Sigma Journal of Engineering and Natural Sciences
Web page info: https://sigma.yildiz.edu.tr
DOI: 10.14744/sigma.2026.2033

Research Article

Hyperparameter optimization web tool: Hyperopt

Fatma ALTINSOY[1,*], Muhammed Maruf ÖZTÜRK[2]

[1]*Graduate School of Natural and Applied Sciences, Suleyman Demirel University, Isparta, Türkiye*
[2]*Department of Computer Engineering Faculty of Engineering and Natural Sciences, Suleyman Demirel University, Isparta, Türkiye*

ARTICLE INFO

Article history
Received: 04 September 2024
Revised: 08 November 2024
Accepted: 27 February

Keywords:
Hyperparameter Optimization; Machine Learning; Optimization Techniques Stack Overflow Dataset; Web-Based Tool

ABSTRACT

This study introduces a web-based application designed to facilitate hyperparameter optimization for machine learning models, leveraging data from the stack overflow dataset. A primary contribution of this research is the development of a novel hyperparameter optimization method, integrated alongside established techniques such as Grid Search, Random Search, Nelder-Mead, and Bayesian Optimization. This integration provides users with the flexibility to explore various optimization strategies and identify the most suitable approach for their specific datasets and models.
The proposed web application enables users to select datasets, define optimization methods, and fine-tune hyperparameters through an intuitive and user-friendly interface. Empirical results demonstrate that the optimized models achieved a 15% improvement in prediction accuracy, attaining approximately 70% accuracy in predicting coding expertise and programming languages. Furthermore, the Proposed Method enhanced memory efficiency by 20% in SVM-based optimizations, with only a modest 10% increase in computational time. These findings underscore the method's effectiveness in balancing accuracy with resource efficiency.

Cite this article as: Altınsoy F, Öztürk MM. Hyperparameter optimization web tool: Hyperopt. Sigma J Eng Nat Sci 2026;44(2):1219–1239.

INTRODUCTION

Maintenance phases in software engineering like debugging, refactoring, and performance improvement are vital for maintaining the trustworthiness, extendibility, and productivity of software systems [1]. Through these methods, the functionalities of the current systems are not only upgraded but also the systems become more capable of adapting to changing requirements and technologies [2]. Among these, one critical performance optimization technique that has gained prominence, particularly in machine learning, is hyperparameter optimization [3].

Hyperparameter optimization is probably one of the most critical steps in further improving model performance in any machine learning project, considering the big data era we live in. In a nutshell, hyperparameters are important settings or options that are not directly learned during the model's training but significantly impact the model's success. Selecting the right hyperparameters leads to enhancing the overall performance and flexibility of the model to

*Corresponding author.
*E-mail address: fatmaaltinsoy@sdu.edu.tr

This paper was recommended for publication in revised form by Editor-in-Chief Ahmet Selim Dalkilic

new data [4]. However, in most cases, the hyperparameter search space is so big and complex that it is very time-consuming and expensive to find the optimal combinations for a particular problem or dataset [5]. Hence, an effective strategy for hyperparameter optimization is highly desired to improve model accuracy while keeping computational resources inefficient.

Among various hyperparameter optimization strategies, widely used methods in machine learning models are Grid Search, Random Search, and Bayesian Optimization [6]. However, such approaches cannot be handled more straightforwardly in big search spaces. The curse of dimensionality, a phenomenon where the computational complexity and data sparsity grow exponentially with the number of hyperparameters, exacerbates the difficulty of searching large spaces efficiently [7]. This challenge requires deep technical expertise, which is cumbersome to provide for big datasets or model architectures where the associated computational costs are truly substantial. Current solutions of hyperparameter optimization are, most of the time, not user-friendly and tend to be problematic for novices to use [8]. This again highlights the demand for platforms that can make hyperparameter optimization processes more accessible with time and resource efficiency.

Web-based tools have become popular for handling such challenges, largely because they are easy to reach, simple to use, and cut down on repetitive setup work. Platforms like Dlib-ml [9], Teachable Machine [10], and H2O.ai [11] let users run data analysis, train models, and perform optimization with little or no coding. Building on this direction, we introduce a web-based hyperparameter optimization platform aimed at a wide audience, from beginners to experienced practitioners. The tool pairs a clean interface with strong optimization methods, so users can spend their time on model performance rather than setup.

Two goals drive this study: designing a new hyperparameter optimization method and testing it on a real-world dataset, the Stack Overflow (SO) platform. We also built a web-based tool to make the method easier to use in practice. SO holds a large body of developer-focused content and offers clues about programming languages, coding habits, and expertise levels [12]. That said, its sheer size and frequent mislabeling make finding the right information difficult [13]. We propose a rule-based labeling method to estimate developer experience levels, which then feeds into better content sorting and recommendation.

The platform is built on the ASP.NET Core MVC framework, with R handling the backend computation. Users can pick a machine learning algorithm, set hyperparameters, and choose an optimization strategy from the interface. Results appear in both visual and numerical form, keeping the process open and easy to follow. Taken together, the study connects optimization theory with day-to-day use, offering a practical tool for hyperparameter tuning in both research and industry settings.

This study is devised to develop an easy-to-use, web-based hyperparameter optimization platform. Specifically, the main contributions are as follows:

1. Enabling users to select datasets, specify optimization strategies, and perform hyperparameter tuning through a web interface.
2. Proposing a heuristic optimization method that improves the model's accuracy and efficiency.
3. A rule-based labeling of coding experience and preferred languages applied to the SO dataset, which makes the data more useful for future work.

The paper continues as follows. Section 2 covers related work and situates this study among existing approaches. Section 3 presents the preliminaries, and Section 4 describes the method in detail. Results and discussion, along with their implications, are presented in Section 5. Finally, Section 6 concludes the study and outlines potential avenues for future research.

RELATED WORK

SO is a large online community in which many types of information are shared with developers, ranging from programming languages to data analysis [14]. The study of the impact of SO on social media has indicated that some features, such as the transparency of information sharing, are likely to cause complex emotional outcomes for developers [15]. Other works have focused more on the reliability of SO as a source of information and the dissemination of knowledge across software development communities [16]. However, due to the huge amount of information available on SO and the number of users mislabeling, access to valid information is rather complicated, which ultimately reduces the overall efficiency [17].

Various data mining and deep learning techniques have been proposed to answer the labeling issues in a large platform like SO: for example, Sonam et al. [18] proposed the TagStack framework to enhance the accuracy of tag prediction, while Harrag and Khamliche [19] proposed the deep learning-based system intended for the correction of programming errors. Among the recent tag prediction methods, the one suggested in a work by He et al. [20], relying heavily on the significance of the title and demonstrating state-of-the-art language models applied in this domain such as CodeBert, is excellent.

High-value answer platforms represent indispensable tools that solve development problems and are the repository of development techniques [21]. Chen et al. [22], in their work related to code reuse, put the accent on the interactions between platforms like GitHub and SO and showed how these platforms have gained an essential place in the software development process.

Hyperparameter optimization is a crucial process for improving the performance of machine learning models [23]. The classical methods have ranged from Grid and Random Search; even the more advanced ones like Bayesian

Optimization have been employed in recent literature [24]. Grid Search is costly as it tests all parameters, while Random Search tries random combinations, and Bayesian Optimization optimizes the search process through inferential reasoning based on previous predictions [25]. The limitations of these methods make them computationally expensive for large datasets. While other alternative methods have been developed, searching for effective solutions in large hyperparameter spaces is still an ongoing process [26].

Hybrid optimization approaches have drawn attention in recent work [27, 28]. Uslu et al. [29], for example, combined Ant Colony Optimization (ACO) with Genetic Algorithms (GA) and reported strong results on tasks with limited computing resources. The Continuous Genetic Algorithm (CGA) has likewise been used on complex systems such as Troesch's and Bratu's problems, which points to its usefulness for machine learning tasks including hyperparameter tuning [30, 31]. Work of this kind helps frame how combinatorial optimization problems can be tackled and opens new directions for hyperparameter optimization [32].

A number of libraries and AutoML tools aim to simplify hyperparameter tuning. Widely used options include Optuna [33], Google Vizier [34], Ray Tune [35], and Microsoft NNI [36], each supporting a range of algorithms and optimization strategies. The catch is that most of them are built as libraries, meaning users have to write code to use them. That limits their reach among people without much programming background.

FairerML [37] is another notable contribution. It is an extensible platform for studying, visualizing, and reducing bias in machine learning models, with a simple upload-and-analyze workflow for both data and models. It also supports joint modeling for multi-target learning, letting users weigh accuracy and fairness at the same time.

DebiAI [38] takes a similar direction. As an open-source tool, it covers analysis, visualization, evaluation, and comparison of machine learning models. Users can work with their data and metadata in several ways, from selection and editing to annotation, and the tool also supports contextual evaluation and bias detection.

LAMDA-SSL [39] is a user-friendly and powerful open-source toolkit offering comprehensive functionality,

Table 1. Comparison of the proposed platform with other AutoML tools

Criterion	AutoGluon	Scikit-learn	Auto-sklearn	TPOT	H2O.ai	Proposed platform
Ease of Use	Easy to use, offers automatic optimization.	Simple API, manual optimization.	Automatic model and hyperparameter selection.	Moderate ease, some settings require effort.	Easy to use with API and web support.	Simplified web interface, accessible to novices.
Performance	Strong across various data types.	Varies by data/model selection.	Strong performance but resource-intensive.	Good but longer training times.	High speed on large datasets.	High accuracy on SO-specific data, optimized models.
Flexibility	Broad model and data support.	Flexible for diverse applications.	Extensive algorithm support.	Automatic feature engineering.	Supports machine and deep learning models.	Supports SO tagging, and multi-label classification.
Resource Usage	High resource usage is possible.	Low resource usage.	Resource-intensive in long processes.	Computation-intensive.	Efficient with distributed computing.	Efficient SO focused optimizations with low memory.
Hyperparameter Optimization	Fully automatic.	Manual, user-controlled.	Automatic and effective.	Genetic algorithms for optimization.	Automatic and efficient.	Heuristic optimization integrated with user controls.
Community and Support	Active community, extensive docs.	Very large, comprehensive docs.	Benefits from scikit-learn ecosystem.	Active, good documentation.	Large community, commercial support available.	Community friendly, SO data focus.
Data Processing	Automatic processing, some features.	Tools are available but manual tweaks are.	Automatic processing/ selection.	Automatic feature engineering.	Automatic feature engineering.	Rule-based SO tagging, TDM matrix generation.
Use Case Relevance	General AutoML.	General AutoML.	General AutoML.	General AutoML.	General AutoML and distributed computing.	Specific to SO, improves data labeling and recommendations.

intuitive interfaces, detailed documentation, and extensive support for algorithms, data types, and tasks compared to similar tools. The goal of LAMDA-SSL research is to make SSL research more feasible and widely applicable, particularly given the scarcity of labeled data.

Additionally, another notable work in the literature is MLweb [40], an open-source software toolkit for web-based machine learning. MLweb's distinguishing feature is that all computations are performed client-side without sending data to a third-party server. Key components of MLweb include a JavaScript API for scientific computations (LALOLib), extensions of this library with machine learning tools (ML.js), and an online development environment with numerous examples (LALOLab).

Another noteworthy project is WebDraw [41], an automation tool developed to enhance the efficiency of web designers. It produces mock-up designs and working websites from screenshots of web pages. The process runs in three stages: GUI element detection, classification, and code generation. WebDraw is designed with a user-friendly interface and is used to improve industrial web design and development processes.

Work in this area shows a mix of approaches to hyperparameter optimization and related machine learning problems. Tools and platforms built for hyperparameter optimization sit at the center of this space. That said, most are shipped as libraries with a narrow set of optimization methods. Table 1 sets HyperOpt, our proposed tool, against popular AutoML options: AutoGluon [42], scikit-learn [43], Auto-sklearn [44], TPOT [45], and H2O.ai [11]. The AutoML tools lean toward automation and ease of setup, whereas our platform is built for flexibility and customization, especially on domain-heavy data like the SO dataset. HyperOpt brings several practical gains: lighter resource use, optimization strategies suited to the task, and an interface that handles complex multi-label datasets without friction.

We address this gap with an interactive web-based platform that handles hyperparameter optimization through the interface itself, so users do not have to touch scripts. The platform runs on ASP.NET Core MVC and lets users pick a machine learning algorithm, set hyperparameters, and choose an optimization strategy. Backend optimization is handled by R, and results come out in both visual and numerical form, which supports qualitative and quantitative review side by side.

BACKGROUND

Web Platform Development

In this research, the developed web application was created based on the ASP.NET Core MVC framework. The main benefit of such an MVC architecture is a clear line of separation between UI and business logic, which itself interacts with data. The above-mentioned configuration makes this suitable for handling machine learning algorithm hyperparameter optimization processes through inputs from the user and offers the ability to run R scripts that work in the background.

Users can select datasets, hyperparameters, machine learning algorithms, and optimization methods from the web interface. Based on the selected parameters, respective R scripts are selected to be executed by passing the arguments to the R interpreter from the operating system command line using the Process class in the System.Diagnostics namespace of .NET. These scripts are run, their outputs captured, error-checked, and presented to the user as results of the analysis: graphical and numerical. Further, this makes the hyperparameter optimization process more intuitive and accessible to both machine learning experts and non-expert users with lesser programming backgrounds.

This paper has used an extensive optimization process to determine the hyperparameter settings of various machine learning algorithms, including SVM, GBM, EBR, EPS, and Random Forest. In order to maximize the performance of each algorithm, different optimization techniques have been used, including Grid Search, Random Search, Bayesian Optimization, Nelder-Mead, and a newly developed optimization method.

The goal, therefore, is to find the θi associated with a model M_i that maximizes the accuracy performance of such a model, evaluated as $\hat{\theta}_i$ assessed as a vector of hyperparameters. This can be done through the following generalized optimization problem:

$$\hat{\theta}_i = \underset{\theta i \in \ominus i}{\operatorname{argmax}} \text{ Accuracy } (\text{Mi } (\theta i), \text{Dtrain}) \quad (1)$$

Here, $\hat{\theta}_i$ represents the set of hyperparameters that maximizes the accuracy performance of the model. The argmax expression aims to find the value of θi that provides the highest accuracy. The hyperparameter search space is denoted as $\theta i \in \ominus i$, which specifies the range within which each hyperparameter is evaluated. Additionally, the expression Accuracy (Mi (θi), D_{train}), represents the accuracy of model Mi trained with a specific set of hyperparameters on the training dataset D_{train}.

The optimization processes have been carried out using the R programming language, and interfaces have been created particularly for optimized models using caret [46], rBayesOptimization [47], gbm [48], e1071 [49], randomForest [50], and other relevant R packages. Numerous experiments have been designed to evaluate the performance of each algorithm. The optimization process followed these steps:

1. We split the data into two subsets, Dtrain (training set) and Dtest (test set), to keep training and evaluation separate.
2. The search space θsearch was set to define where hyperparameters could be tuned.

3. Each θ (hyperparameter) went through cross-validation during training, so we could check how stable the results were.
4. Performance metrics were then computed for every model that came out of this process.
5. Execution time and memory usage (Ptime, Pmemory) were tracked throughout, since working within resource limits was part of the problem.
6. Finally, performance metrics and the impact of hyperparameters were plotted using the ggplot2 [51] package.

Another side of this work is the careful use of computing resources. This matters for large datasets and for scaling up machine learning models. Tracking CPU and memory usage gives a clearer picture of what each algorithm actually costs to run. Algorithms that hit the CPU hard can drag down performance and scalability, and memory-heavy operations tend to choke on big data.

Looking at these metrics lets us tune implementations, clean up workflows, and place computing resources where they are needed most. The upshot is a setup that scales, holds up under load, and remains workable for machine learning tasks, even when the data volume and computing demands are large. For each hyperparameter setting, we also generate performance curves, which make it easier to see how parameter changes shape model behavior and to guide choices during optimization.

Optimization Techniques

Grid Search consists of hyperparameter group H and all possible values of the hyperparameters ξ. Then a comprehensive search algorithm is employed, and by using the Cartesian product of ξ, a parameter grid is made. This grid covers all possible combinations of hyperparameters [52]. The complexity of Grid Search is $O(k^n)$, where n is the number of hyperparameters and k represents the search range defined by the user. The complexity of Grid Search increases as the number of hyperparameter combinations grows. Consequently, it may become costly when dealing with a relatively large search space. In Equations 2-4, the hyperparameter group H, the set of possible values ξ, and the grid structure of the algorithm are defined, respectively.

$$H = h_1, h_2, \ldots, h_3 \tag{2}$$

$$\xi = \xi_1, \xi_2, \ldots, \xi_m \tag{3}$$

$$\text{Grid} = h_1 * \xi, h_2 * \xi, h_3 * \xi \ldots, h_n * \xi \tag{4}$$

In Random Search, a probability function f takes a hyperparameter h as an argument. f(h) is expressed as random sampling.

$$\phi = (h_1, \xi_1), (h_2, \xi_2), \ldots, (h_n, \xi_n) \tag{5}$$

In Equation 5, $\phi \sim f(h_i)$ is defined for i=1,2,…,n [53]. This random sampling determines the hyperparameter range $H = h_1, h_2, ..., h_n$ for tuning. Random Search skips the exhaustive sweep that Grid Search does. It works on a user-defined subdomain of H, where values are drawn randomly from H itself.

Its time complexity is O(p), with p being the number of iterations. Note that p_h is not necessarily equal to p, since some iterations can land on values already tried earlier. This makes Random Search faster than Grid Search and lighter on memory, which helps most on tuning problems of modest size.

Bayesian Optimization looks for the minimum or maximum of an objective function, aiming to get as close to the optimum as it can [54]. Its complexity is $O(f * n^2)$, where n is the number of hyperparameters and f is the cost of evaluating them. It belongs to the family of black-box optimization methods and picks candidate values from the search space through a probability model.

This model compares previous results with new ones to explore the parameter space effectively, enabling the achievement of the global optimum with fewer trials. However, tuning the cost depends on the number of values defined in the search space, which could result in a high cost due to the curse of dimensionality. The acquisition function f is used to find the global optimum x∗ as shown in Equation 6 where $A \subseteq R$ [55].

$$x^* = \text{argmax}_{x \in A} f(x) \tag{6}$$

Here, x* represents the best solution value that corresponds to the maximum value of the objective function. This value is the optimal hyperparameter combination found within the scope of the optimization problem. A stands for the solution space under search and holds all possible ranges that the hyperparameter values can fall into. The objective function f(x) measures how well each hyperparameter combination performs on the model, and it is this function that guides the search toward better-performing hyperparameters. What Bayesian Optimization really seeks is the point x in A where f(x) reaches its maximum, which in practice means finding the best hyperparameter combination for the problem at hand.

On the other hand, Nelder-Mead [56] is specially designed for stochastic responses. In this method, candidate points form the set *S* and are provided to a function as shown in Equation 7.

$$f : S^n \rightarrow S \tag{7}$$

Here, S^n represents the set of candidate points in n-dimensional space, while the function f maps these points to another set S. This mapping implies that the function takes a point from the set S^n as input and returns a different point in the set S as output. The Nelder-Mead method is not classified as a direct search method because it does not utilize gradient information. The complexity of Nelder-Mead is $O(n^3)$, where n represents the number of variables used for the optimization. It updates the objective function

by comparing the previous result with the next one, gradually modifying an optimization triangle throughout the remaining optimization steps. Nelder-Mead is well-suited for small-scale optimization problems, although it performs more slowly compared to its counterparts.

The Proposed Method and proofs provide a comprehensive mathematical foundation for optimization methods. The Proposed Method operates through a unique mechanism: the execution cost of tuning steps is closely related to the number of iterations, where the execution cost is expected to be equal to or greater than one-fifth of the iteration count.

One strength of this approach is how it handles different hyperparameters within a Grid Search through a linear relationship, so every possibility gets covered without having to split the space.

Another feature is cost control through the number of words used in labeling functions. We found that as the word count grows, the total cost climbs quadratically, which tells us something useful about where the cost comes from.

There is also an accuracy threshold that caps how many iterations run. This cutoff does not tie directly to the iteration count, which gives the algorithm more room to breathe during the search.

Algorithm 1 handles the tuning of an arbitrary value, which we call optimizationValue. Its aim is to find the best setting for a given hyperparameter when building a model on the training data with a machine learning algorithm. In Step 3, the value range is defined as a vector bounded by an upper value x and lower value y, and this vector is assigned to optimizationValue.

In Step 4, the first entry of optimizationValue is assigned to j, which stands for a hyperparameter. Steps 5 through 7 take care of setting initial values for flag, accuracy, and accuracyOld. The flag acts as a control switch: it decides whether j should be increased on the next iteration. Accuracy is evaluated at each iteration of the algorithm to modify the hyperparameter value accordingly, while accuracyOld stores the previous accuracy obtained in the previous iteration.

Steps 9-17 replace the better hyperparameter and accuracy with their respective old values. Steps 19 through 26 bring the loop to a close, leaving us with the hyperparameter that gave the highest accuracy in the algorithm. The variable v controls how much the hyperparameter value changes from one iteration to the next. For instance, if the hyperparameter value fluctuates between 0.1 and 0.9, v is set to 0.1 to observe the impact on accuracy. The codes devised to experiment are publicly available for replication purposes.

Algorithm 1. Heuristic tuning algorithm

1: **Input:** dataset, threshold, iteration, v
2: **Output:** maxAccuracy, optimalHyperparameter
3: optimizationValue ← c(x : y)
4: j ← optimizationValue[1]
5: flag ← 0
6: accuracyOld ← 0.6
7: thresholdAccuracy ← 0.8, optimalHyperparameter ← −1
8: count ← 0
9: **while** (accuracy < thresholdAccuracy||count <= iteration) **do**
10: accuracy ← ϕ(j)
11: **if** accuracy >= accuracyOld **then**
12: maxAccuracy ← accuracy
13: optimalHyperparameter ← j
14: j ← j + v
15: **else**
16: j ← j − v
17: **end if**
18: accuracyOld ← accuracy
19: **if** accuracy > thresholdAccuracy **then**
20: flag ← 1
21: optimalHyperparameter ← j
22: break
23: **end if**
24: count ← count + 1
25: **end while**
26: **return** (optimalHyperparameter, maxAccuracy) ▷ result includes optimal configurations found by the algorithm

Machine Learning Algorithms

Support Vector Machines (SVM) are robust algorithms widely used in classification tasks [57]. SVM gives us room to tune several hyperparameters, including kernel, cost, degree, and gamma, which lets it take on both linear and non-linear problems [58]. With kernel functions such as linear, polynomial, sigmoid, and Radial Basis Function (RBF), SVM pushes the data into higher-dimensional spaces, where complex decision boundaries become tractable [59].

We used SVM to predict developer expertise and spot programming languages. Optimization techniques were applied to tune its hyperparameters, pushing up accuracy without letting the computational load grow out of hand.

Random Forest is another ensemble learning method, and it works for both classification and regression tasks. It merges many decision trees, avoiding overfitting of this combined model and improving generalization performance [60]. It has two major hyperparameters: mtry, defining the number of features available for each split, and ntree, fixing the number of trees in a forest [61]. The performance of Random Forest in this study showed a better trend for large and high-dimensional datasets. However, it is the balance between robust accuracy and computational cost. An optimized mtry and ntree often yield the best results.

Gradient Boosting Machine (GBM) encompasses an ensemble method that iterates and improves weak learners by using their errors to correct them [62]. Some of the key

hyperparameters of the said model are shrinkage (learning rate), n.trees (number of trees), and n.minobsinnode (minimum number of observations in each node). Proper tuning of these parameters can significantly improve the model's performance. While this approach maximized accuracy in predicting developer expertise, it required a careful balance of computational resources.

Ensemble of Probabilistic Subclassifiers (EPS) is a probabilistic ensemble method designed for multi-label classification; it is particularly effective on sparse datasets [63]. Other important hyperparameters are subsample (the ratio of data used in training), strategy (the approach for handling sparse label input), b (the number of subsets used in the strategy), and p (pruning rate to reduce error) [64]. Performance on sparse data conditions made it best suited for recommendation systems relying on a few labeled examples.

Ensemble of Binary Relevance (EBR) is a multi-label classification method in which every label is independently treated by training one model for each [65]. Key hyperparameters in this regard include subsample, which defines the data sample ratio used during training, and attr.space, which indicates the proportion of features used in training [66]. Tuning these parameters gave us balanced generalization and kept overfitting in check.

Table 2 lays out the machine learning algorithms we used along with their hyperparameter settings. Tuning these hyperparameters mattered a lot for how well the algorithms performed, especially on the harder multi-label classification tasks.

METHOD

Dataset

Within this study, a new dataset has been created to predict a programmer's experience using tag and question datasets obtained from the SO website [67]. The dataset has been enriched with features extracted from the content of the questions, alongside tag information corresponding to each question.

The resulting dataset follows a structural format where each record has a unique identifier. In addition to features, the dataset contains tags representing different programming languages (JavaScript, SQL, Java, C#, Python, C++, C, PHP, Ruby, Swift, Objective- C, VB.NET, Perl, Bash, CSS, Scala, HTML, Lua, Haskell, Markdown, and R) and experience levels. This dataset has been created based on

Table 3. The features of the dataset

Dataset	Feature	Description
Question	Id	Unique number
	OwnerUserId	User id
	CreationDate	The date input is entered
	Score	The reputation of the input
	Title	Title of the input
	Body	Context of the input
	ClosedDate	The date input is closed
Label	Id	Id of the input
	Tag	Tags are given by the user

Table 2. Summary of machine learning algorithms and hyperparameter settings

Algorithm	Parameter	Description	Strengths	Range	Step
SVM	kernel	Kernel type for SVM	Effective for non-linear problems	-	-
	degree	Polynomial kernel exponent		1-10	1
	gamma	Kernel coefficient		2^{-10}-2^{10}	0.1
	cost	Penalty of the error term		2^{-10}-2^{10}	1
RF	mtry	Features for each split	Generalization, scalability	1-8	1
	ntree	Trees in the forest		100-500	1
GBM	shrinkage	Impact of additional base-learner	High accuracy, iterative learning	0.001-1	0.1
	n.trees	Trees to fit		100-500	1
	n.minobsinnode	Min. observations in terminal nodes	Sparse data, multi-label tasks	5-15	1
EPS	subsample	Training instance rate		0.1-1	0.1
		Strategy for sparse clusters			
	strategy	A Number used in strategy		A-B	-
	b	Instances pruned		1-4	1
	p			1-4	1
EBR	subsample	Training instance rate		0.1-1	0.1
	attr.space	Attributes rate	Label independence, scalability	0.1-1	1

tags expressing relationships between the skill levels of programmers and programming languages, representing their experiences. For detailed information about the corpus, refer to Table 3.

Text Mining Operations

To achieve optimal results, the dataset size was incrementally increased during the experiments. Each experiment was run at least 20 times to ensure reliable results, allowing the observation of the impact of different dataset sizes on the inputs. It was determined that 20,000 records appeared to be a good fit. The maximizing of model parameters through text processing techniques set the stage for correctly tuning the data that was used to maximize the suggestion system, overall, endowing it to provide more successful outcomes. We also created subsets of records containing 500, 1000, and 3000 records to test certain scalability and performance dimensions on several sizes of datasets. These subsamples enabled a thorough analysis of the effect of dataset size on accuracy, memory consumption, and computational complexity metrics.

Data Cleaning and Preprocessing

Text mining techniques were used to transform the titles and contents of questions in the SO dataset into vectors. In the dataset, every word was taken as the feature, and occurrences of the same word in a document were quantified. Data were cleaned by removing punctuation, numbers, and spaces in this step. Subsequently, a Term Document Matrix (TDM) was generated using the processed data [68]. This resulted in the addition of two new features, word count and character count, to the dataset, allowing for more granular insight into the text.

After applying data mining techniques to the datasheet, the words extracted from the dataset were graphically represented using a visualization technique referred to as a "word cloud." Word clouds are a visualization tool to highlight unique words from textual data; words are represented in varying sizes according to their frequency [69]. This visualization has word size corresponding to occurrence frequency in the text. More frequent or more robust words in the dataset appear larger. Thus, the transmission of the essence of data can be understood more rapidly and effectively. The visual analysis illustrated in Figure 1 played a role in dissecting important pieces of the data, generating keywords for functional programming, and rendering outcomes in a way that's easier to digest.

Experience Tagging and User Level

To create the labels, 21 prominent programming languages from the past 12 years were selected and labeled from y1 to y21. These languages include JavaScript, SQL, Java, C#, Python, C++, C, PHP, Ruby, Swift, Objective-C, VB.NET, Perl, Bash, CSS, Scala, HTML, Lua, Haskell, Markdown, and R [70]. Stack Overflow's feature of publicly sharing datasets every two years enabled meaningful insights to be extracted from the entries. Labeling for these languages was performed using rule-based functions developed in the R programming language. Each function assigns a value of "1" when keywords related to the respective language are found in the question titles or contents.

The y22 label was used to determine the experience levels of SO users. User experience was classified into three

Figure 1. Word cloud of the processed dataset.

levels: beginner (1), intermediate (2), and advanced (3). Figure 2 illustrates the categorization of coding experience into these three categories. If the questions lacked words or phrases indicating programming experience, the y22 label was assigned a value of "0." For the classification of user experience, the identified words or patterns in the questions were taken into account.

Table 4 presents an example of the labeled dataset after preprocessing and tagging. The dataset includes 21 binary labels for programming languages (y1 to y21) and a multiclass label (y22) for user experience.

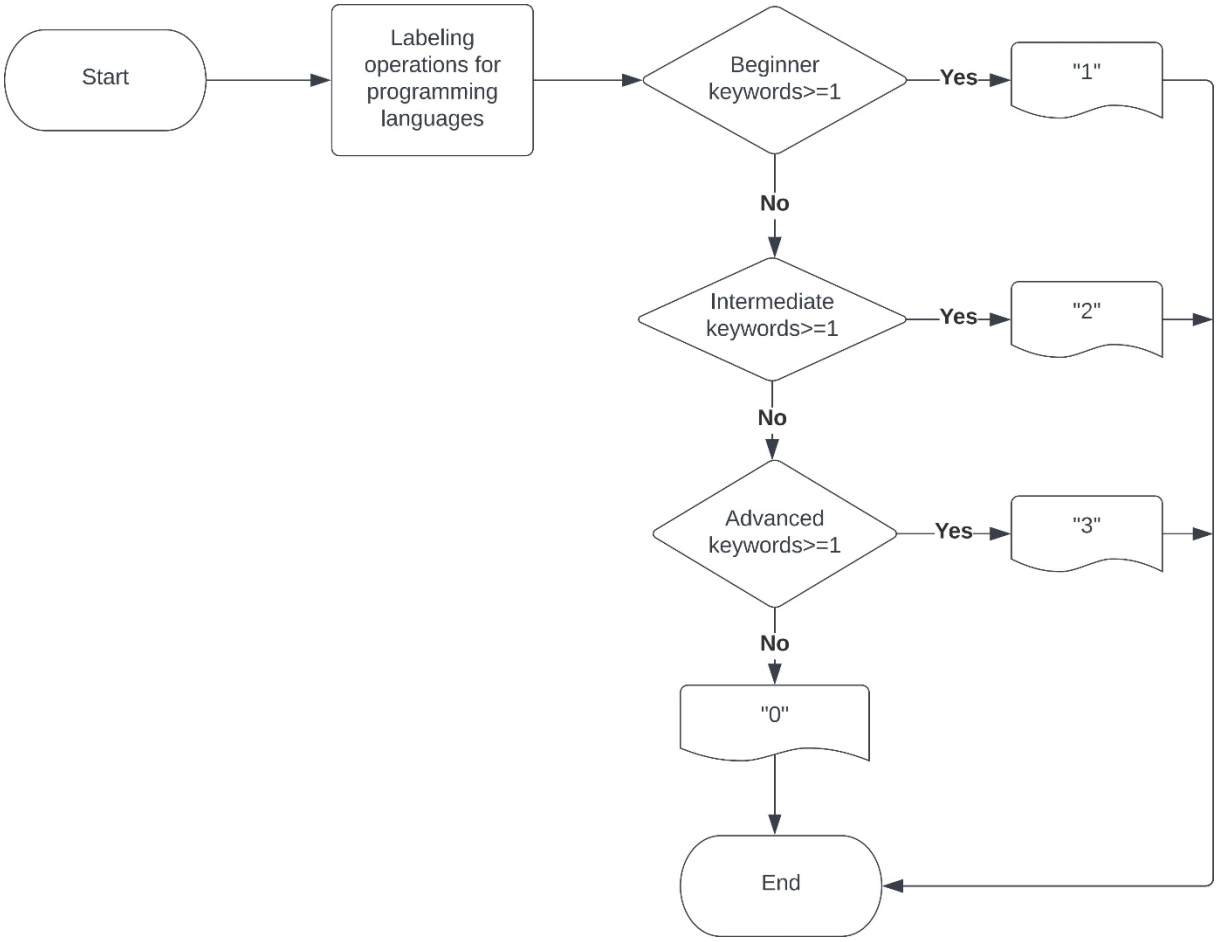

Figure 2. Steps of the labeling process.

Table 4. Processed dataset

y1	y2	y3	y4	y5	y6	y7	y8	y9	y10	y11	y12	y13	y14	y15	y16	y17	y18	y19	y20	y21	y22
0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
0	1	0	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	2
0	0	0	0	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
0	0	0	1	0	0	0	0	0	0	0	0	0	0	0	1	0	0	1	1	0	1
0	0	0	0	1	1	0	0	0	0	0	0	0	0	0	0	0	0	1	1	0	1
0	0	0	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	3
0	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1
0	0	0	0	1	0	0	0	0	0	0	0	0	0	0	1	0	0	0	0	0	0
0	0	0	0	1	0	0	0	0	0	0	0	0	0	0	1	0	0	0	0	0	1

Software Framework (HyperOpt)

HyperOpt was built to make hyperparameter optimization straightforward and to let R code run as part of the same workflow. Figure 3 shows the architecture, which pulls together a few core components into one flexible and efficient platform. Things start at the data input stage, where users upload their datasets and the system pushes them through to the optimization pipeline. From there, the optimization module tests hyperparameters using Grid Search, Random Search, Bayesian Optimization, Nelder-Mead, and our proposed heuristic method, all with the goal of getting the model to perform as well as it can. Each method iteratively refines hyperparameters based on user-defined thresholds and evaluation metrics like accuracy, resource usage, and computation time.

The HyperOpt web platform is divided into four main sections, two key components being: (a) hyperparameter optimization tools and (b) result visualization. These tools together allow the user to set up optimization processes, monitor the progress, and effectively analyze the results.

Some of the techniques that can be used (some are shown in Figure 4(a)) are Grid Search, Random Search, Bayesian Optimization, Nelder-Mead, and a heuristic approach proposed in this paper. All of these methods systematically go through all the possible ranges of hyperparameters to search for the best model performance. The threshold value and iteration count are examples of parameters that can be defined by the user to drive the optimization. In each iteration, the platform logs the accuracy along with its associated hyperparameter combination. Also, to facilitate decision-making, relevant indicators like elapsed time in seconds and memory consumption in MB are computed and displayed.

Figure 4(a) shows a plot of the range across different values of one specific hyperparameter, illustrated in this example by using the mtry hyperparameter. Thus, on the horizontal axis, it shows the range for a certain mtry, while on the vertical axis, the scores of accuracy are reflected, and recorded for the optimizations through this range. Though mtry is used as the example here, any other hyperparameter is open to be analyzed with the help of visualization for assessing a particular value that may be most influential toward improving the model.

Figure 4(b) provides a detailed representation of the relationships between hyperparameter settings and model performance. This graph illustrates both the mean accuracy and its standard deviation for various hyperparameter configurations. The horizontal axis carries the values of the chosen hyperparameter, and the vertical axis shows the accuracy metrics that go with them. A smooth curve

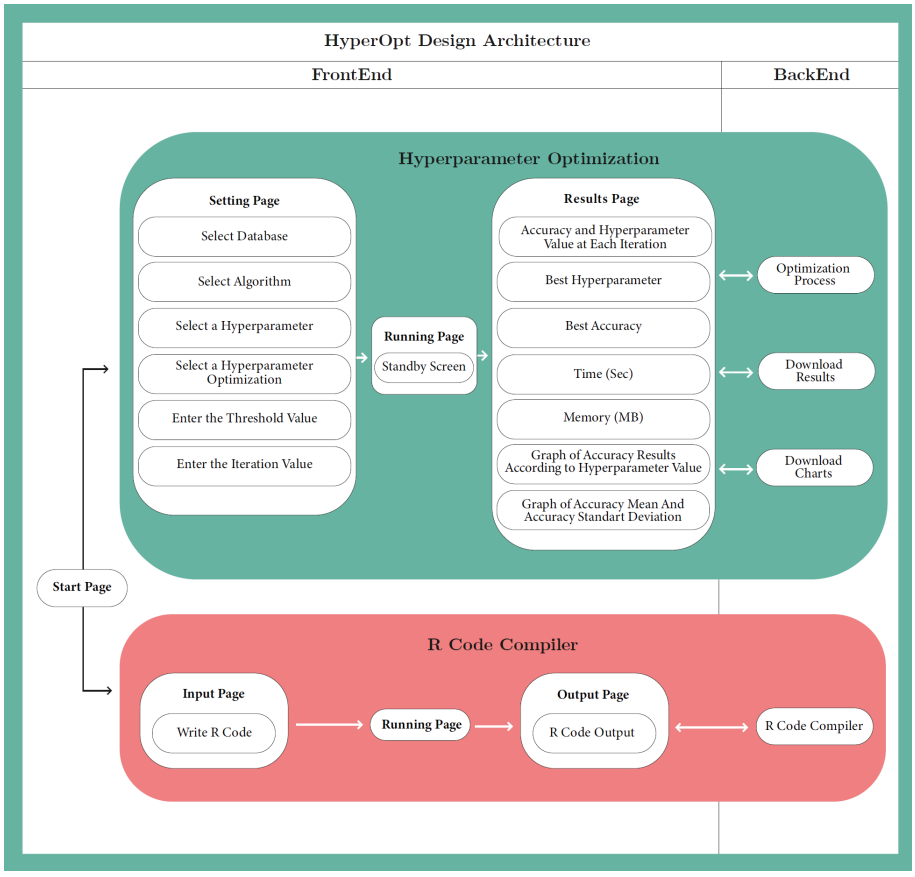

Figure 3. Architecture of HyperOpt.

traces the mean accuracy, while a shaded region or error bars mark the standard deviation. Taken together, these give users a quick read on performance trends, how much results vary, and how consistent the model is across different configurations.

The mean accuracy is simply the average accuracy across several runs for a given hyperparameter value. Formula 8 below shows how we compute it:

$$\text{Accuracy Mean} = \frac{1}{N}\sum_{i=1}^{N} \text{Accuracy}_i \tag{8}$$

In this formula, N is the total number of iterations, and Accuracy_i is the accuracy obtained at the i-th iteration for the hyperparameter value in question. This gives us a single number to anchor on when we want to judge how the model performs under a particular hyperparameter setting.

The standard deviation reflects the variability or spread of accuracy scores. Its formula is given as 9:

$$\text{Accuracy Standard Deviation} = \sqrt{\frac{1}{N}\sum_{i=1}^{N}(\text{Accuracy}_i - \text{Accuracy Mean})^2} \tag{9}$$

In this calculation, (Accuracyi−Accuracy Mean) refers to the deviation of an individual accuracy score from the mean, squared to emphasize outliers. When the standard deviation is small, the model performs consistently across iterations; when it is larger, the results vary more.

HyperOpt also makes life easier on the saving side: users can export the resulting graphs as JPG, PNG, or PDF files, which keeps storing and sharing simple. This feature supports collaboration and thorough documentation, making the platform invaluable for researchers and practitioners focused on model optimization.

Additionally, an integrated R code compiler (illustrated in Figure 5) enables users to execute R scripts directly within the HyperOpt environment. The user can enhance the workflow efficiency due to advanced dynamic adjustment of parameters: in real-time, they see all the results and can immediately validate changes to the model. As discussed, HyperOpt supports iterative model refinement without requiring any additional application; hence, it is very powerful and practical.

HyperOpt embeds a tailor-made R code compiler, which is an interesting approach to problems in a certain field. Modifications of scripts can be easily done to fit either with the specific dataset or the model, which makes this platform useful for more complex analyses. This thus makes HyperOpt a general-purpose tool that can be used for many research directions and more sophisticated modeling activities.

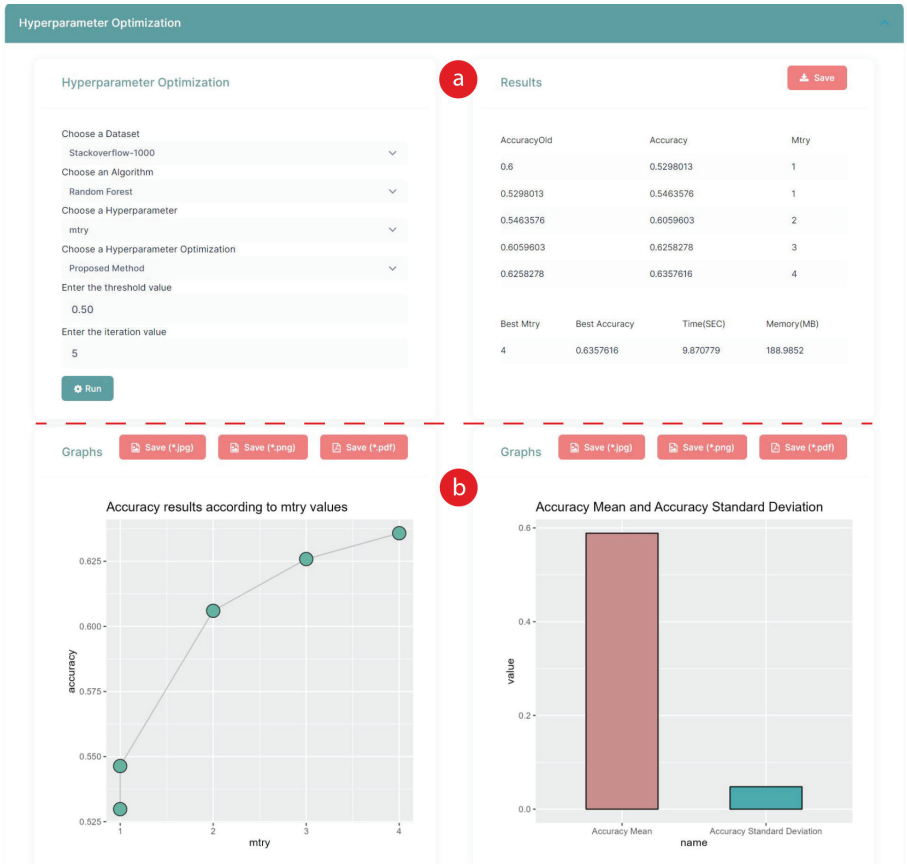

Figure 4. HyperOpt web interface.

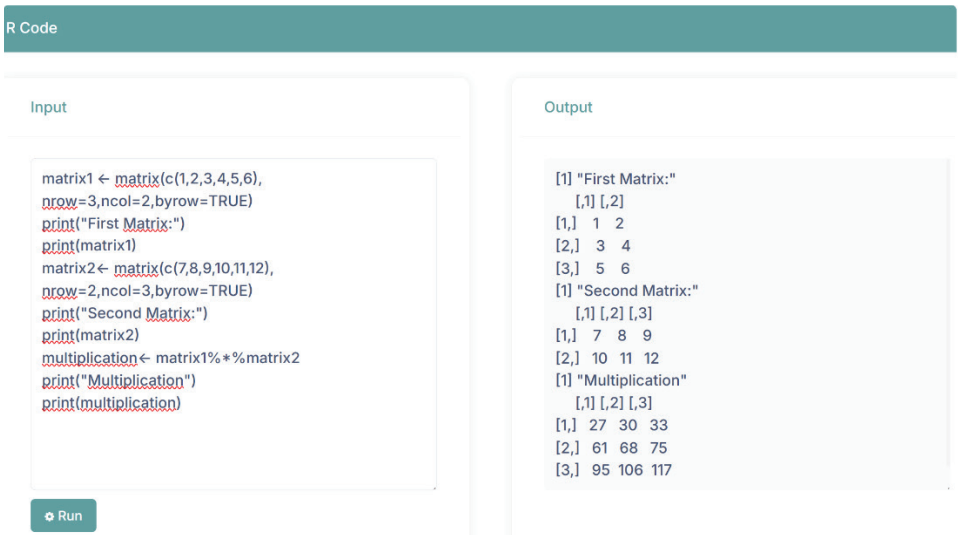

Figure 5. R code compiler.

With a persistent focus on design, HyperOpt aims to make state-of-the-art optimization tools simple to use. It is designed to be an easy-to-use tool regardless of experience but also serves as a powerful tool for users who are experienced.

The usable and functional interface of HyperOpt is designed with the end user in mind. This allows them the opportunity to test a variety of optimization techniques, track trends in accuracy and how they are affected by different hypers and allows them to gain an insight into the model better. This delineation empowers users to fine-tune the models and other elements with ease, therefore achieving the best possible outcome with minimal effort.

RESULTS AND DISCUSSION

This research presents a detailed performance comparison of machine learning models with different dataset sizes in terms of optimization techniques and algorithms used. Each method was performed in detail using different configurations of hyperparameters and each of the results was met with a broad comparison.

In Figure 6, the authors provide the results in a bar graph comparing the accuracy of five algorithms which are Random Forest, GBM, SVM, EBR and EPS across six optimization techniques including: Before Optimization, Grid Search, Random Search, Bayesian Optimization, Nelder Mead and Proposed Method. The results were based on sample datasets of 500, 1000, and 3000 while averages were obtained from 10 runs per sample and sample method.

The status Before Optimization are as a comparison where one can truly see the effects that hyperparameter tuning would have. It was observed that for every dataset size used, the baseline accuracy figures were on the lower side. For instance, for the dataset with 3000 records, the baseline accuracy of both Random Forest and GBM stood at 0.62, this indicates that there is so much room for further enhancement. SVM, EBR, and EPS had baseline accuracies of 0.66, 0.51, and 0.46, respectively and similarly better results were expected.

In general, the Proposed Method outperformed others in post-optimization accuracy across datasets and algorithms, hence proving efficiency in navigating hyperparameter spaces. For Random Forest, the accuracy increased to 0.66, 0.65, and 0.66 for datasets with 500, 1000, and 3000 entries, respectively, which again proved the scalability and robustness of the method. Similarly, GBM reached the highest values of accuracy with the Proposed Method, outperforming all other alternative optimization approaches across dataset sizes.

While the accuracy for SVM was steadily at 0.66 for all dataset sizes, surprisingly, some methods like Bayesian Optimization and Nelder-Mead have shown a drop in accuracy with larger datasets. On the other hand, the Proposed Method acted to negate such effects, thus giving back the baseline accuracy of these methods and further supporting their reliability.

The Proposed Method had considerable improvements, such as EBR and EPS, for models with low baseline accuracies. For instance, EBR had an increase in accuracy to 0.58, 0.57, and 0.58 for the three dataset sizes, while EPS had a constant accuracy of 0.56 across the datasets. This reflects how well the method generalizes to models with complex hyperparameter landscapes.

On the other hand, alternative methods for Bayesian Optimization and Nelder-Mead have mostly failed on datasets of larger sizes or with more complex models. Except for these, Grid Search and Random Search could give a decent improvement in accuracy but are much more

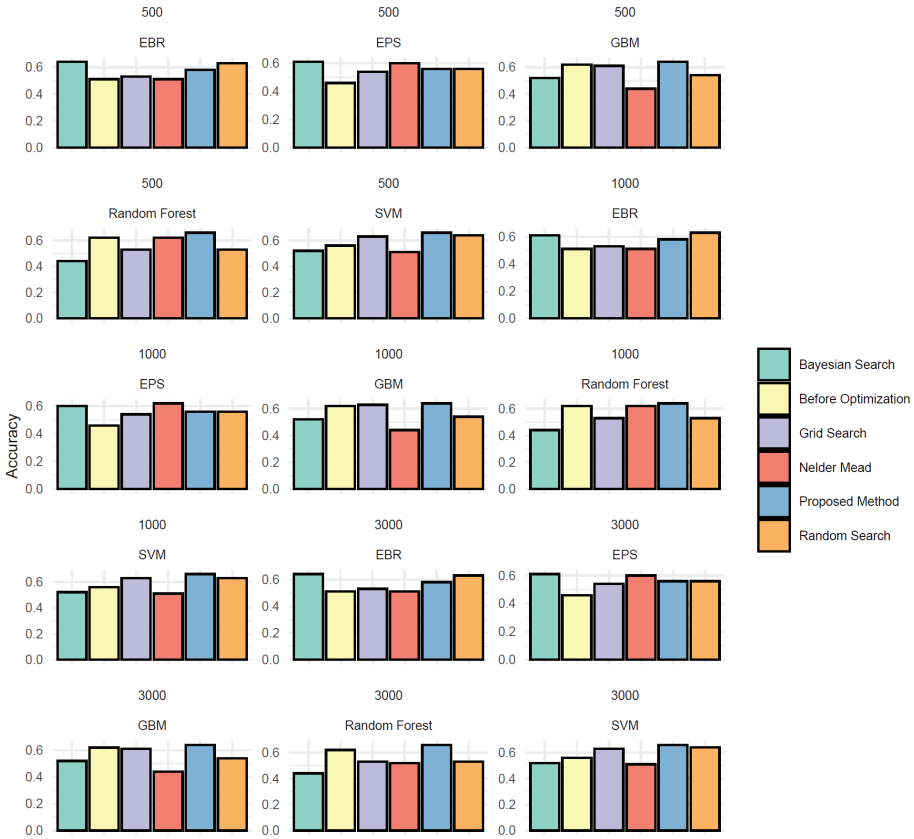

Figure 6. Accuracy comparison of machine learning algorithms under different optimization techniques.

computationally expensive reflected by longer execution time and higher memory usage.

Figure 6 summarizes, in general, that the Proposed Method does well in every algorithm and dataset scale concerning accuracy, while it provides a clear indication of different optimization techniques that perform badly across different conditions.

Figure 7, the performance comparison of five algorithms, namely, Random Forest, GBM, SVM, EBR, and EPS, for different optimization methods, namely, Before Optimization, Grid Search, Random Search, Bayesian Optimization, Nelder-Mead, and the Proposed Method. These results are averaged over several runs on datasets with 500, 1000, and 3000 entries.

This analysis puts into perspective the computational cost of each of these optimization techniques. It is good to refer to baseline times, such as from the Before Optimization scenario, for additional overhead brought in by the hyperparameter tuning process. Taking the example of the 3000-entry dataset, we observe that the baseline time taken for both Random Forest and GBM were well below times for Grid Search and Random Search. The numbers do not consider the performance improvement achieved by these optimizations.

Among all the algorithms and datasets, Grid Search and Random Search were the most time-consuming. That is in line with their exhaustive and stochastic approaches since they have to try a load of combinations of hyperparameters. For instance, for SVM on the 3000-entry dataset, Grid Search had to spend a significantly longer amount of time compared to the Proposed Method. It reflected the trade-off between time and performance: one invests time and enjoys superior improvements.

In most applications, the Proposed Method presented almost consistent low time consumption in contrast to competitive optimization output. It has shown competitive matches or outperformed those methods, like Random Forest and GBM, with significantly lower required time, underlining efficiency in performing hyperparameter exploration.

Bayesian Optimization and Nelder-Mead showed mixed performance. Bayesian Optimization, on the one hand, is faster compared to Grid Search and Random Search but often returns less significant improvements in accuracy for EPS and EBR. The results of Nelder-Mead were moderately consumptive with time but inferior to the Proposed Method in terms of performance and scalability, especially on large datasets.

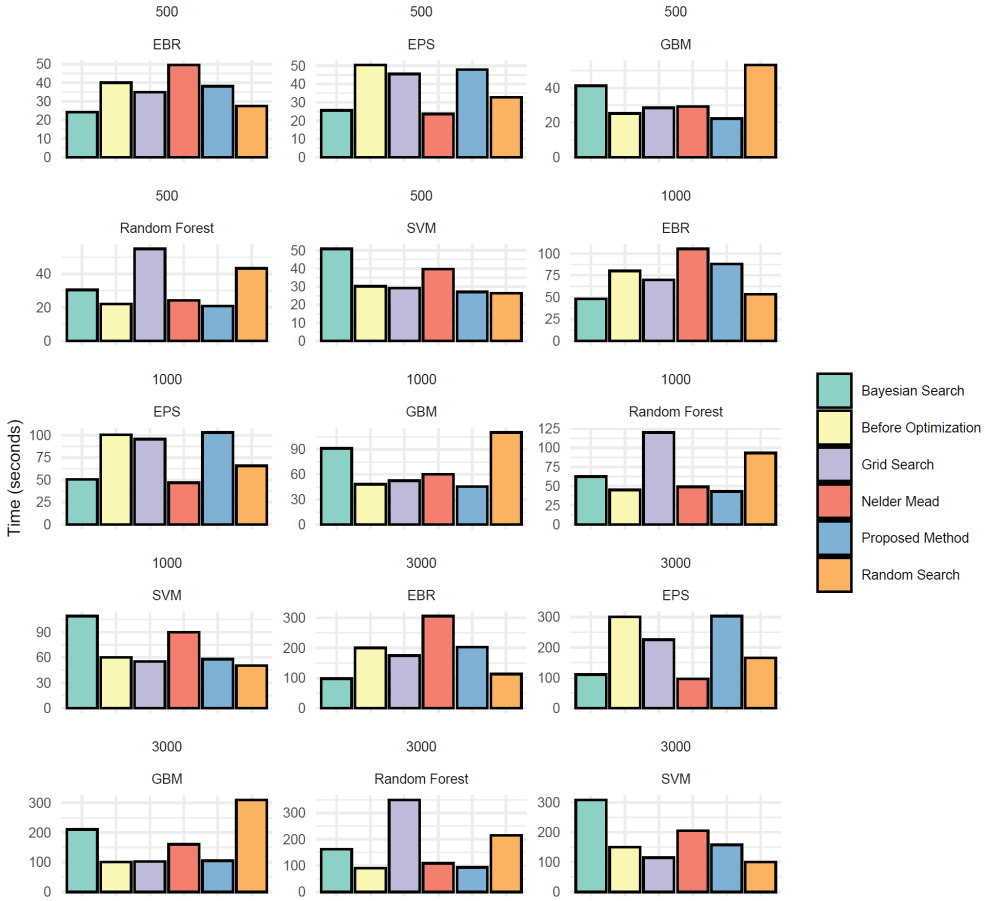

Figure 7. Time comparison of machine learning algorithms under different optimization techniques.

Among these, time consumption analysis underlined again that the Proposed Method would be effective in balancing between optimization effectiveness and computational efficiency, establishing a stronger option of hyperparameter tuning across diverse settings.

Figure 8 presents the memory consumption of the five algorithms, namely Random Forest, GBM, SVM, EBR, and EPS using different optimization methods: Before Optimization, Grid Search, Random Search, Bayesian Optimization, Nelder-Mead, and the Proposed Method. The memory consumptions were averaged over multiple runs while the datasets contained 500, 1000, and 3000 entries.

This analysis gives an indication of the differing memory requirements for hyperparameter tuning techniques. The Before Optimization scenario acts as our baseline. It shows what memory usage looks like without any iterative tuning. Not surprisingly, baseline memory stayed lower than any of the optimization methods across every dataset size we tried, since no extra computation is happening at this stage.

Grid Search and Random Search turned out to be the heaviest on memory in every scenario. Both have to go through a large number of hyperparameter combinations (Grid Search exhaustively, Random Search stochastically), and that pushes up the resource demand. On the 3000-entry dataset, for example, Grid Search on GBM and EPS used noticeably more memory than our Proposed Method did.

The Proposed Method came out ahead on memory efficiency. For Random Forest, GBM, and SVM, it produced results that were competitive with or better than the alternatives, but with a much lighter memory footprint. That makes it a good fit for setups where computing resources are tight, or where the work needs to scale up to bigger datasets.

Bayesian Optimization landed somewhere in the middle on memory use, doing a better job of balancing efficiency and computational demand than Grid Search or Random Search. The catch is that it did not always deliver large enough performance gains to justify itself. Nelder-Mead used the least memory of all the methods, but its results were often weak, especially on algorithms with complicated hyperparameter spaces like EBR and EPS.

To wrap up, the memory analysis points to the Proposed Method as the one that best balances resource use and

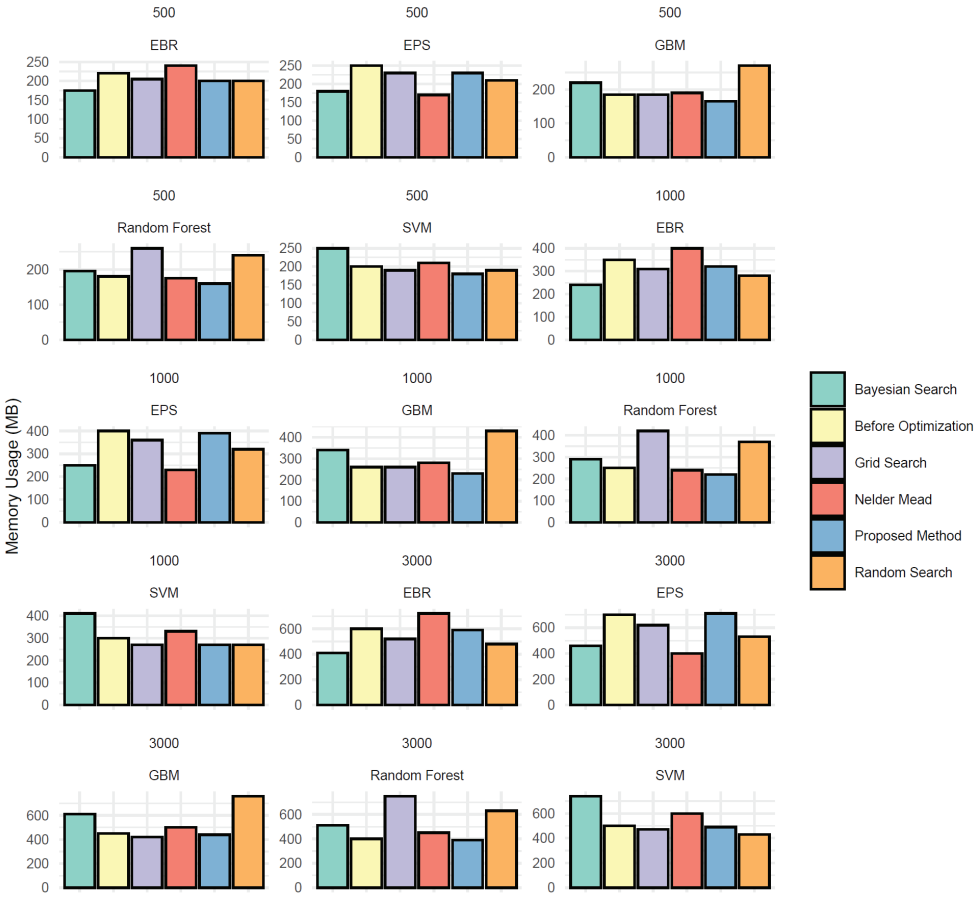

Figure 8. Memory comparison of machine learning algorithms under different optimization techniques.

optimization quality. That makes it a solid option across a range of applications, whether you are working in a tight computing environment or handling large-scale datasets.

The findings in Table 5 highlight the performance of various hyperparameters across optimization methods, dataset sizes, accuracy, processing time, and memory usage. Each hyperparameter's best-performing method is identified, and the effect of dataset size on overall performance is analyzed comprehensively.

As illustrated in Figure 9, the hyperparameter optimization results are further visualized across the three metrics (accuracy, time, and memory) for five optimization methods: Proposed Method, Grid Search, Random Search, Bayesian Optimization, and Nelder-Mead. The figure offers a detailed breakdown of the performance of these methods for each hyperparameter and dataset size (500, 1000, and 3000 entries).

mtry: The Proposed Method achieved the best performance for the mtry parameter across all dataset sizes. While the accuracy for the small dataset (500 samples) was 0.6174, it increased to 0.6644 for the large dataset (3000 samples). Processing time grew with dataset size, from 5.83 seconds for the small dataset to 60.27 seconds for the large dataset. Similarly, memory usage increased with larger datasets.

ntree: The ntree hyperparameter performed best with the Grid Search method, achieving the highest accuracy of 0.6857 on the medium-sized dataset (1000 samples). However, this performance came with significant costs in processing time and memory usage. For the large dataset, processing time rose to 540 seconds, with memory usage reaching 226 MB.

degree: The Proposed Method provided high accuracy for the degree parameter on small and medium-sized datasets but showed a decline in accuracy (0.5851) for the large dataset (3000 samples). Processing time significantly increased for larger datasets, reaching 20.43 seconds.

gamma: The Proposed Method outperformed other methods for small datasets, while Random Search achieved the best accuracy on medium-sized datasets. For large datasets, the Proposed Method maintained shorter processing times than other methods, though its accuracy was limited to 0.5608.

cost: For the cost parameter, the Proposed Method demonstrated shorter processing times and lower memory

Table 5. Hyperparameter optimization results across different dataset sizes and methods

Hyperparameter	Best Method	Dataset Size	Best Accuracy	Time (s)	Memory (MB)
mtry	Proposed Method	500	0.6174497	5.83135	181.3848
		1000	0.65	16.0182	189.792
		3000	**0.664451**	**60.2661**	**227.0672**
ntree	Grid Search	500	0.671428	60.1103	189.615
		1000	0.685714	120.629	198.834
		3000	0.682577	540.857	226.311
degree	Proposed Method	500	0.614864	1.43987	158.584
		1000	**0.634551**	**3.36655**	**159.090**
		3000	0.585177	20.4345	160.967
gamma	Proposed Method	**500**	**0.590604**	**0.7777**	**161.242**
	Random Search	1000	0.63815	16.1707	176.018
	Proposed Method	**3000**	**0.560840**	**22.3856**	**160.930**
cost	Proposed Method	500	0.53333	60.2175	158.583
		1000	0.51986	300.907	158.890
		3000	**0.5608**	**23.8777**	**161.124**
ntree	Proposed Method	**500**	**0.664429**	**26.146**	**161.801**
		1000	0.6369	26.294	162.451
		3000	0.64269	32.2069	165.083
minobsinnode	Proposed Method	500	0.65100	25.295	161.797
		1000	**0.67109**	**27.135**	**162.448**
		3000	0.651548	32.2606	165.078
shrinkage	Bayesian Optimization	500	0.68	25.6277	160.619
		1000	0.6345	35.2656	162.121
	Proposed Method	**3000**	**0.63676**	**33.465**	**165.079**
eps-subsample	Bayesian Optimization	500	0.2234	6.7714	180.346
	Proposed Method	**1000**	**0.5619**	**51.800**	**227.340**
	Grid Search	3000	0.503	900.151	291.569
eps-b	Proposed Method	**500**	**0.482**	**240.811**	**360.983**
	Random Search	1000	0.3660	300.320	266.552
	Bayesian Optimization	3000	0.5009	420.680	182.422
eps-p	Random Search	500	0.2391	60.457	211.74
		1000	0.3604	180.436	240.547
	Bayesian Optimization	3000	0.4885	420.122	182.422
ebr-subsample	Grid Search	500	0.3678	360.503	395.283
	Random Search	1000	0.3769	1380.72	467.708
	Grid Search	3000	0.5409	3600.05	733.654
ebr-attrspace	Random Search	500	0.3395	660.852	387.531
		1000	0.4126	1860.81	468.678
		3000	0.540	3600.92	181.742

usage. That said, on accuracy, it trailed behind Grid Search and Bayesian Optimization.

minobsinnode and ntree: For these two hyperparameters, the Proposed Method held up well across dataset sizes of every scale, staying efficient on both runtime and memory use relative to the other methods.

shrinkage: On small datasets, Bayesian Optimization came out on top with an accuracy of 0.68. As the dataset grew, though, the Proposed Method generalized better, which speaks to how well it adapts under different conditions.

eps-subsample and eps-p: While Grid Search and Bayesian Optimization had better accuracy, these methods

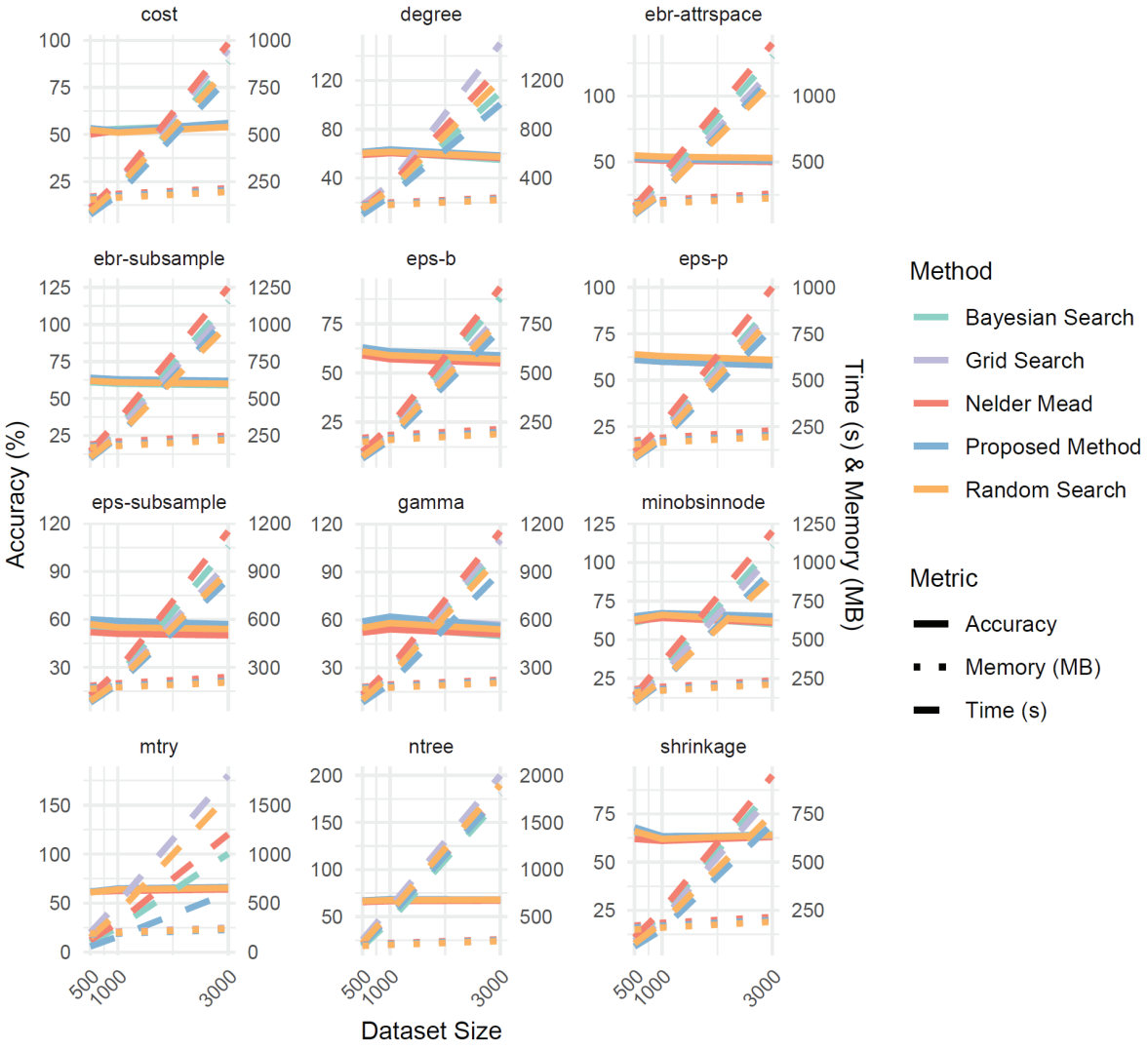

Figure 9. Hyperparameter optimization results for different methods and dataset sizes across various metrics.

consumed quite a lot of processing time and memory, especially for large datasets. On the contrary, the Proposed Method yielded balanced results, performing exceptionally well on medium-sized datasets, hence quite feasible for application.

ebr-subsample and ebr-attrspace: The Grid Search produced the best accuracy with 0.5409 for big datasets but at the price of very high processing time, 3600 seconds, and memory, 733 MB, hence it becomes computationally infeasible in resource-constrained settings.

In a nutshell, the Proposed Method has established itself as a robust and efficient optimization method for a variety of hyperparameters and dataset sizes. While traditional methods like Grid Search and Random Search are highly accurate, their computational cost is prohibitively high for larger datasets. Bayesian Optimization and Nelder-Mead gave a moderate performance but failed to be consistent. The Proposed Method strikes a good balance between accuracy, processing time, and memory efficiency; hence, it is the most practical choice for hyperparameter optimization in diverse contexts.

CONCLUSION

This paper proposes a holistic framework for predicting developers' experiences with programming languages using the SO dataset. The researchers have prepared three datasets of different sizes: 500, 1000, and 3000 entries. They applied different machine learning algorithms along with different optimization techniques. Results show the interrelation between dataset size, optimization methods, and important performance metrics such as accuracy, memory usage, and computational efficiency.

The Proposed Method came out to be the best among them, as it always outperformed traditional optimization techniques in the prediction of developer experiences from

the SO dataset. It achieves better performance and efficiency due to enhanced hyperparameter tuning and higher-order machine-learning strategies. This approach marks a leap in the development of hyperparameter optimization by incorporating easy-to-use tools with advanced optimization techniques.

Additionally, in many cases where the lighter datasets were indeed produced, larger datasets are usually associated with less precise results and impose a cost on higher computation. Some of the above considerations denote a delicate balance that existed between dataset size and algorithmic choice toward optimal performance. The Proposed Method presents excellent adaptability among different optimization techniques that include Grid Search, Random Search, and Bayesian Optimization. Coupled with Bayesian Optimization, the approach grants an excellent balance between exploration and exploitation for improved performance with low computational overhead.

One of the key contributions of this study is the development of HyperOpt, a web application designed to automate hyperparameter optimization and support model building. Beyond the new optimization method we propose here, HyperOpt gives users an R-based platform they can actually work with, letting them predict developer experience from the programming languages they prefer. It takes much of the pain out of hyperparameter tuning on large datasets, which is where the real bottleneck tends to sit, and that usefulness carries across both research and industry settings.

In the future, many avenues are open to explore for improving both the methodology and its application. Much more can be done to extend the range of supported machine learning algorithms, introducing more hyperparameters for an even more flexible and fine-grained insight into model performances. The testing of methodology on various datasets and research into the adaptation of this approach within a real-time system would further assert robustness.

Apart from these, a comparative study incorporating techniques like Hyperband, TPE, and genetic algorithms may also be presented to establish the efficacy of the Proposed Method against those in the state of the art. Also, besides providing accuracy, performance metrics like precision, recall, F1 score, and ROC curves should not go unmentioned. At the backend, a much more scalable and responsive HyperOpt is achieved through some additional steps of optimization regarding parallel computation, the use of distributed systems, and feedback mechanisms in real time.

The interface of the web application, including result visualization and integration of user feedback, requires further improvements to enhance its usability. Such enhancements are expected to make the application significantly more accessible to users ranging from beginners to expert practitioners. Working on these aspects pushes the Proposed Method further toward being a method that scales well, holds up under real use, and stays approachable for practitioners looking at hyperparameter optimization.

This work sets a base to build on, both for optimization techniques themselves and for the web-based systems that put them to use. Sharpening the optimization methods, bringing in more algorithms, and folding in a wider set of evaluation metrics are the natural next steps if the goal is to stay close to the state of the art. What we have learned along the way feeds into building machine learning tools that run leaner and reach further.

AUTHORSHIP CONTRIBUTIONS

Authors equally contributed to this work.

DATA AVAILABILITY STATEMENT

The dataset used in the manuscript is taken from one source: https://www.kaggle.com/datasets/stackoverflow/stacksample.

Code availability: The codes for the experiment are available at

https://github.com/fatmaaltinsoy/a-heuristic-technique-for-hyperpameter-tuning.git.

Web interface: The web interface is available at https://hypermlopt.sdu.edu.tr.

CONFLICT OF INTEREST

The author declared no potential conflicts of interest with respect to the research, authorship, and/or publication of this article.

ETHICS

There are no ethical issues with the publication of this manuscript.

STATEMENT ON THE USE OF ARTIFICIAL INTELLIGENCE

Artificial intelligence was not used in the preparation of the article.

REFERENCES

[1] Mejía J. The software engineering evolution: A look through 11 years. In: 11th International Conference on Software Process Improvement (CIMPS); 2022. p. 245–248. [CrossRef]

[2] Glass R. A study about software maintenance. Inf Syst Manag 2012;29:338–339. [CrossRef]

[3] Karl F, Pielok T, Moosbauer J, Pfisterer F, Coors S, Binder M, et al. Multi-objective hyperparameter optimization in machine learning—An overview. ACM Trans Evol Learn Optim 2022;3:1–50. [CrossRef]

[4] Yang L, Shami A. On hyperparameter optimization of machine learning algorithms: Theory and practice. Neurocomputing 2020;415:295–316. [CrossRef]
[5] Yu S, Ma P, Singh B, Silva S, Pritchard M. Two-step hyperparameter optimization method: Accelerating hyperparameter search by using a fraction of a training dataset. Artif Intell Earth Syst 2024;3:e230013. [CrossRef]
[6] Bartz E, Bartz-Beielstein T, Zaefferer M, Mersmann O. Hyperparameter tuning for machine and deep learning with R: A practical guide. Singapore: Springer Nature; 2023. [CrossRef]
[7] Chen S, Bolufé-Röhler A, Montgomery J, Zhang W, Hendtlass T. Using average-fitness based selection to combat the curse of dimensionality. In: IEEE Congress on Evolutionary Computation (CEC); 2022. p. 1–8. [CrossRef]
[8] Andonie R. Hyperparameter optimization in learning systems. J Membr Comput 2019;1:279–291.
[9] King D. Dlib-ml: A machine learning toolkit. J Mach Learn Res 2009;10:1755–1758. [CrossRef]
[10] Carney M, Webster B, Alvarado I, Phillips K, Howell N, Griffith J, et al. Teachable machine: Approachable web-based tool for exploring machine learning classification. In: Extended Abstracts of the 2020 CHI Conference on Human Factors in Computing Systems; 2020. CHI 2020, April 25–30, 2020, Honolulu, HI, USA. [CrossRef]
[11] LeDell E, Poirier S. H2O AutoML: Scalable automatic machine learning. In: 7th ICML Workshop on Automated Machine Learning (AutoML); 2020.
[12] Tambad S, Nandwani R, McIntosh SK. Analyzing programming languages by community characteristics on GitHub and StackOverflow. arXiv 2020;2006.01351. doi:10.48550/arXiv.2006.01351.
[13] Sengupta S, Haythornthwaite C. Learning with comments: An analysis of comments and community on Stack Overflow. In: Proceedings of the 53rd Hawaii International Conference on System Sciences; 2020; Hawaii, USA. p. 2898–2907. [CrossRef]
[14] Tehrani MJ, Arjomand P. Finding the potential accepted answer on Stack Overflow: A text mining approach. Research Square 2022. Preprint. doi: 10.21203/rs.3.rs-2067571/v1 [Epub ahead of print] [CrossRef]
[15] Storey M-A, Treude C, Van Deursen A, Cheng LT. The impact of social media on software engineering practices and tools. In: Proceedings of the FSE/SDP Workshop on Future of Software Engineering Research; 2010; New York, USA. p. 359–364. [CrossRef]
[16] Nasehi SM, Sillito J, Maurer F, Burns C. What makes a good code example? A study of programming Q&A in StackOverflow. In: Proceedings of the 28th IEEE International Conference on Software Maintenance; 2012; Trento, Italy. p. 25–34. [CrossRef]
[17] Vasilescu B, Filkov V, Serebrenik A. StackOverflow and GitHub: Associations between software development and crowdsourced knowledge. In: Proceedings of the 2013 International Conference on Social Computing; 2013; Alexandria, VA, USA. p. 188–195. [CrossRef]
[18] Sonam S, Verma A, Lal S, Sardana N. TagStack: Automated system for predicting tags in Stack Overflow. In: Proceedings of the 2019 International Conference on Signal Processing and Communication; 2019; India. p. 223–228. [CrossRef]
[19] Harrag F, Khamliche M. Mining Stack Overflow: A recommender systems-based model. Preprints 2020;2020080265. [CrossRef]
[20] He J, Xu B, Yang Z, Han D, Yang C, Lo D. PTM4Tag: Sharpening tag recommendation of Stack Overflow posts with pre-trained models. In: Proceedings of the 30th IEEE/ACM International Conference on Program Comprehension; 2022. p. 1–11. [CrossRef]
[21] Zhu W, Zhang H, Hassan AE, Godfrey MW. An empirical study of question discussions on Stack Overflow. Empir Softw Eng 2022;27:148. [CrossRef]
[22] Chen X, Xu F, Huang Y, Zhou X, Zheng Z. An empirical study of code reuse between GitHub and Stack Overflow during software development. J Syst Softw 2024;210:111964. [CrossRef]
[23] Bergstra J, Bengio Y. Random search for hyper-parameter optimization. J Mach Learn Res 2012;13:281–305.
[24] Liashchynskyi P, Liashchynskyi P. Grid search, random search, genetic algorithm: A big comparison for NAS. arXiv 2019;1912.06059.
[25] Bischl B, Binder M, Lang M, Pielok T, Richter J, Coors S, et al. Hyperparameter optimization: Foundations, algorithms, best practices, and open challenges. Wiley Interdiscip Rev Data Min Knowl Discov 2023;13:e1484. [CrossRef]
[26] Hossain S, Islam F, Tayeb NT, Aslam M, Kim JH. Performance enhancement of the micromixer by the multiobjective genetic algorithm and surrogate model based on a Navier–Stokes analysis using trade-off objective functions. Math Probl Eng 2021;2021:9924849. [CrossRef]
[27] Wang C, Wilhelm M, Stuber M. Semi-infinite optimization with hybrid models. Ind Eng Chem Res 2022. [CrossRef]
[28] Azevedo B, Rocha A, Pereira A. Hybrid approaches to optimization and machine learning methods. In: 2023 IEEE 10th International Conference on Data Science and Advanced Analytics (DSAA); 2023. p. 1–2. [CrossRef]
[29] Uslu MF, Uslu S, Bulut F. An adaptive hybrid approach: Combining genetic algorithm and ant colony optimization for integrated process planning and scheduling. Appl Comput Informatics 2022;18:101–112. [CrossRef]

[30] Abu Arqub O, Abo-Hammour Z. Numerical solution of systems of second-order boundary value problems using continuous genetic algorithm. Inf Sci 2014;279:396–415. [CrossRef]
[31] Abo-Hammour Z, Abu Arqub O, Momani S, Shawagfeh N. Optimization solution of Troesch's and Bratu's problems of ordinary type using novel continuous genetic algorithm. Discrete Dyn Nat Soc 2014;2014:401696. [CrossRef]
[32] Parouha R, Verma P. Design and applications of an advanced hybrid meta-heuristic algorithm for optimization problems. Artif Intell Rev 2021;54:5931–6010. [CrossRef]
[33] Akiba T, Sano S, Yanase T, Ohta T, Koyama M. Optuna: A next-generation hyperparameter optimization framework. In: Proceedings of the 25th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining; 2019; New York, USA. p. 2623–2631. [CrossRef]
[34] Golovin D, Solnik B, Moitra S, Kochanski G, Karro J, Sculley D. Google Vizier: A service for black-box optimization. In: Proceedings of the 23rd ACM SIGKDD International Conference on Knowledge Discovery and Data Mining; 2017; Halifax, Canada. p. 1487–1495. [CrossRef]
[35] Liaw R, Liang E, Nishihara R, Moritz P, Gonzalez JE, Stoica I. Tune: A research platform for distributed model selection and training. arXiv 2018;1807.05118.
[36] Microsoft. Neural Network Intelligence. GitHub; 2018. Available at: https://github.com/microsoft/nni. Accessed on Oct 07, 2024.
[37] Yuan B, Gui S, Zhang Q, Wang Z, Wen J, Mao B, et al. FairerML: An extensible platform for analysing, visualising, and mitigating biases in machine learning. IEEE Comput Intell Mag 2024;19:129–141. [CrossRef]
[38] Mansion T, Braud R, Amrani A, Chaouche S, Adjed F, Cantat L. Debiai: Open-source toolkit for data analysis, visualisation and evaluation in machine learning. In: 20th International Conference on Autonomic and Autonomous Systems; 2024; Athens, Greece.
[39] Jia L, Guo L, Zhou Z, Li Y. LAMDA-SSL: A comprehensive semi-supervised learning toolkit. Sci China Inf Sci 2023;67:1. [CrossRef]
[40] Lauer F. MLWeb: A toolkit for machine learning on the web. Neurocomputing 2018;282:74–77. [CrossRef]
[41] Kaluarachchi T, Wickramasinghe M. WebDraw: A machine learning-driven tool for automatic website prototyping. Sci Comput Program 2024;233:103056. [CrossRef]
[42] Erickson N, Mueller J, Shirkov A, Zhang H, Larroy P, Li M, Smola A. AutoGluon-Tabular: Robust and accurate AutoML for structured data. arXiv 2020;2003.06505.
[43] Pedregosa F, Varoquaux G, Gramfort A, Michel V, Thirion B, Grisel O, et al. Scikit-learn: Machine learning in Python. J Mach Learn Res 2011;12:2825–2830.
[44] Feurer M, Klein A, Eggensperger K, Springenberg J, Blum M, Hutter F. Auto-sklearn: Efficient and robust automated machine learning. In: Advances in Neural Information Processing Systems; 2015. p. 2962–2970.
[45] Olson RS, Bartley N, Urbanowicz RJ, Moore JH. Evaluation of a tree-based pipeline optimization tool for automating data science. In: Proceedings of GECCO; 2016. p. 485–492. [CrossRef]
[46] Kuhn M. Building predictive models in R using the caret package. J Stat Softw 2008;28:1–26. [CrossRef]
[47] Yan Y. rBayesOptimization: Bayesian optimization of hyperparameters. CRAN; 2016. [CrossRef]
[48] Ridgeway G. Generalized boosted models: A guide to the GBM package. 2007.
[49] Dimitriadou E, Hornik K, Leisch F, Meyer D, Weingessel A. E1071: Misc functions of the Department of Statistics (E1071), TU Wien; 2009.
[50] Liaw A, Wiener M. Classification and regression by randomForest. R News 2002;2:18–22.
[51] Wickham H. ggplot2: Elegant graphics for data analysis. Springer; 2016. [CrossRef]
[52] Huang Q, Mao J, Liu Y. An improved grid search algorithm of SVR parameters optimization. In: 2012 IEEE 14th International Conference on Communication Technology; 2012; Chengdu, China. p. 1022–1026. [CrossRef]
[53] Larochelle H, Erhan D, Courville A, Bergstra J, Bengio Y. An empirical evaluation of deep architectures on problems with many factors of variation. In: Proceedings of ICML; 2007. p. 473–480. [CrossRef]
[54] Wu J, Chen XY, Zhang H, Xiong LD, Lei H, Deng SH. Hyperparameter optimization for machine learning models based on Bayes optimization. J Electron Sci Technol 2019;17:26–40.
[55] Marinov D, Karapetyan D. Hyperparameter optimisation with early termination of poor performers. In: 2019 11th Computer Science and Electronic Engineering Conference (CEEC); 2019; Colchester, UK. p. 160–163. [CrossRef]
[56] Kolda TG, Lewis RM, Torczon V. Optimization by direct search: New perspectives on some classical and modern methods. SIAM Rev 2003;45:385–482. [CrossRef]
[57] Cortes C, Vapnik V. Support-vector networks. Mach Learn 1995;20:273–297. [CrossRef]
[58] Feurer M, Hutter F. Hyperparameter optimization. In: Automated Machine Learning: Methods, Systems, Challenges. Springer; 2019. p. 3–33. [CrossRef]
[59] Boser BE, Guyon IM, Vapnik VN. A training algorithm for optimal margin classifiers. In: Proceedings of the 5th Annual ACM Workshop on Computational Learning Theory; 1992; New York, USA. p. 144–152. [CrossRef]

[60] Breiman L. Random forests. Mach Learn 2001;45:5–32. [CrossRef]
[61] Probst P, Bischl B, Boulesteix AL. Tunability: Importance of hyperparameters of machine learning algorithms. arXiv 2018;1802.09596.
[62] Friedman JH. Greedy function approximation: A gradient boosting machine. Ann Stat 2001;29:1189–1232. [CrossRef]
[63] Read J, Pfahringer B, Holmes G. Multi-label classification using ensembles of pruned sets. In: 2008 IEEE International Conference on Data Mining; 2008; Pisa, Italy. p. 995–1000. [CrossRef]
[64] Chen H, Tiňo P, Yao X. Predictive ensemble pruning by expectation propagation. IEEE Trans Knowl Data Eng 2009;21:999–1013. [CrossRef]
[65] Read J, Pfahringer B, Holmes G, Frank E. Classifier chains for multi-label classification. Mach Learn 2011;85:333–359. [CrossRef]
[66] Rivolli A, de Carvalho ACPLF. The utiml package: Multi-label classification in R. R J 2019;10:24–37. [CrossRef]
[67] Kaggle. Stack Overflow dataset. Available at: https://www.kaggle.com/datasets/stackoverflow/stacksample. Accessed on Oct 17, 2023.
[68] Feinerer I. wordnet package. CRAN; 2024. Available from: https://cran.r-project.org/web/packages/wordnet/.
[69] Fellows I. wordcloud package. CRAN; 2024. Available at: https://cran.r-project.org/web/packages/wordcloud/
[70] Peruma A, Simmons S, AlOmar EA, Newman CD, Mkaouer MW, Ouni A. How do I refactor this? An empirical study on refactoring trends and topics in Stack Overflow. Empir Softw Eng 2022;27:1. [CrossRef]