameters for a model can significantly improve its performance, as sub-optimal hyperparameters can lead to overfitting or underfitting, resulting in poor generalization. The goal of most machine learning algorithms is to develop a function between input and output that minimizes some loss. The algorithm learns this function by minimizing the loss of its predictions on a training dataset by modifying a set of parameters. Parameter formulation in the model is essential to the optimization process, whereas pre-training hyperparameter setting determines how the learning process will behave. Linear regression, a type of regression model with the following formulation, can be used to illustrate this idea:

$$y = \beta_0 + \beta_1 * x_1 + \beta_2 * x_2 + ...\beta_n * x_n \tag{1}$$

In linear regression, the value to be predicted is denoted by y, while x_i represents the feature i and β_i is associated with the coefficient i. Linear regression coefficients are the model's hidden parameters that are adjusted during training to reduce discrepancies between the y values predicted by the model and their corresponding characteristics in the training data. Before beginning the learning process, there are a few decisions to be made, such as which iterative algorithms to use to optimize the coefficients, how many iterations can be performed before the algorithms converge, and which type and degree of regularization will be used to improve generalization to new data. It's also important to note that the intercept component β_0 is optional in the model formulation. The basic purpose of machine learning algorithms is to minimize the loss on the validation dataset, and these hyperparameters are a major factor in doing so.

These hyperparameters have a major impact on the final loss on the dataset used for validation, so it's crucial to find the optimal values for them. Since there is no meaningful information on how the loss varies with respect to the hyperparameters, they cannot be automatically improved in the same way that the parameters of a learning algorithm can be. Therefore, different learning strategies need to be employed.

$$\lambda^* = argmin_{\lambda \epsilon \Lambda}(L(X_{validation}, A_{\lambda}(X_{train}))) \tag{2}$$

The optimization task is formally defined in Equation (2), which aims to find the optimal values of hyperparameters λ. The space of possible hyperparameter values is represented by Λ, and the loss function L takes in the dataset for validation $X_{validation}$ along with the function $A_{\lambda}(X_{train})$ estimated by algorithm A with hyperparameters λ after training on the training dataset X_{train}. The loss function is computed by comparing the results of the $A_\lambda(X_{validation})$ to the results of the $X_{validation}$.

The optimization of hyperparameters is different from that of a learning algorithm's parameters because it cannot be done automatically. This is due to the fact that there is no helpful data showing how the loss varies across the training and validation datasets as a function of the hyperparameters. Consequently, it is not possible to use the same learning strategies as with the algorithm's parameters because there is no differentiation of hyperparameters in relation to the datapoints.

1.4 Importance of Hyperparameter Optimization

Hyperparameter optimization helps to find the most suitable set of hyperparameters for a machine learning model, resulting in improved accuracy, precision, recall, and other performance metrics. A common issue in machine learning is overfitting, and hyperparameter optimization helps to avoid it by ensuring the model generalizes well on unseen data. Optimizing hyperparameters can lead to faster model training times by lowering the number of iterations required to attain convergence. Hyperparameter optimization can help reduce model complexity by selecting the most important hyperparameters and removing insignificant hyperparameters. Hyperparameter optimization can help create more robust models by ensuring that they perform well across a range of datasets. Hyperparameter optimization can give businesses a competitive advantage by creating more accurate and efficient machine learning models than their competitors. Hyperparameter optimization can help improve the machine learning models' scalability by ensuring that they can handle larger datasets and compute resources.

Mantovani et al. [5] investigated the significance of hyperparameter optimization, where the authors predicted whether hyperparameter optimization would result in improved outcomes compared to models built using default hyperparameters using meta-features, which are properties of the learning task. Furthermore, Lavesson and Davidsson [6] demonstrated that optimization hyperparameters are frequently more important than selecting a machine learning algorithm. Van Rijn and Hutter [7] investigated the significance of specific hyperparameters as well as the interactions between hyperparameters. Important hyperparameters are defined by the authors as parameters that explain the majority of the variation in performance across several datasets. A functional ANOVA framework is used to determine the explained variance. The results of this method do not directly convert into suggestions for which hyperparameters to modify, which is one of its limitations. For instance, a hyperparameter that has a single setting that consistently produces good results is crucial since it describes performance variance. However, substantial optimization is not necessary for such a value. Instead, it can be tuned to a setting that consistently produces positive outcomes. Needless to say, it is a critical step in any machine learning project because it leads to optimal model results.

1.5 Types of Hyperparameter Optimization

There are two approaches to optimizing the hyperparameters of machine learning models:

1.5.1 Manual Hyperparameter Optimization

Manual hyperparameter optimization entails manually experimenting with different sets of hyperparameters using intuition, experience, and the trial-and-error method. Manual hyperparameter optimization involves selecting a set of hyperparameters to test, training the model with those hyperparameters, and then evaluating the performance of the resulting model. This process is repeated with various sets of hyperparameters until the best combination is found. One can start by selecting a set of hyperparameters that one believes will work well based on their experience or knowledge of the problem domain. They can then train the model with those hyperparameters and evaluate its performance. If the performance is not satisfactory, one can adjust the hyperparameters and repeat the process until one finds the best combination.

Although manual hyperparameter optimization can enhance a machine learning model's performance, it can also be time-consuming due to the need to test numerous alternative combinations of hyperparameters and the high level of skill required. For manual hyperparameter optimization, a machine learning practitioner does not require a special library; instead, they must test various hyperparameter combinations for the model and choose the one that performs the best. Because there are so many trials to keep track of, manual optimization takes a lot of time and money. When there are fewer hyperparameters available, manual hyperparameter optimization is simple and quick to conduct, but it becomes extremely challenging and impractical when there are many hyperparameters to take into account. Hyperparameter optimization can be aided by automated tools and algorithms, which can speed up the process and enhance a model's functionality.

1.5.2 Automated Hyperparameter Optimization

Automated hyperparameter optimization is the process of using automated algorithms to determine the appropriate hyperparameters for a given machine learning model. Automated hyperparameter optimization involves selecting a range of hyperparameters to test and using an automated algorithm to search the hyperparameter space to find the best combination. The automated algorithm iteratively tests different combinations of hyperparameters and evaluates the machine learning model's performance. The procedure is repeated until the ideal set of hyperparameters is identified. For automated hyperparameter optimization, a user must define a set of hyperparameters and the values of those hyperparameters beforehand. The algorithm will then do the heavy lifting for you. It runs those trials and returns the best set of hyperparameters for optimal results.

Automated hyperparameter optimization can save time and improve the performance of a machine learning model compared to manual hyperparameter optimization. Automated hyperparameter optimization algorithms can explore a large hyperparameter space quickly and efficiently, saving valuable time and effort compared to manual optimization. Automated algorithms can find hyperparameters that may be difficult or impossible to discover manually, resulting in improved model performance. Manual hyperparameter optimization may be influenced by the user's bias or subjective opinions, whereas automated optimization is based purely on data-driven optimization, reducing the risk of bias. Automated hyperparameter optimization can be easily scaled to accommodate large datasets and complex models, which may not be feasible in manual optimization. Automated hyperparameter optimization enables the same hyperparameters to be used consistently across different experiments, leading to greater reproducibility of results. Automated hyperparameter optimization reduces the risk of human error, such as typos or incorrect parameter settings, which can adversely affect model performance.

There are several algorithms available for automated hyperparameter optimization, including grid search, random search, and Bayesian optimization. Grid search involves testing all possible combinations of hyperparameters in a grid-like pattern. The random search samples the hyperparameters from a predefined range in a random fashion. Bayesian optimization involves creating a probabilistic model of the model's performance based on previous evaluations and utilizing it to select the most promising hyperparameters to test next.

A more detailed overview of algorithms that perform automated hyperparameter optimization (HPO) is as follows:

A. Grid Search: HPO has traditionally utilized grid search, which involves exhaustively searching through a manually chosen subset of a learning algorithm's hyperparameter space. Nonetheless, this method is susceptible to the curse of dimensionality. It implies that the number of evaluations required for each additional hyperparameter or dimension in the search space grows exponentially. Consequently, this approach may become prohibitively computationally expensive, particularly for models with many hyperparameters that significantly impact performance. Despite its computational inefficiency, this strategy has traditionally been the preferred way for optimizing hyperparameters in both real-world applications and the field of machine learning research. Consider the results of a survey of all NIPS 2014 approved papers, which found that grid search was used in 82 of 86 instances where automatic hyperparameter search was used.

B. Random Search: A second straightforward strategy is to generate a grid of points, randomly assign trials to each cell in the grid, and then observe the results. The user does not specify a set of values for each hyperparameter but rather a range from which to draw search results, generally by randomly sampling within that range. A less expensive approach than grid search that randomly samples the search space to evaluate only a limited number of models, this technique is not only suitable for discrete settings but also extends to continuous and mixed spaces. Particularly when only a few hyperparameters have a major impact on the ML algorithm's final performance, it can outperform grid search. Random search can be a useful option when the lengths of the value vectors representing the hyperparameters that constitute the search space vary significantly. This means that certain hyperparameters have a vast number of potential values to experiment with, while others have only a few.

Figure 1 depicts an illustration of the preceding observation. In this two-dimensional example, one of the hyperparameters has a far larger impact on the outcome. The marginal distribution of the metric in relation to the hyperparameters is shown on the sides. In contrast to the three trial values for the significant hyperparameter that evenly spaced grid search generates out of a total of nine trials, random search generates nine trial values for it, helping us focus on a greater potential value for that dimension and, by extension, the overall hyperparameter settings.

C. Gaussian Processes: When it comes to Bayesian optimization, Gaussian processes (GPs) are frequently used as the default. By nature, they are non-parametric and are able to approximate complex functions. GPs establish a prior distribution over a set of functions (i.e., probability distributions with respect to the search space). During each iteration, the GP samples values from the search space as training data

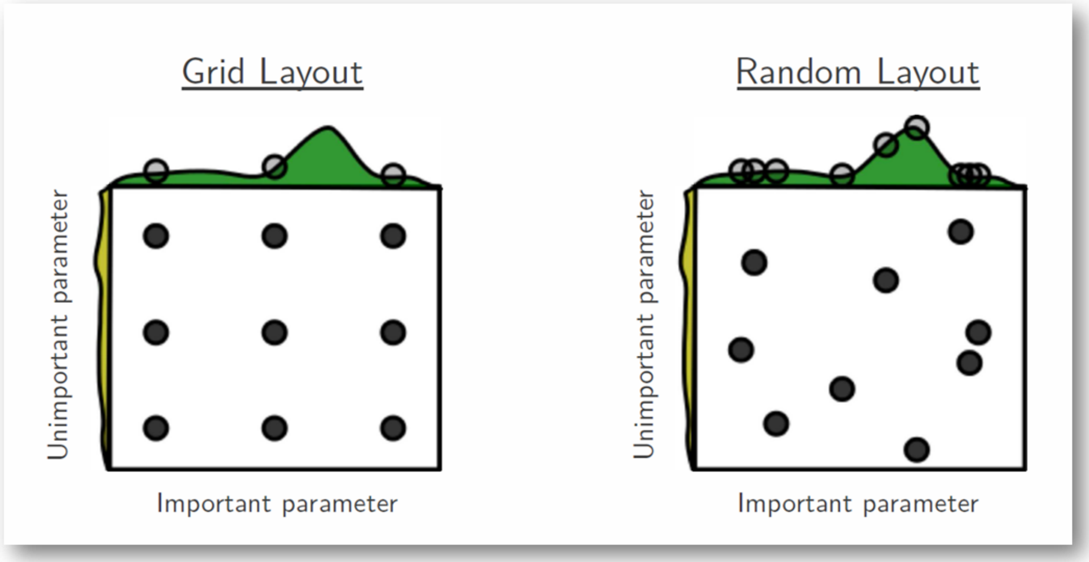

Figure 1: Grid Search vs Random Search (Bergstra and Bengio [1])

and applies a kernel function—one of several types—to obtain a posterior distribution. Gaussian distributions have the desirable property of being able to determine the joint probability of several variables, or hyperparameters, and the conditional probability of one hyperparameter given any other hyperparameter.

D. Tree Parzen Estimator (TPE): The TPE search algorithm is a Bayesian optimization algorithm used for hyperparameter optimization. It is an improvement over the original Parzen estimator method and was proposed by [2]. The TPE technique is used to optimize quantization hyperparameters to obtain the highest potential latency improvement and an estimated accuracy objective. TPE is an iterative process that constructs a probabilistic model based on past evaluations of hyperparameters and employs it to recommend the subsequent set of hyperparameters to evaluate. The TPE uses the Bayes method to calculate P(x|y) and P(y), where x stands for the related quality score and y for the hyperparameters. By altering the generative process of hyperparameters and substituting non-parametric densities for the configuration prior's distributions, P(x|y) is modeled. The key idea behind TPE is to use a tree structure to model the relationship between hyperparameters and performance. At each iteration, the algorithm uses the tree to select a new set of hyperparameters to evaluate based on the performance of previous evaluations. The tree structure allows TPE to quickly identify promising regions of the hyperparameter space and focus its search on those regions. It is particularly useful for high-dimensional hyperparameter spaces where an exhaustive search is infeasible.

2 Methods

In the present day, numerous AutoML hyperparameter optimization tools are available that incorporate advanced hyperparameter searching methods for finding hyperparameters for machine learning algorithms such as gradient boosting algorithms [8] [9] [10] [11] (Xgboost, Catboost, etc.). Among them, notable names include Raytune [12], Optuna [13], HyperOpt [14], SMAC [15], Spearmint [16], and many more. For this study, two of the most popular AutoML hyperparameter optimization tools, Optuna and HyperOPT, are utilized and compared based on their computational time and accuracy scores. In the comparison study, four popular OpenML datasets were utilized, including eye movement [17], gas concentration [18], gesture phase [19], and airline delay [20]. This section briefly describes the tools that were used.

2.1 Optuna

Optuna is a popular open-source framework for hyperparameter optimization that is widely employed in machine learning tasks. It's meant to be a versatile, user-friendly, and scalable toolkit for fine-tuning ML models' hyperparameters. Optuna utilizes a range of samplers, including grid search, random search, Bayesian optimization, and evolutionary algorithms, to automatically identify the most effective hyperparameter values. Optuna uses a variant of the TPE algorithm to search for the optimal hyperparameters for a particular machine learning model. The framework provides a flexible Application Programming Interface (API) that enables developers to define the search space of hyperparameters and the objective function to be optimized. Users can also set constraints on the hyperparameters and specify the type of optimization (minimization or maximization).

One of the key advantages of Optuna is its ability to efficiently handle high-dimensional search spaces. The framework uses a pruning technique to discard unpromising trials and focus on more promising ones, which reduces the computational resources required for hyperparameter optimization. The intermediate objective values are monitored for this purpose, and values that do not comply with defined criteria are stopped, which is allowed by an asynchronous successive halving algorithm. Optuna also provides built-in visualization tools that allow users to monitor the optimization process and analyze the results.

Optuna is developed to interface quickly with a variety of popular machine learning frameworks, including TensorFlow, PyTorch, and Scikit-Learn. To assist customers in analyzing and comprehending the results of optimization, it also offers a variety of visualization tools. Because of its extreme flexibility, Optuna enables users to design intricate search areas, restrictions, and conditional parameter distributions with just a few lines of code. Finding the best hyperparameters in complicated models may become simpler as a result. Moreover, Optuna offers distributed optimization, enabling customers to execute simultaneous hyperparameter optimization on many workstations, greatly accelerating the process. The framework also offers a number of customization options, including trial storage, pruning techniques, and user-defined search engines.

To assist customers in analyzing and comprehending the results of optimization, it also offers a variety of visualization tools. Because of its extreme flexibility, Optuna enables users to design intricate search areas, restrictions, and conditional parameter distributions with just a few lines of code. Finding the best hyperparameters in complicated models may become simpler as a result. Moreover, Optuna offers distributed optimization, enabling customers to execute simultaneous hyperparameter optimization on many workstations, greatly accelerating the process. The framework also offers a number of customization options, including trial storage, pruning techniques, and user-defined search engines.

2.2 HyperOpt

HyperOpt is a free and open-source Python library for optimizing hyperparameters. It can deal with both continuous and discrete hyperparameters and a wide variety of optimization tasks. HyperOpt offers other optimization techniques in addition to TPE, adaptive TPE, and random search. One of HyperOpt's important advantages is its support for distributed computing, which enables concurrent hyperparameter searches across numerous Central Processing Unit (CPU) cores or even machines. This drastically cuts down on the amount of time needed to find the best hyperparameters, especially for complex models or large-scale datasets.

Additionally, HyperOpt offers advanced features such as early stopping, which can help terminate the optimization process early if it is not making progress. HyperOpt also supports various integration options with popular machine learning libraries, including Scikit-Learn, Keras, and PyTorch. HyperOpt provides an intuitive and user-friendly interface for visualizing the results of hyperparameter searches and for selecting the best set of hyperparameters. Another important feature of HyperOpt is its ability to handle noisy or stochastic objective functions, which are common in machine learning. It uses Bayesian optimization to model the objective function and its noise and then uses the model to guide the search for optimal hyperparameters.

The core elements of HyperOpt are an objective function, a search domain, and an optimization method. Since the search domain can be specified by continuous, ordinal, or categorical variables, it offers more flexibility during the optimization process. An objective function can be any Python function defined by the user that takes in a set of variables and returns a loss function for that set of variables.

3 Datasets

In this study, four popular OpenML [21] tabular datasets, Eye Movement, Gas Concentrations, Gesture Phase, and Airline Delay, were used. Tabular datasets are collections of rows and columns, where columns represent the features and rows are the actual values. The datasets contain mixed-type data, which is a combination of both categorical and numerical data types. The numerical data type is the type of data that is expressed in terms of numbers rather than categorical descriptions, for example, age, weight, height, etc. On the other hand, a categorical data type is a type of data that can be stored into groups or categories, for example, ethnicity, gender, hair color, etc. More details, such as the number of target classes, size of the dataset, number of numerical features, and categorical features of the datasets, are described in Table 1.

Table 1: Dataset Details

Dataset	Classes	Size	Num. Feat.	Cat. Feat.
Eye movement	4	10.9k	24	4
Gas concentration	6	13.9k	129	1
Gesture phase	9	9.8k	32	1
Airlines Delay	2	539k	3	5

3.1 Eye Movement

Eye movement has attributes such as lineNo, assgNo, prevFixDur, firstfixDur, etc. The data is organized in a tabular format where features are represented as columns and feature vectors are represented as rows. Each time sequence is comprised of 22-dimensional feature vectors. This dataset consists of a number of assignments, each of which is made up of a question and ten sentences (news item titles). Five of the statements are unrelated to question (I), while just one of them is the right response to question (C). Although they don't directly address the question (R), four of the statements are related to it. The objective of the task is to make predictions on the classification labels (I, R, and C).

3.2 Gas Concentration

The data for gas concentration comes from sixteen chemical sensors subjected to six gases at varying concentrations. The concentration level to which the sensors were subjected for each measurement is included in this supplementary dataset to the Gas Sensor Array Drift Dataset. The objective of the challenge is to classify six gases at varying concentrations.

3.3 Gesture Phase

The gesture phase is composed of features taken from seven videos of people gesticulating. This dataset was created to study the task of gesture phase segmentation, which involves identifying the different phases within a gesture, such as preparation, stroke, and retraction. The gestures were chosen to be representative of a range of common human movements, including pointing, waving, and swiping. The objective is to classify phases as follows: D (rest position), H (hold), P (preparation), R (retraction), and S (stroke).

3.4 Airline Delay

Airlines include features such as airline, flight, airport, etc., and the task is to forecast whether a particular flight will be delayed based on the scheduled departure details. The target variable is the delay status of the flight, which is represented as a binary variable indicating whether the flight was delayed or not.

4 Experiments and Results

Table 2: Default Model Accuracy

Dataset	Xgboost Accuracy	Catboost Accuracy
Eye movement	0.7249	0.6970
Gas concentration	0.9957	0.9946
Gesture phase	0.9418	0.9170
Airlines Delay	0.6937	0.7048

In this study, two experiments were conducted to investigate the effect of different ways of dividing the dataset on the performance of hyperparameter optimization.

4.1 Experiment 1

For the first experiment, the datasets were randomly split into a training set (consisting of 80% of the data) and a testing set (consisting of 20% of the data) in a stratified way to ensure a proportional representation of each class in the target variable. The training set was used to train classifier models using the Python gradient boosting packages XGBoost and CatBoost. For each model, 5-fold cross-validation was applied to select the best hyperparameters from a predefined range of values. Cross-validation is a valuable technique in machine learning that allows us to evaluate the performance of a model on a limited dataset and is particularly useful for hyperparameter optimization. TPE and random search algorithms were utilized as optimization algorithms for AutoML tools Optuna and HyperOpt, as both are incorporated in both tools. The best hyperparameters for the models were obtained using this setup. After identifying the best hyperparameters, they were used for the final model training. The final training was performed using the entire initially split training set, and then the performance of the models was evaluated on the corresponding testing set using accuracy as the evaluation metric. This approach allowed for the assessment of the model's generalization ability and ensured its performance on new, unseen data. Another performance metric evaluated in this study is the total time taken for the optimization process, with the methods being assessed over 30 iterations.

4.2 Experiment 2

For the second experiment, each dataset was randomly split into three sets in a stratified way: a training set containing 70% of the data, a validation set containing 10%, and a testing set containing 20%. The same two machine learning models from Experiment 1 were selected for training and evaluation, using the same hyperparameters, evaluation metrics, and iterations. An initial validation set was utilized as an evaluation set during the training of the final model for evaluation purposes. The final model was assessed on the testing set, while an evaluation set was employed to monitor potential issues with overfitting or underfitting during the final training process.

To ensure the fairness of the comparison between the two experiments, the same machine learning algorithms, hyperparameter optimization algorithms, hyperparameters, and evaluation metrics were used in both experiments. Additionally, the same data preprocessing steps were applied to the dataset in both experiments.

In addition, Hydra [22], an open-source Python framework that streamlines the development of complex applications and research, was utilized in both experiments.

Table 3: Experiment 1 Performance Result

Dataset	Method	Xgboost		Catboost	
		Acc.	Time(in min)	Acc.	Time(in min)
Eye movement	HyperOpt Random	0.7590	37.35	0.7367	39.96
	HyperOpt TPE	0.7079	**33.28**	0.7550	**24.56**
	Optuna Random	**0.7660**	35.28	**0.7609**	25.38
	Optuna TPE	0.7568	38.31	0.7559	36.42
Gas concentration	HyperOpt Random	0.9960	74.02	0.9960	80.23
	HyperOpt TPE	0.9957	**61.68**	0.9957	**77.98**
	Optuna Random	0.9949	99.08	0.9960	79.27
	Optuna TPE	**0.9964**	110.28	**0.9964**	82.46
Gesture phase	HyperOpt Random	0.9417	**58.74**	0.9367	52.78
	HyperOpt TPE	**0.9529**	95.02	0.9387	**49.48**
	Optuna Random	0.9417	66.14	0.9362	51.08
	Optuna TPE	0.9448	84.61	**0.9402**	49.57
Airlines Delay	HyperOpt Random	0.7028	180.54	0.7070	75.49
	HyperOpt TPE	**0.7119**	239.64	0.7075	**74.73**
	Optuna Random	0.6927	**140.44**	0.7072	77.25
	Optuna TPE	0.7096	265.98	**0.7080**	88.35

The primary advantage is its capability to generate a hierarchical configuration dynamically, which can be modified via configuration files and command-line overrides. All the experiments were run using Simple Linux Utility for Resource Management (SLURM) on the Ohio Supercomputer Center (OSC) Pitzer Cluster [23] which has 40 and 48-core CPU nodes of dual Intel Xeon 6148s Skylakes and dual Intel Xeon 8268s Cascade Lakes processors with 192GB of memory.

Based on the result of Experiment 1, Table 3, it is noticeable that, in terms of accuracy, Xgboost performs better than Catboost on all four datasets, whereas Catboost is faster in most cases. Among the hyperparameter optimization tools, Optuna performs better than HyperOpt in terms of accuracy in almost all cases except in the

Table 4: Experiment 2 Performance Result

Dataset	Method	Xgboost		Catboost	
		Acc.	Time(in min)	Acc.	Time(in min)
Eye movement	HyperOpt Random	0.7244	**39.06**	0.7262	44.12
	HyperOpt TPE	0.7399	40.91	0.7299	**43.42**
	Optuna Random	**0.7522**	43.24	**0.7367**	46.61
	Optuna TPE	0.7313	44.73	0.7207	47.47
Gas concentration	HyperOpt Random	0.9953	91.76	0.9957	**70.72**
	HyperOpt TPE	0.9957	**77.92**	0.9957	90.10
	Optuna Random	**0.9960**	89.31	**0.9967**	80.07
	Optuna TPE	0.9946	79.98	0.9960	72.80
Gesture phase	HyperOpt Random	0.9311	**52.72**	**0.9271**	53.00
	HyperOpt TPE	0.9377	60.39	0.9240	**40.25**
	Optuna Random	0.9382	57.87	0.9235	50.41
	Optuna TPE	**0.9448**	103.3	0.9235	47.32
Airlines Delay	HyperOpt Random	0.6970	**138.73**	0.7059	76.55
	HyperOpt TPE	0.7006	209.61	**0.7075**	**74.73**
	Optuna Random	0.7009	251.75	0.7069	75.67
	Optuna TPE	**0.7038**	302.78	0.7049	84.18

case of Gesture phase Xgboost and Airlines Xgboost. In terms of time, HyperOpt is the fastest tool for hyperparameter optimization on all four datasets, except in the case of Airlines Xgboost. Also, Optuna with TPE and HyperOpt with TPE are effective combinations for achieving high accuracy and faster computation, respectively. In the figure, Figure 2 the X-axis shows the number of trials used during hyperparameter optimization for getting the best hyperparameters. The Y-axis shows the accuracy as a performance metric on the validation set during hyperparameter optimization. It is apparent from the figure that Optuna is more stable than HyperOpt.

From the data presented in Table 4, it can be observed that, while Catboost is faster in terms of computational time, Xgboost outperforms it in terms of overall accuracy. Among the hyperparameter optimization tools, Optuna performs better than HyperOpt in terms of accuracy in almost all cases except in the case of the Gesture phase Catboost and Airlines Catboost. In terms of time, HyperOpt is the fastest tool for hyperparameter optimization on all four datasets. From the accuracy vs. trials plot Figure 3, it is evident that Optuna is more stable than HyperOpt.

Overall, the experiments' results show that Optuna produced more accurate results than Hyperopt, while Hyperopt demonstrated faster computational times. Despite their differences, both frameworks succeeded in improving the machine learning models' performance, as demonstrated by the comparison between the default accuracy of models in Table 2 and the performance of optimized models.

5 Conclusion

In conclusion, the results from both experiments indicated that Optuna and HyperOpt are both effective hyperparameter optimization frameworks, although with different strengths. Optuna performed well in terms of accuracy, while HyperOpt excelled in terms of computational efficiency. Specifically, Optuna was found to produce better accuracy results than HyperOpt, while HyperOpt demonstrated faster computational times. These findings suggest that the choice of hyperparameter optimization framework should be based on the specific requirements of the task at hand, whether it prioritizes accuracy or speed. Nonetheless, results showed that both frameworks helped boost machine learning models' efficiency.

6 References

[1] James Bergstra and Yoshua Bengio. Random search for hyper-parameter optimization. *Journal of machine learning research*, 13(2), 2012.

[2] James Bergstra, Rémi Bardenet, Yoshua Bengio, and Balázs Kégl. Algorithms for hyper-parameter optimization. *Advances in neural information processing systems*, 24, 2011.

[3] Bernd Bischl, Martin Binder, Michel Lang, Tobias Pielok, Jakob Richter, Stefan Coors, Janek Thomas, Theresa Ullmann, Marc Becker, Anne-Laure Boulesteix, Difan Deng, and Marius Lindauer. Hyperparameter optimization: Foundations, algorithms, best practices, and open challenges. *Wiley Interdiscip. Rev. Data Min. Knowl. Discov.*, January 2023.

[4] Chris Thornton, Frank Hutter, Holger H Hoos, and Kevin Leyton-Brown. Auto-weka: Combined selection and hyperparameter optimization of classification algorithms. In *Proceedings of the 19th ACM SIGKDD international conference on Knowledge discovery and data mining*, pages 847–855, 2013.

[5] Rafael G Mantovani, André LD Rossi, Joaquin Vanschoren, Bernd Bischl, and André CPLF Carvalho. To tune or not to tune: recommending when to adjust svm hyper-parameters via meta-learning. In *2015 International joint conference on neural networks (IJCNN)*, pages 1–8. Ieee, 2015.

[6] Niklas Lavesson and Paul Davidsson. Quantifying the impact of learning algorithm parameter tuning. In *AAAI*, volume 6, pages 395–400. Citeseer, 2006.

[7] Jan N Van Rijn and Frank Hutter. Hyperparameter importance across datasets. In *Proceedings of the 24th ACM SIGKDD International Conference on Knowledge Discovery & Data Mining*, pages 2367–2376, 2018.

[8] Léo Grinsztajn, Edouard Oyallon, and Gaël Varoquaux. Why do tree-based models still outperform deep learning on tabular data? *arXiv preprint arXiv:2207.08815*, 2022.

[9] Yury Gorishniy, Ivan Rubachev, Valentin Khrulkov, and Artem Babenko. Revisiting deep learning models for tabular data. *Advances in Neural Information Processing Systems*, 34:18932–18943, 2021.

[10] Tianqi Chen and Carlos Guestrin. Xgboost: A scalable tree boosting system. In *Proceedings of the 22nd acm sigkdd international conference on knowledge discovery and data mining*, pages 785–794, 2016.

[11] Anna Veronika Dorogush, Vasily Ershov, and Andrey Gulin. Catboost: gradient boosting with categorical features support. *arXiv preprint arXiv:1810.11363*, 2018.

[12] Richard Liaw, Eric Liang, Robert Nishihara, Philipp Moritz, Joseph E Gonzalez, and Ion Stoica. Tune: A research platform for distributed model selection and training. *arXiv preprint arXiv:1807.05118*, 2018.

[13] Takuya Akiba, Shotaro Sano, Toshihiko Yanase, Takeru Ohta, and Masanori Koyama. Optuna: A next-generation hyperparameter optimization framework. In *Proceedings of the 25th ACM SIGKDD international conference on knowledge discovery & data mining*, pages 2623–2631, 2019.

[14] James Bergstra, Brent Komer, Chris Eliasmith, Dan Yamins, and David D Cox. Hyperopt: a python library for model selection and hyperparameter optimization. *Computational Science & Discovery*, 8(1):014008, 2015.

[15] Marius Lindauer, Katharina Eggensperger, Matthias Feurer, André Biedenkapp, Difan Deng, Carolin Benjamins, Tim Ruhkopf, René Sass, and Frank Hutter. Smac3: A versatile bayesian optimization package for hyperparameter optimization. *Journal of Machine Learning Research*, 23(54):1–9, 2022.

[16] Katharina Eggensperger, Matthias Feurer, Frank Hutter, James Bergstra, Jasper Snoek, Holger Hoos, Kevin Leyton-Brown, et al. Towards an empirical foundation for assessing bayesian optimization of hyperparameters. In *NIPS workshop on Bayesian Optimization in Theory and Practice*, volume 10, 2013.

[17] Jarkko Salojarvi, Kai Puolamaki, Jaana Simola, Lauri Kovanen, Ilpo Kojo, and Samuel Kaski. Inferring relevance from eye movements: Feature extraction. *Publications in Computer and Information Science*, 2005.

[18] Alexander Vergara, Shankar Vembu, Tuba Ayhan, Margaret A Ryan, Margie L Homer, and Ramón Huerta. Chemical gas sensor drift compensation using classifier ensembles. *Sens. Actuators B Chem.*, 166-167:320–329, May 2012.

[19] Renata CB Madeo, Clodoaldo AM Lima, and Sarajane M Peres. Gesture unit segmentation using support vector machines: segmenting gestures from rest positions. In *Proceedings of the 28th Annual ACM Symposium on Applied Computing*, pages 46–52, 2013.

[20] Chaitanya Manapragada, Heitor M Gomes, Mahsa Salehi, Albert Bifet, and Ge-

offrey I Webb. An eager splitting strategy for online decision trees in ensembles. *Data Mining and Knowledge Discovery*, 36(2):566–619, 2022.

[21] Joaquin Vanschoren, Jan N Van Rijn, Bernd Bischl, and Luis Torgo. Openml: networked science in machine learning. *ACM SIGKDD Explorations Newsletter*, 15(2):49–60, 2014.

[22] Omry Yadan. Hydra - a framework for elegantly configuring complex applications. Github, 2019.

[23] Ohio Supercomputer Center. Ohio supercomputer center, 1987.

Hyperparameters and search spaces

CATBOOST

- Learning rate: Log-Uniform distribution [e^{-5}, 1]
- Random strength: Discrete uniform distribution [1, 20]
- Max size: Discrete uniform distribution [0, 25]
- L2 leaf regularization: Log-Uniform distribution [1, 10]
- Bagging temperature: Uniform distribution [0, 1]
- Leaf estimation iterations: Discrete uniform distribution [1, 20]

XGBOOST

- Number of estimators: Uniform distribution [100, 4000]
- Eta: Log-Uniform distribution [e^{-7}, 1]
- Subsample: Uniform distribution [0.2, 1]
- Colsample bytree: Uniform distribution [0.2, 1]
- Colsample bylevel: Uniform distribution [0.2, 1]
- Max depth: Discrete uniform distribution [1, 10]
- Min child weight: Log-Uniform distribution [e^{-16}, e^{5}]
- Alpha: Uniform choice 0, Log-Uniform distribution [e^{-16}, e^{1}]
- Lambda: Uniform choice 0, Log-Uniform distribution [e^{-16}, e^{1}]
- Gamma: Uniform choice 0, Log-Uniform distribution [e^{-16}, e^{1}]

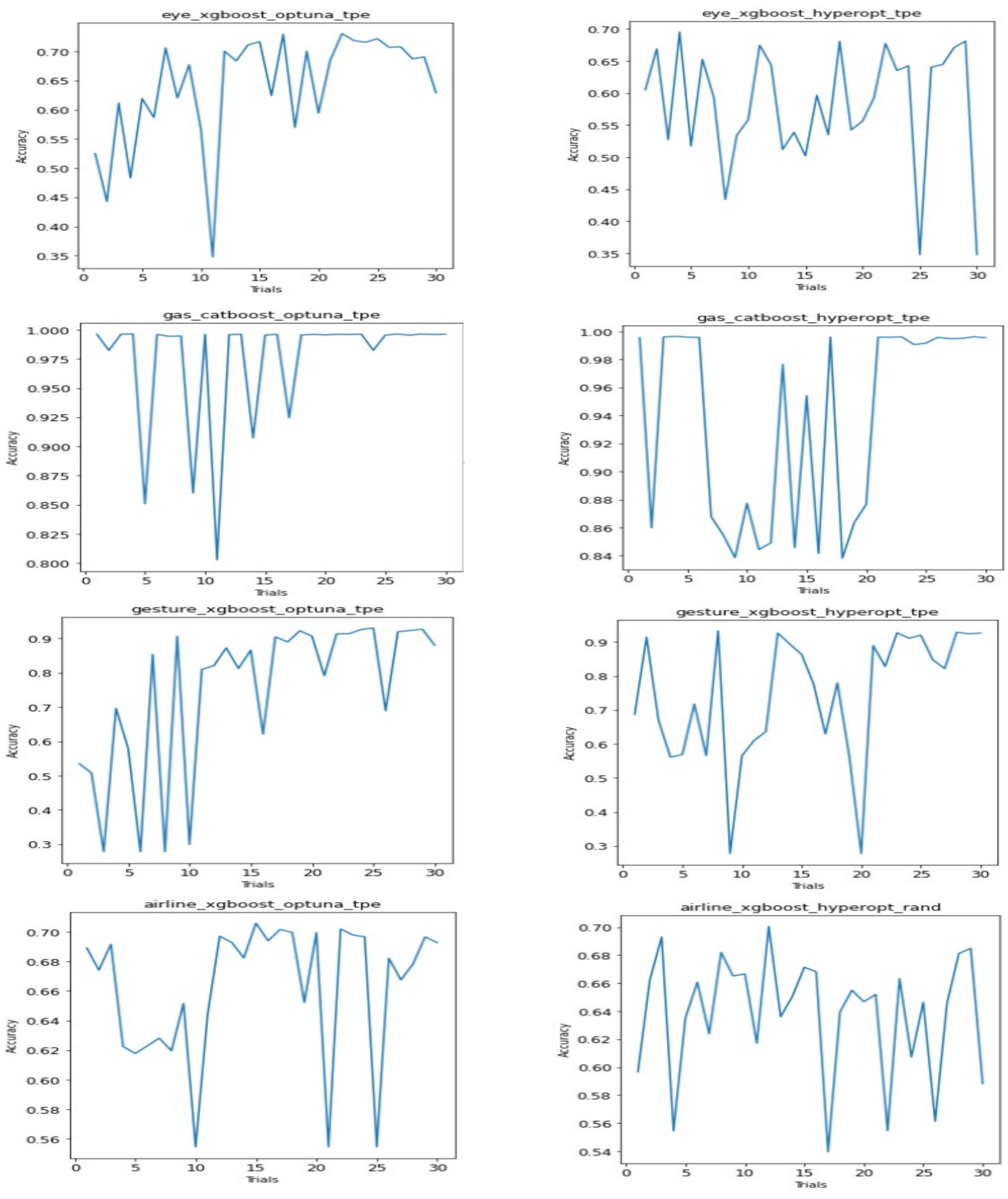

Figure 2: Experiment 1 Trial vs Accuracy plots

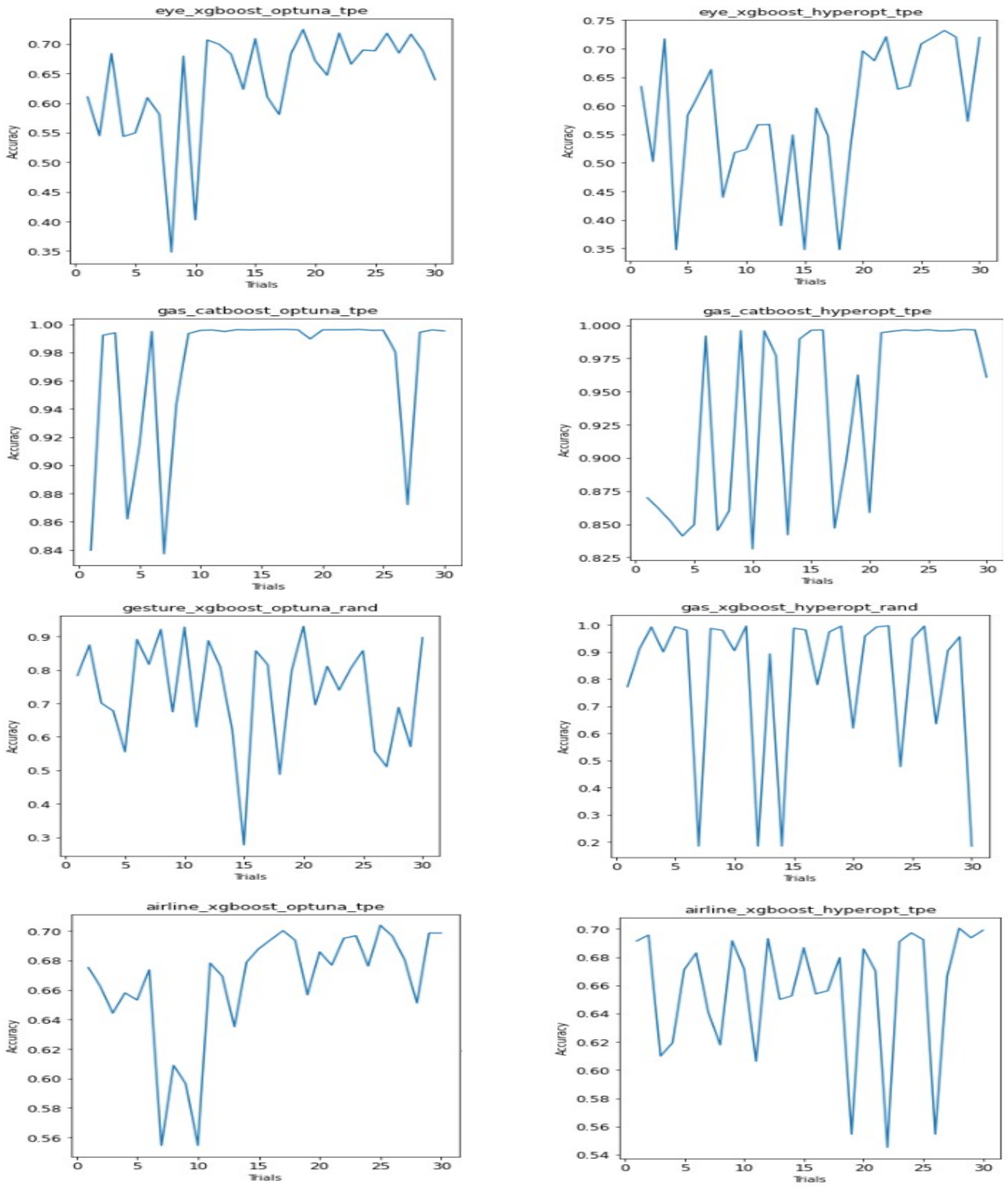

Figure 3: Experiment 2 Trial vs Accuracy plots