A Comparison of AutoML Hyperparameter Optimization Tools For Tabular Data

Prativa Pokhrel
Youngstown State University
Youngstown, Ohio, USA
ppokhrel03@student.ysu.edu

Alina Lazar
Youngstown State University
Youngstown, Ohio, USA
alazar@ysu.edu

Abstract

The performance of machine learning (ML) methods for classification and regression tasks applied to tabular datasets is sensitive to hyperparameters values. Therefore, finding the optimal values of these hyperparameters is integral in improving the prediction accuracy of an ML 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 that control the algorithm's behavior while building the model. Hyperparameter optimization (HPO) 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, we compare the performance of two frequently used AutoML HPO frameworks, Optuna and HyperOpt, on popular OpenML tabular datasets to identify the best framework for tabular data. The results of the experiments show that Optuna performs better than HyperOpt, whereas HyperOpt is the fastest for hyperparameter optimization.

Introduction and Background

Optimizing the performance of ML algorithms by searching for the best hyperparameter sets takes time and computing resources. Performing this process by hand is convoluted and time-consuming. The automation of the parameter search allows data scientists to focus on feature engineering and interpreting results parts. For users with less ML expertise, finding the best combination of hyperparameters requires trial and error, which is tedious. Nevertheless, studies have demonstrated that an optimal set of hyper-parameters enhances the model's performance. [2] [4]

The idea of automated ML hyperparameter tuning emerged in recent years due to efforts to automate the hyperparameter tuning step in the ML workflow due to the surge in the number of non-specialists using ML. AutoML addresses this issue and allows data scientists to focus on algorithm development and improving the model performance by finding optimal hyperparameters fast. The idea behind hyperparameter tuning is to add an optimization loop to the ML model learning process to identify the hyperparameters that result in the lowest testing error on the validation set. As a result, a validation set must be designated, and a loss must be specified.

Hyperparameters govern the learning process, and unlike parameters learned through model fitting or loss function optimization, hyperparameters cannot be inferred automatically. Furthermore, different methods may use distinct types of hyperparameters, and they may also have other effects on the performance of models. For instance, in the Catboost, the depth of the trees and the number of estimators are hyperparameters that can significantly affect the model's performance. At the same time, the least cost complexity pruning parameter might have an impact based on the amount of noise in the data. In this situation, most algorithms choose hyperparameters by involving the analyst. However, because the models are so sensitive to selecting hyper-parameters, it can be expensive to pick them from a group of algorithms.

There are various approaches or methods for performing HPO, such as:

A. Grid Search: HPO has traditionally utilized grid search, which involves exhaustively searching through a manually selected 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.

B. Random Search: A less expensive approach than grid search that randomly samples the search space to evaluate only a limited number of models. In particular, it can outperform Grid search, especially when only a few hyperparameters significantly influence the ML algorithm's final performance. 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 certain hyperparameters have many potential values to experiment with, while others have only a few.

C. Gaussian Processes: Regarding Bayesian Optimization, Gaussian Processes (GP) are frequently used as the default. In terms of nature, they are non-parametric and can approximate complex functions. GP establishes a prior distribution over a set of functions (i.e., probability distributions concerning the search space). During each iteration, the GP

samples values from the search space as training data and applies a kernel function - one of several types - to obtain a posterior distribution. Gaussian distributions have the desirable property of determining the joint probability of several settings (hyperparameters) and the conditional probability of one hyperparameter given any other hyperparameter.

D. TPE: The Tree-Structured Parzen Estimator (TPE) technique is made 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(xly) 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(xly) is modeled.

Methods

For finding hyperparameters for ML algorithms such as gradient boosting algorithms [7] [6] (Xgboost, Catboost, etc.), we use two most popular AutoML HPO Tools: Optuna [1] and HyperOpt [3]. These frameworks are compared based on their computational time and accuracy score. For the comparison study, we use four popular datasets: eye movement [10], gas concentration [13], gesture phase [8], and airline delay [9]. This section describes the two AutoML tools/frameworks we have used:

Optuna is an open-source framework for HPO that automates hyperparameter search. It automatically identifies the optimal hyperparameter values by leveraging different sampling methods, including random search, grid search, Bayesian optimization, and evolutionary algorithms. In addition, Optuna includes a pruning feature that allows you to terminate non-optimal runs early. To accomplish this, Optuna monitors the intermediate objective values and removes the ones that do not satisfy predetermined conditions. An asynchronous successive halving algorithm facilitates this process, which can utilize distributed computing capabilities.

HyperOpt is an open-source Python library for Bayesian optimization. A search space, an objective function, and an optimization method make up the core elements of HyperOpt. Since the search domain can be specified by continuous, ordinal, or categorical variables, it offers more flexibility during the optimization process. The objective function is a user-defined function that takes the variables and returns a loss function matching that variable combination. HyperOpt has TPE, adaptive TPE, and random search as options. Additionally, it offers the ability to use Apache Spark and MongoDB for parallel implementation.

Datasets

We used four popular OpenML [12] datasets, Eye Movement, Gas Concentrations, Gesture Phase and Airlines Delay, three of them are used in the "Deep Learning is not all you need" paper [11]. Dataset classes, size, etc are described in Table 1.

A. Eye movement has attributes such as lineNo, assgNo, prevFixDur, firstfixDur, etc. The dataset is divided into several assignments, each of which consists of a question followed by ten sentences. One of the sentences is the correct answer (C), and the other five are irrelevant to the question (I). Four of the sentences are related to the question (R), but they do not provide an answer. The task here is to predict the classification labels (I, R, C).

B. Gas concentration is made up of data from 16 chemical sensors that were exposed to six different gases at varying concentrations. This dataset includes information about the concentration level at which the sensors were exposed for each measurement. The task is to classify six gases at various concentration levels.

C. Gesture phase is made up of features extracted from seven videos of people gesticulating, with the goal of studying Gesture Phase Segmentation. The goal is to categorize the following phases: D (rest position), P (preparation), S (stroke), H (hold), and R. (retraction).

D. Airlines dataset includes features like Airline, Flight, AirportTo, and so on, and the task is to predict if a given flight will be delayed based on the scheduled departure information.

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

Experiments and Results

For experiments, we train classifier models using the Python Gradient Boosting packages XGBoost and CatBoost. We use TPE and random search methods as optimization methods or algorithms for both Optuna and HyperOpt, as both are incorporated in both tools. We split the dataset into 80% training and 20% testing data. We used the training set to perform cross-validation and evaluate the model's performance. We used cv=5 or 5 folds for cross-validation. This allowed us to obtain the best hyperparameters for our model. Once we identified the best hyperparameters, we used them for the final model training.

We performed the final training using the entire initially split training set. We then evaluated the model's performance or accuracy using the corresponding testing set. This approach enabled us to assess the model's generalization ability and ensure that it would perform well on new, unseen data. We also evaluate the total time taken for the optimization process as another performance metric. The methods are evaluated over 30 iterations. In addition, we utilized Hydra [14], an open-source Python framework that streamlines the development of complex applications and research. The

Table 2: Classification Performance on Testing Data Sets

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

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 SLURM on the Ohio Supercomputer Center (OSC) Pitzer Cluster [5] that 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 Table 2 result, we see that, in terms of accuracy, Xgboost performs better than Catboost on all four datasets. Among the hyperparameter optimization tools, Optuna performs better than HyperOpt in terms of accuracy in almost all cases except in the 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. From the figure, Figure 1 we can see that Optuna is more stable than HyperOpt.

Conclusions

In conclusion, both HyperOpt and Optuna are great automated HPO tools for optimizing models with different datasets. However, after conducting extensive training with both tools on four datasets and two models (Catboost and XGBoost), Optuna performs better than HyperOpt in terms of accuracy on most cases, whereas HyperOpt is the fastest tool for hyperparameter optimization for the most part. Optuna was found to be user-friendly and very easy to use due to its documentation and tutorial examples. Overall, our experimentation has demonstrated the usefulness of these packages in improving model performance.

Hyperparameters and search spaces

A list of hyperparameters and their search spaces used for Catboost and Xgboost respectively.

CATBOOST

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

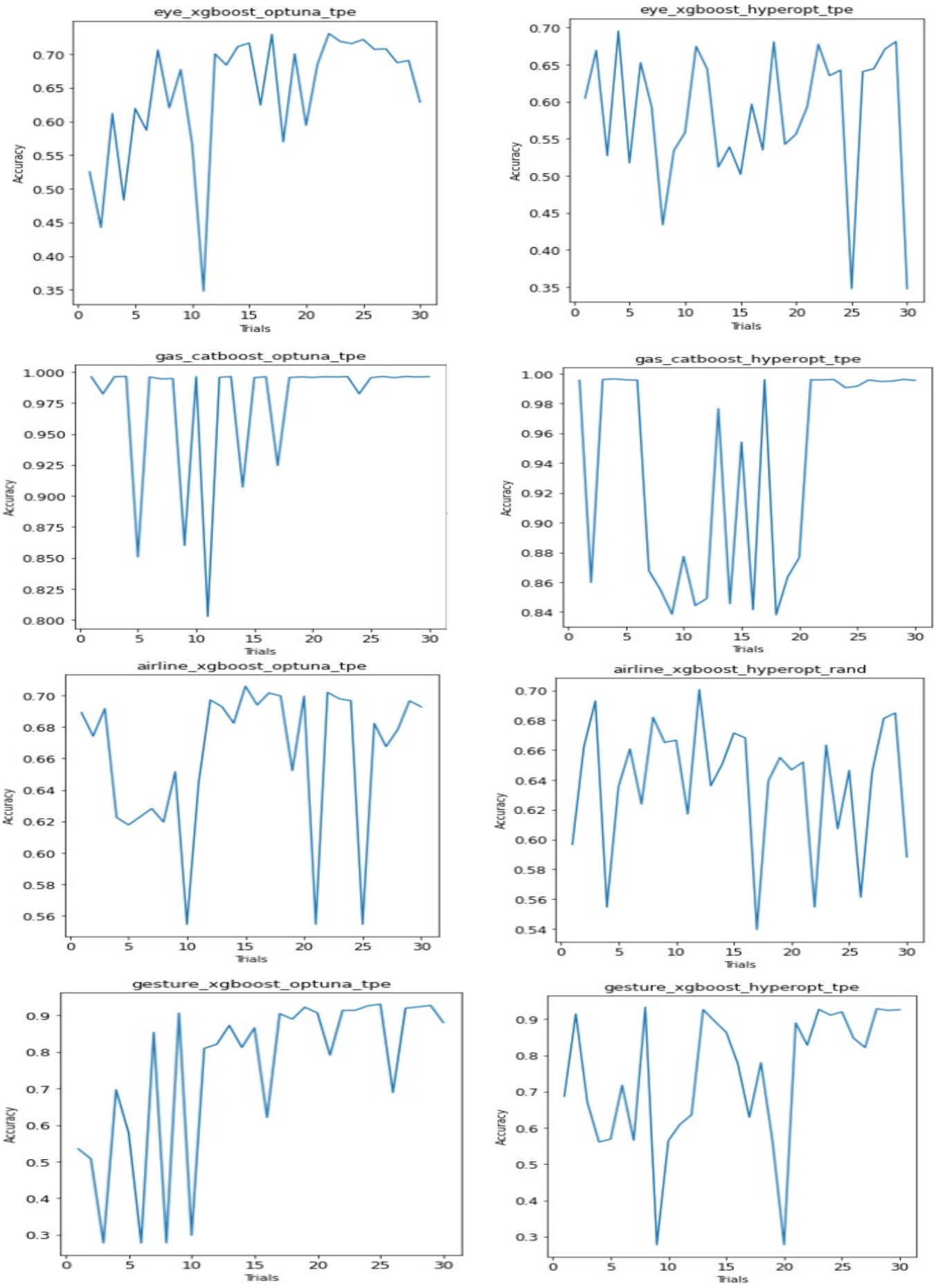

Figure 1: Trial vs Accuracy plots

References

[1] Akiba, T.; Sano, S.; Yanase, T.; Ohta, T.; and Koyama, M. 2019. Optuna: A next-generation hyperparameter optimization framework. In *Proceedings of the 25th ACM SIGKDD international conference on knowledge discovery & data mining*, 2623–2631.

[2] Bergstra, J.; Bardenet, R.; Bengio, Y.; and Kégl, B. 2011. Algorithms for hyper-parameter optimization. *Advances in neural information processing systems* 24.

[3] Bergstra, J.; Komer, B.; Eliasmith, C.; Yamins, D.; and Cox, D. D. 2015. Hyperopt: a python library for model selection and hyperparameter optimization. *Computational Science & Discovery* 8(1):014008.

[4] Bischl, B.; Binder, M.; Lang, M.; Pielok, T.; Richter, J.; Coors, S.; Thomas, J.; Ullmann, T.; Becker, M.; Boulesteix, A.-L.; Deng, D.; and Lindauer, M. 2023. Hyperparameter optimization: Foundations, algorithms, best practices, and open challenges. *Wiley Interdiscip. Rev. Data Min. Knowl. Discov.*

[5] Center, O. S. 1987. Ohio supercomputer center.

[6] Gorishniy, Y.; Rubachev, I.; Khrulkov, V.; and Babenko, A. 2021. Revisiting deep learning models for tabular data. *Advances in Neural Information Processing Systems* 34:18932–18943.

[7] Grinsztajn, L.; Oyallon, E.; and Varoquaux, G. 2022. Why do tree-based models still outperform deep learning on tabular data? *arXiv preprint arXiv:2207.08815*.

[8] Madeo, R. C.; Lima, C. A.; and Peres, S. M. 2013. Gesture unit segmentation using support vector machines: segmenting gestures from rest positions. In *Proceedings of the 28th Annual ACM Symposium on Applied Computing*, 46–52.

[9] Manapragada, C.; Gomes, H. M.; Salehi, M.; Bifet, A.; and Webb, G. I. 2022. An eager splitting strategy for online decision trees in ensembles. *Data Mining and Knowledge Discovery* 36(2):566–619.

[10] Salojarvi, J.; Puolamaki, K.; Simola, J.; Kovanen, L.; Kojo, I.; and Kaski, S. 2005. Inferring relevance from eye movements: Feature extraction. *Publications in Computer and Information Science*.

[11] Shwartz-Ziv, R., and Armon, A. 2022. Tabular data: Deep learning is not all you need. *Information Fusion* 81:84–90.

[12] Vanschoren, J.; Van Rijn, J. N.; Bischl, B.; and Torgo, L. 2014. Openml: networked science in machine learning. *ACM SIGKDD Explorations Newsletter* 15(2):49–60.

[13] Vergara, A.; Vembu, S.; Ayhan, T.; Ryan, M. A.; Homer, M. L.; and Huerta, R. 2012. Chemical gas sensor drift compensation using classifier ensembles. *Sens. Actuators B Chem.* 166-167:320–329.

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