AutoML Framework for Classical Machine Learning

Teja Kiran Reddy Sirivella
Rochester Institute of Technology
United States
ts7244@g.rit.edu

1 INTRODUCTION

Due to the success of machine learning techniques in recent times, it has been incorporated into many applications to provide advanced features to attract users. Every machine learning task involves collecting data, training, validation, and testing to produce a working model that best fits the required task. This is a long and tedious exploration process. Typically, data scientists use their domain knowledge and a set of experiments to obtain the best algorithm and tune hyper-parameters to achieve the best test accuracy. This process demands a lot of resources and expertise to arrive at the solution and might not be suitable for all skill levels or resource budgets.

"Automated Machine Learning" is a framework that aims to enable non-machine learning experts to arrive at the best algorithm and hyper-parameters for their data set thereby saving a lot of resources and helping focus on the business and data aspects. It bridges the gap between seasoned experts and non-professionals by efficiently exploring the potential algorithms and hyper-parameters within resource constraints. [1].Several frameworks, such as Auto-Weka[15], Auto-sklearn[7] and Hyperopt[4], have been introduced to solve this problem with varying scopes and techniques. The scope of the framework is restricted to classical machine learning algorithms to narrow down the search space.

AutoML comprises search space and search algorithms that explore the search space to optimize the performance of the model. Many such algorithms have been published to reduce the time and resources needed to achieve the desired result. However, due to widespread usage of large datasets and a wide range of hyper-parameters, even an efficient algorithm might not be able to converge within the desired resource budget. This sparked the idea of using historical experiments on datasets and similarity metrics and leveraging the results of similar datasets to find a set of the best algorithms and warm start the search process with hyper-parameters. This process is known as "Meta-Learning"[8].

The current framework leverages meta-learning to extract a portfolio [6] of promising configurations and warm-start Bayesian optimization using Random Forest as the surrogate model and Expected Improvement as the acquisition function, in order to provide top configurations for a dataset within the scope of classical machine learning models.

2 RELATED WORK

The earliest framework, Auto-WEKA [15], was designed to address both hyperparameter tuning and model selection, often referred to as the CASH problem, by leveraging innovations in search strategies such as Bayesian optimization. The performance of the framework was significantly better than that achieved by traditional model selection and hyperparameter optimization methods.

Several search strategies [2], supporting both current and traditional frameworks for hyperparameter optimization, were explored, including random search and sequential model-based methods. The study demonstrates the effectiveness of these strategies over manual tuning.

Random search [3] is a popular strategy that selects configurations at random for exploration and often yields more promising results than grid search or manual tuning under the same computational budget.

Bayesian optimization [14] was introduced to improve existing search strategies such as grid search and random search. It employs surrogate models, specifically Gaussian Processes, along with an acquisition

CSCI 788, Machine Learning, Project Report
2025. ACM ISBN 978-x-xxxx-xxxx-x/YY/MM
https://doi.org/10.1145/nnnnnnn.nnnnnnn

function to select the most promising configuration in each iteration. This approach has been shown to outperform traditional methods in terms of optimization performance.

Random Forests [5] were introduced as an alternative surrogate model in Bayesian optimization to better support categorical and conditional hyperparameters, which are not effectively handled by Gaussian Processes.

SMBO [9] is an optimization technique within the Bayesian optimization framework, designed to handle general algorithm configuration tasks involving both categorical and numerical parameters. It introduced Sequential Model-Based Algorithm Configuration (SMAC), a specific SMBO method tailored for such complex search spaces.

Reinforcement learning [12] was explored as an alternative search strategy for efficient hyperparameter optimization, utilizing value-based and policy-based approaches. It incorporates meta-learning to leverage prior experience and accelerate learning on new tasks.

Alternatively, optimizing hyperparameter search in complex and deep neural networks requires a more efficient search process due to the large search space. [11] , Auto-Keras [10] and CMA-ES-based approaches [13] introduced Neural Architecture Search (NAS), which focuses on optimizing architectural hyperparameters in detail.

A key limitation of existing methods is their failure to leverage historical data to accelerate configuration optimization. To address this, methods like [8] and Auto-sklearn [7] employed meta-learning, using meta-features and similarity measures across datasets to suggest promising configurations and warm-start search strategies like Bayesian optimization.

The computation of meta-features for a new dataset can be time-consuming, depending on its dimensionality and size. OBOE [17] addressed this by training a meta-learning model on historical data to estimate meta-features instantly, enabling faster initialization of the optimization process.

Auto-sklearn 2.0 [6] proposed an alternative use of meta-learning by eliminating the reliance on meta-features altogether. It generates a portfolio of configurations in an offline phase and leverages this portfolio to obtain an initial set of configurations for the algorithms.

Wistuba et al. [16] introduced the use of a greedy algorithm to construct such portfolio configurations efficiently across multiple datasets.

3 METHODOLOGY

3.1 Datasets

Datasets are chosen from the OpenML platform suite "OpenML-CC18", which consists of 72 diverse datasets varying in the number of features, samples, and class distributions. To obtain a representative set for portfolio construction, datasets with fewer than 500 rows or more than 100,000 rows are excluded. Datasets containing more than 50% sparsity, or those related to time series or textual data, are also removed. All datasets are required to have at least two attributes.

This filtering results in 62 datasets, which are then subjected to data cleaning. During this process, missing values are imputed based on feature type: categorical features are imputed with the most frequent value, while numerical features are imputed with the mean.

3.2 Configuration Space

The framework defines a configuration space for supported classical machine learning models, each with its own set of hyperparameters and value ranges. This space is modeled as a conditional and hierarchical search space. The model type is treated as a top-level categorical hyperparameter. Each model has its own associated hyperparameters, which are conditionally active only when that specific model is selected. An overview of the configuration space is provided in Table 1.

3.3 Portfolio construction

Portfolio construction is a meta-learning technique used to initialize the search process with top-performing configurations. Traditional meta-learning approaches rely on meta-features to identify similar datasets and use their best configurations to warm-start optimization. However, computing meta-features, especially landmarking features, can be time-consuming and resource-intensive.

To address this, we adopt a meta-feature-free meta-learning approach to construct a configuration portfolio.

Initially, the top configuration for each dataset is collected from the OpenML platform. These configurations are then evaluated across all datasets to form a performance matrix, where each entry represents the accuracy of a configuration on a specific dataset.

To build a portfolio up to a predefined size, configurations are greedily selected based on their ability to reduce the estimated generalization error across all meta-datasets. This results in a portfolio containing diverse configurations that generalize well across a wide variety of datasets.

To efficiently construct the 62 × 62 performance matrix, the Ray framework was deployed on a distributed cluster with 164 CPUs across seven machines. Ray's dashboard was used to monitor task execution, and results were stored on the head node for analysis.

For a new dataset, all configurations in the portfolio are evaluated using a budgeting strategy (Successive Halving), and the top-k configurations are selected to warm-start the Bayesian optimization process.

3.4 Successive Halving

Successive Halving is a budget allocation strategy used to evaluate a large number of configurations under limited time or computational constraints. It progressively allocates resources by pruning poorly performing configurations early and assigning more resources to promising ones. In each iteration, a fixed number of configurations are evaluated with a minimal budget (e.g., number of epochs). The top fraction (typically 1/eta) based on performance is selected to proceed to the next round, where their budget is increased by a factor of eta. This process is repeated until only a few configurations remain, which are evaluated with the maximum budget.

In our framework, Successive Halving is used to evaluate the portfolio configurations on a new dataset. This enables the system to efficiently identify the most promising candidates without expending the full budget on every configuration. The top-k configurations selected from this process are then used to warm-start the Bayesian optimization process.

3.5 Bayesian Optimization with SMAC

Sequential Model-based Algorithm Configuration (SMAC) is a framework that jointly optimizes both the machine learning model and its hyperparameters by implementing the Bayesian Optimization strategy to find the best-performing configuration for a given dataset. It uses a surrogate model to predict the performance of a given configuration based on previously evaluated configurations and their observed accuracies. After each evaluation, the surrogate model is updated with the new observations, improving the quality of future predictions. An acquisition function is used to identify the most promising configuration to evaluate next, balancing exploration and exploitation.

The AutoML framework uses SMAC with a Random Forest surrogate model, which supports hierarchical and conditional configuration spaces. It uses Expected Improvement (EI) as the acquisition function to guide the optimization. While SMAC, by default, samples initial configurations randomly to train the surrogate model, the current framework instead passes portfolio configurations to SMAC to warm-start the optimization process with promising candidates.

To handle resource constraints, the AutoML framework employs SMAC's Multi-Fidelity Facade, which allows specifying minimum and maximum budgets (e.g., number of epochs or iterations), as well as time-based constraints such as the wall time limit.

Additionally, the AutoML framework provides a clean user interface that accepts a dataset and optional parameters. It automatically performs one-hot encoding for categorical features and assigns default values for unspecified arguments, streamlining the process for users.

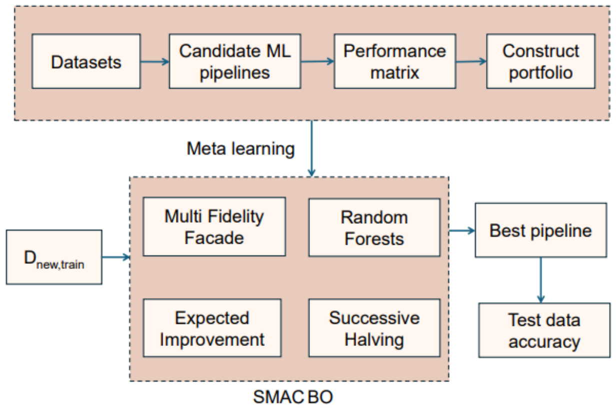

Figure 1: AutoML Framework Workflow

Hyperparameter	**Domain**	**Default**	**Log**
Classifier	{Extra Trees, Gradient Boosting, MLP, Random Forest, Passive Aggressive, SGD}	-	-
Extra Trees: Bootstrap	{True, False}	False	-
Extra Trees: Criterion	{gini, entropy}	gini	-
Extra Trees: Max Features	[0.0, 1.0]	0.5	No
Extra Trees: Min Samples Leaf	[1, 20]	1	No
Extra Trees: Min Samples Split	[2, 20]	2	No
Gradient Boosting: Early Stopping	{off, valid, train}	off	-
Gradient Boosting: L2 Regularization	[1e^{-10}, 1.0]	0.0	Yes
Gradient Boosting: Learning Rate	[0.01, 1.0]	0.1	Yes
Gradient Boosting: Max Leaf Nodes	[3, 2047]	31	Yes
Gradient Boosting: Min Samples Leaf	[1, 200]	20	Yes
Gradient Boosting: N Iter No Change	[1, 20]	10	No
Gradient Boosting: Validation Fraction	[0.01, 0.4]	0.1	No
MLP: Activation	{tanh, relu}	relu	-
MLP: Alpha	[1e^{-7}, 0.1]	0.0001	Yes
MLP: Early Stopping	{valid, train}	valid	-
MLP: Hidden Layer Depth	[1, 3]	1	No
MLP: Learning Rate Init	[0.0001, 0.5]	0.001	Yes
MLP: Num Nodes Per Layer	[16, 264]	32	Yes
Passive Aggressive: C	[1e^{-5}, 10.0]	1.0	Yes
Passive Aggressive: Average	{False, True}	False	-
Passive Aggressive: Loss	{hinge, squared hinge}	hinge	-
Passive Aggressive: Tol	[1e^{-5}, 0.1]	0.0001	Yes
Random Forest: Bootstrap	{True, False}	True	-
Random Forest: Criterion	{gini, entropy}	gini	-
Random Forest: Max Features	[0.0, 1.0]	0.5	No
Random Forest: Min Samples Leaf	[1, 20]	1	No
Random Forest: Min Samples Split	[2, 20]	2	No
SGD: Alpha	[1e^{-7}, 0.1]	0.0001	Yes
SGD: Average	{False, True}	False	-
SGD: Epsilon	[1e^{-5}, 0.1]	0.0001	Yes
SGD: Eta0	[1e^{-7}, 0.1]	0.01	Yes
SGD: L1 Ratio	[1e^{-9}, 1.0]	0.15	Yes
SGD: Learning Rate	{optimal, invscaling, constant}	invscaling	-
SGD: Loss	{hinge, log, modified Huber, squared hinge, perceptron}	log	-
SGD: Penalty	{l1, l2, elasticnet}	l2	-
SGD: Power T	[1e^{-5}, 1.0]	0.5	No
SGD: Tol	[1e^{-5}, 0.1]	0.0001	Yes

Table 1: Configuration Space for AutoML Classifiers

4 EXPERIMENTAL SETUP

Tasks were collected from the AutoML benchmark suite available on the OpenML platform. Datasets were extracted from these tasks, preprocessed, and cleaned before experimentation. The selected test datasets range from a minimum of approximately 500 rows to a maximum of 100,000 rows, with the number of features ranging from 846 to 1,637.

The first set of experiments evaluates the impact of meta-learning on the framework's performance. These experiments were conducted under controlled resource constraints: a maximum runtime of 10 minutes per task, a minimum budget of 10, a maximum budget of 500, and a memory limit of 6 GB per task. Test accuracies are reported based on the best configurations suggested by the framework.

The second set of experiments compares the proposed framework, enhanced with meta-learning, against Auto-sklearn under similar resource constraints.

All experiments were executed on a distributed cluster of machines, using the Ray framework to parallelize training and testing tasks across datasets.

5 RESULTS

The experiment evaluating the impact of meta-learning shows considerable improvement across the majority of datasets. Since meta-learning is particularly effective under shorter optimization budgets, the time limit was set to 10 minutes per task. As illustrated in the Figure 2, approximately 80% of the datasets either exhibited improved test accuracy or achieved similar performance, with a difference of less than 1%. These results demonstrate that constructing a portfolio and warm-starting the framework leads to better optimization outcomes under resource constraints.

The benchmark comparison against Auto-sklearn, illustrated in Figure 3, shows that the proposed framework achieves competitive test accuracy across the evaluated datasets. The proposed framework outperformed Auto-sklearn on approximately 50% of the datasets, while 30% of the datasets exhibited a marginal difference of less than 1% in test accuracy. In cases where there was a large drop in accuracy, it was primarily due to the dataset size: the models could not complete full training within the allocated time budget. The proposed framework occasionally achieved slightly higher accuracy in these cases because it does not implement partial fitting, allowing configurations that are mid-training to exceed the budget and complete, unlike Auto-sklearn.

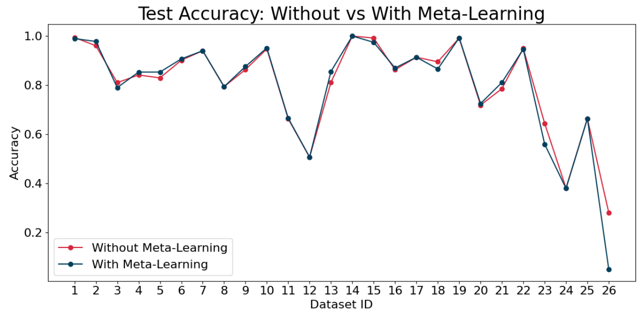

Figure 2: Test accuracy comparison with and without meta-learning. Meta-learning improves performance on approximately 80% of datasets under a 10-minute optimization budget.

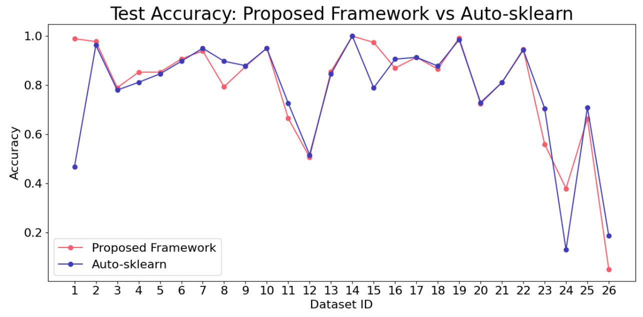

Figure 3: Test accuracy comparison between the proposed framework and Auto-sklearn. The proposed framework outperformed Auto-sklearn on approximately 50% of the datasets, with 30% showing marginal differences of less than 1% under a 10-minute optimization budget.

6 CONCLUSION

In this project, we developed an AutoML framework for classical machine learning algorithms, incorporating meta-learning to accelerate the optimization process. By constructing a portfolio of configurations and using it to warm-start Bayesian optimization, the framework improved early performance across many datasets under tight resource constraints. Experiments demonstrated that meta-learning led to better optimization efficiency and improved test accuracy on approximately 80% of the benchmark datasets.

Furthermore, when compared to Auto-sklearn under identical computational budgets, the proposed framework achieved competitive performance, outperforming Auto-sklearn on about 50% of the datasets and achieving near-equal results on another 30%. These results validate the effectiveness of integrating meta-learning into the AutoML pipeline for improving efficiency and competitiveness in real-world constrained environments.

REFERENCES

[1] Mitra Baratchi, Can Wang, Steffen Limmer, Jan N. van Rijn, Holger Hoos, Thomas Bäck, and Markus Olhofer. Automated machine learning: Past, present and future. *Artificial Intelligence Review*, 57(5):122, April 2024.

[2] James Bergstra, Rémi Bardenet, Yoshua Bengio, and Balázs Kégl. Algorithms for hyper-parameter optimization. In *Proceedings of the 25th International Conference on Neural Information Processing Systems*, NIPS'11, page 2546–2554, Red Hook, NY, USA, 2011. Curran Associates Inc.

[3] James Bergstra and Yoshua Bengio. Random search for hyper-parameter optimization. *J. Mach. Learn. Res.*, 13(null):281–305, February 2012.

[4] James Bergstra, Brent Komer, Chris Eliasmith, Dan Yamins, and David Cox. Hyperopt: A python library for model selection and hyperparameter optimization. *Computational Science Discovery*, 8:014008, 07 2015.

[5] Leo Breiman. Random forests. *Mach. Learn.*, 45(1):5–32, October 2001.

[6] Matthias Feurer, Katharina Eggensperger, Stefan Falkner, Marius Lindauer, and Frank Hutter. Auto-sklearn 2.0: hands-free automl via meta-learning. *J. Mach. Learn. Res.*, 23(1), January 2022.

[7] Matthias Feurer, Aaron Klein, Katharina Eggensperger, Jost Tobias Springenberg, Manuel Blum, and Frank Hutter. *Auto-sklearn: Efficient and Robust Automated Machine Learning*, pages 113–134. Springer International Publishing, Cham, 2019.

[8] Matthias Feurer, Jost Tobias Springenberg, and Frank Hutter. Using meta-learning to initialize bayesian optimization of hyperparameters. In *Proceedings of the 2014 International Conference on Meta-Learning and Algorithm Selection - Volume 1201*, MLAS'14, page 3–10, Aachen, DEU, 2014. CEUR-WS.org.

[9] Frank Hutter, Holger H. Hoos, and Kevin Leyton-Brown. Sequential model-based optimization for general algorithm configuration. In *Proceedings of the 5th International Conference on Learning and Intelligent Optimization*, LION'05, page 507–523, Berlin, Heidelberg, 2011. Springer-Verlag.

[10] Haifeng Jin, Qingquan Song, and Xia Hu. Auto-keras: An efficient neural architecture search system. In *Proceedings of the 25th ACM SIGKDD International Conference on Knowledge Discovery & Data Mining*, KDD '19, page 1946–1956, New York, NY, USA, 2019. Association for Computing Machinery.

[11] Chenxi Liu, Barret Zoph, Maxim Neumann, Jonathon Shlens, Wei Hua, Li-Jia Li, Li Fei-Fei, Alan Yuille, Jonathan Huang, and Kevin Murphy. Progressive neural architecture search. In *Computer Vision – ECCV 2018: 15th European Conference, Munich, Germany, September 8-14, 2018, Proceedings, Part I*, page 19–35, Berlin, Heidelberg, 2018. Springer-Verlag.

[12] Xiyuan Liu, Jia Wu, and Senpeng Chen. Efficient hyperparameters optimization through model-based reinforcement learning and meta-learning. In *2020 IEEE 22nd International Conference on High Performance Computing and Communications; IEEE 18th International Conference on Smart City; IEEE 6th International Conference on Data Science and Systems (HPCC/SmartCity/DSS)*, pages 1036–1041, 2020.

[13] Ilya Loshchilov and Frank Hutter. Cma-es for hyperparameter optimization of deep neural networks. *ArXiv*, abs/1604.07269, 2016.

[14] Jasper Snoek, Hugo Larochelle, and Ryan P. Adams. Practical bayesian optimization of machine learning algorithms. In *Proceedings of the 26th International Conference on Neural Information Processing Systems - Volume 2*, NIPS'12, page 2951–2959, Red Hook, NY, USA, 2012. Curran Associates Inc.

[15] Chris Thornton, Frank Hutter, Holger H. Hoos, and Kevin Leyton-Brown. Auto-weka: combined selection and hyperparameter optimization of classification algorithms. In *Proceedings of the 19th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, KDD '13, page 847–855, New York, NY, USA, 2013. Association for Computing Machinery.

[16] Martin Wistuba, Nicolas Schilling, and Lars Schmidt-Thieme. Sequential model-free hyperparameter tuning. In *2015 IEEE International Conference on Data Mining*, pages 1033–1038, 2015.

[17] Chengrun Yang, Yuji Akimoto, Dae Won Kim, and Madeleine Udell. Oboe: Collaborative filtering for automl model selection. In *Proceedings of the 25th ACM SIGKDD International Conference on Knowledge Discovery & Data Mining*, KDD '19, page 1173–1183, New York, NY, USA, 2019. Association for Computing Machinery.