Autostacker: A Compositional Evolutionary Learning System

Boyuan Chen
Columbia University
New York City, New York, USA
bchen@cs.columbia.edu

Harvey Wu
Columbia University
New York City, New York, USA
wu.harvey@columbia.edu

Warren Mo
University of Chicago
Chicago, Illinois, USA
warrenmo@uchicago.edu

Ishanu Chattopadhyay
University of Chicago
Chicago, Illinois, USA
ishanu@uchicago.edu

Hod Lipson
Columbia University
New York City, New York, USA
hod.lipson@columbia.edu

ABSTRACT

In this work, an automatic machine learning (AutoML) modeling architecture called Autostacker is introduced. Autostacker combines an innovative hierarchical stacking architecture and an evolutionary algorithm (EA) to perform efficient parameter search without the need for prior domain knowledge about the data or feature preprocessing. Using EA, Autostacker quickly evolves candidate pipelines with high predictive accuracy. These pipelines can be used in their given form, or serve as a starting point for further augmentation and refinement by human experts. Autostacker finds innovative machine learning model combinations and structures, rather than selecting a single model and optimizing its hyperparameters. When its performance on fifteen datasets is compared with that of other AutoML systems, Autostacker produces superior or competitive results in terms of both test accuracy and time cost.

CCS CONCEPTS

• **Computing methodologies** → **Supervised learning**: *Genetic algorithms*;

KEYWORDS

AutoML, Machine Learning, Evolutionary Machine Learning

ACM Reference Format:

Boyuan Chen, Harvey Wu, Warren Mo, Ishanu Chattopadhyay, and Hod Lipson. 2018. Autostacker: A Compositional Evolutionary Learning System. In *GECCO '18: Genetic and Evolutionary Computation Conference, July 15–19, 2018, Kyoto, Japan*. ACM, New York, NY, USA, 8 pages. <https://doi.org/10.1145/3205455.3205586>

1 INTRODUCTION

Wolpert's No Free Lunch theorem [35] implies that no model can be expected to generalize well to all data. Based on this premise, upon encountering each new dataset, Machine Learning practitioners

Permission to make digital or hard copies of all or part of this work for personal or classroom use is granted without fee provided that copies are not made or distributed for profit or commercial advantage and that copies bear this notice and the full citation on the first page. Copyrights for components of this work owned by others than the author(s) must be honored. Abstracting with credit is permitted. To copy otherwise, or republish, to post on servers or to redistribute to lists, requires prior specific permission and/or a fee. Request permissions from permissions@acm.org.

GECCO '18, July 15–19, 2018, Kyoto, Japan

© 2018 Copyright held by the owner/author(s). Publication rights licensed to Association for Computing Machinery.

ACM ISBN 978-1-4503-5618-3/18/07...\$15.00

<https://doi.org/10.1145/3205455.3205586>

Figure 1: A typical pipeline generated by Autostacker. Each column represents a layer, whereas each node in a layer represents a machine learning primitive model (e.g., SVM, MLP). The number of layers and nodes per layer can be specified a priori, or treated as a hyperparameter. A raw dataset is used as input for the first layer. In the subsequent layers, the prediction results from each node will be added to the raw dataset as synthetic features (new colors). The new dataset generated by each layer serves as input to the next layer.

must ask: *What models can we use, and how can we select the best hyperparameters for our chosen model?* Successful model selection requires considerable experience and knowledge, as optimal hyperparameter identification often necessitates time-intensive tuning. Automating both parts of the modeling procedure—model selection and hyperparameter optimization—would render the output of machine learning accessible to a wider community, making it highly relevant to both academia and industry.

An AutoML system aims to achieve this objective by providing an automatically generated baseline to facilitate solving machine learning problems. Such a system takes in a formatted dataset as input and outputs one or more modeling pipelines that achieve reasonable performance on the dataset. Recent efforts in AutoML, such as AutoSklearn[7] and TPOT[19], demonstrate success when applied on a variety of datasets.

In this work, an AutoML architecture called Autostacker is introduced. Inspired by the stacking method[35][2] of ensemble learning, Autostacker automatically discovers pipelines comprising of one or

Figure 2: An overview of pipeline generation. We randomly generate initial pipelines and feed those into the basic EA algorithm, looping the process to generate winning pipelines. The hyperparameters of each pipeline (number of layers and nodes) can be explicitly defined by the user or tuned by Autostacker.

many models. When its performance on fifteen datasets is compared to that of other AutoML frameworks, Autostacker demonstrates competitiveness in both accuracy and time cost. The following three properties of Autostacker allow it to generalize well to new data:

- **Cascading** Despite the growing popularity of "big data", many datasets are still small and sparse. In this work, this challenge is overcome through cascading, i.e., by always using the original dataset in all stacking layers while concatenating synthetic features in each stacking layer. More details are provided in the Approach section below.
- **Model Flexibility** Existing AutoML frameworks generate a full pipeline that includes data preprocessing, feature engineering, and model selection. Model selection usually involves optimization of a single machine learning primitive, such as a Support Vector Machine (SVM)[5], or a traditional ensemble method, such as Boosting[13][27][8]. Autostacker permits flexible combinations of many machine learning primitives, resulting in a larger search space.
- **Evolutionary Search Algorithm** EAs allow good solutions to be tractably found in a large variable space [6]. Such variables include the type of primitive machine learning models, the framework configuration settings (for instance, the number of primitive models in each stacking layer) and the hyperparameters in each primitive model. In the present work, all of the aforementioned elements are considered as hyperparameters. Instead of treating AutoML as an optimization problem [7], in Autostacker, it is modeled as a search problem pertaining to this large space of hyperparameters. Exploiting the parallel nature of evolutionary algorithms, Autostacker quickly finds good candidate pipelines. As shown in the Results section, competitive performance is achieved with only a very basic version of EA.

2 RELATED WORK

2.1 Stacking and Cascading

Stacking is a decades-old method of ensemble learning [35]. The original dataset serves as the input to the first layer, whereby the outputs of the classifiers in the first layer are input into the next layer, and the same process applies to all subsequent layers. Intuitively, the latter layers can identify and correct the mistakes that the classifiers in the preceding layers have made. The related approach of cascading—taking the output of one model and using it as the input to another—was first explored as an ensemble learning technique by Viola and Jones [34]. In this approach, data is subjected to a series of binary classifiers. If a classifier outputs true, the data is transmitted to the next classifier. Conversely, if a classifier outputs false, the iteration ends and the cascade returns false. Thus, if the last classifier outputs true, the entire cascade returns true. Cascaded classification models (CCMs), a more sophisticated approach to cascading, were introduced by Heitz et al [11], as a way to decompose the complex problem of scene understanding into constituent problems.

2.2 Automatic Machine Learning

Extant AutoML research has focused on combining machine learning pipeline building and intelligent model hyperparameter search. One of the tools based on this approach, Auto-Weka [30][14] selects a single machine learning primitive and optimizes its hyperparameters. Auto-Weka expands upon the widely used Weka [10], as it uses Bayesian optimization (sequential model-based optimization) to search for optimal hyperparameter settings of the pipeline. In this context, the pipeline follows the traditional machine learning work process, starting with data preprocessing, through feature engineering, to single model prediction. However, fixed-order pipelines, especially with a single model prediction, are unsuitable for complex problems or small sample datasets. A similar methodology was adopted when developing AutoSklearn [7], where the scikit-learn [23] machine learning library serves as a toolbox, and Bayesian optimization is employed to tune the hyperparameters.

Several works on Bayesian optimization are designed specifically for large-scale parameter configuration problems such as AutoML. For example, RoBO [29] includes multiple implementations of different Bayesian optimization algorithms, while allowing the components of this process to be modified based on requirements. Hyperopt [1], on the other hand, takes advantage of sequential model-based optimization and considers the choice of classification models and preprocessing models jointly, as an integral optimization problem. Other Bayesian approaches for large-scale parameter search include SMAC [12] and Spearmint [28].

The use of EAs to perform hyperparameter optimization in an AutoML setting was recently explored in the TPOT architecture [21]. Extending the traditional so-called data scientists' pipeline used in AutoWeka and AutoSklearn, TPOT allows for parallel feature engineering prior to model prediction. Subsequently, evolutionary algorithms are employed, permitting the parameter configuration problem to be treated as a search problem.

All of the aforementioned approaches, however, focus on configuring a single machine learning primitive, with traditional ensemble architectures as a supplement. For example, while AutoSklearn allows ensemble models to be built in real time, it only considers traditional ensemble approaches. Similarly, Ensemble Selection [3] performs stacking [35] efficiently, while underperforming on gradient-free numerical optimization, with a tendency to overfit [7].

Autostacker, on the other hand, is an ensemble method by default. Consequently, it handles single model and ensemble approaches simultaneously as basic primitives. The cascading architectures generated by Autostacker permit synergistic combinations of ML primitives to "correct each other's mistakes" and improve generalization. Moreover, Autostacker allows multiple ensemble models to be used in the same architecture. Thus, as ensemble learning is generally more robust and can outperform individual models most of the time [22][25][26], it should be examined in more depth in the AutoML context. Some similar efforts have been made as in Freeze Thaw Ensemble Construction [17] and H2O [9].

In this work, instead of taking the traditional route of designing an AutoML system that learns to choose a single model and optimize it, Autostacker is designed to find innovative combinations or arrangements of ML primitives. This model flexibility is posited to contribute considerably to the Autostacker's empirical success when compared to other AutoML systems.

However, stacking models on top of each other renders the search space much larger than that of single-model AutoML systems, such as TPOT or AutoSklearn. Moreover, as the primitives in a candidate pipeline need to be optimized as well, this feature further exacerbates this issue. In this work, this problem is overcome by using a basic evolutionary algorithm, rather than Bayesian optimization, to identify the most suitable hyperparameters. EAs have recently seen a renaissance in other machine learning fields, such as neural network optimization and reinforcement learning [18][24], confirming their superior performance when applied to large search spaces. It should be noted that TPOT also uses EAs to perform parameter search.

3 METHODS

3.1 Problem Setting

In supervised learning, a model serves as a mapping function f^i from the sample input data $\mathbf{X}$ to the output data $\mathbf{Y}$:

$$\mathbf{Y} \leftarrow f_{\mathbf{H}, \Theta}^i(\mathbf{X}) \quad (1)$$

The model f^i , belonging to a family of models F , is governed by two parameters. Here $\mathbf{H}$ denotes hyperparameters, while Θ represents model parameters. In AutoML, selection of the most appropriate f^i and identification of suitable $\mathbf{H}$ is the main objective, while Θ is delegated to the training process. The scope of $\mathbf{H}$ varies, as it is system-dependent.

In the present investigation, the following terminology is adopted to ensure clarity in problem definition:

- **Primitive and Pipeline:** Here, primitive denotes an existing machine learning model, such as a DecisionTree. However, it can also include traditional ensemble learning models, such as AdaBoost and Bagging. The pipeline is the output of Autostacker, which is a single primitive or a combination of primitives.
- **Layer and Node:** Figure 1 depicts the Autostacker architecture, comprising of multiple stacking layers and multiple nodes in each layer. Each node represents a machine learning primitive model.

3.2 System Architecture

The working process of Autostacker is shown in Figure 2 and a sample pipeline built by Autostacker is provided in Figure 1. As noted previously, each pipeline consists of multiple layers, which in turn comprise of multiple nodes, denoting primitive machine learning models. The i th layer takes in the dataset X_i , and outputs the prediction result $Y_{i,j}$, where $Y_{i,j}$ denotes the prediction result of the j th node in the i th layer ($i = 0, 1, 2, \dots, I, j = 0, 1, 2, \dots, J$). Once each layer generates a prediction, these prediction results are added as synthetic features to the dataset. This newly generated dataset serves as the input for the next layer. In other words, the input of i th layer X_i is updated as follows:

$$X_i = X_{i-1} \cup Y_{i-1,0} \cup Y_{i-1,1} \dots \cup Y_{i-1,J'} \quad (2)$$

where J' is the number of nodes in $(i-1)$ th layer. With each new layer, the dataset is augmented by a new set of synthetic features, until the last layer—which consists of a single node—is reached. The output of the last layer serves as the final output of the machine learning problem.

Thus, if we use f_k to denote the k th ($k = 0, 1, 2, \dots, K$) feature in the dataset, the final dataset will contain

$$(K+1) + \sum_{i=0}^{I-1} (N_i + 1) \quad (3)$$

features and this new dataset will be used in the last layer prediction. Here, N_i ($0, 1, 2, \dots$) is the number of nodes in the i th layer. It should be noted that the total number of features in the dataset before the last layer can be specified by users.

Unlike the traditional stacking algorithm employed in ensemble learning, which provides the prediction results as the only inputs

into the next layer, in this proposed architecture, the information is always cascaded directly from the raw dataset. To the best knowledge of the authors, this is the first attempt to generalize this algorithm and incorporate it into the AutoML system. However, it should be noted that similar methods have been adopted in practice to solve specific machine learning problems. In the approach described here, the following aspects must be considered:

- If the dataset comprises of a small number of items, the prediction result from each layer could contain very little information about the problem. Consequently, it is very likely that the outcomes will be largely dependent on, and potentially significantly biased by, the primitives. Accordingly, the raw dataset must be incorporated into the process to avoid highly biased prediction results that are unsuitable for generalization. This is particularly important in situations where subsequent availability of additional training data is anticipated.
- Moreover, combining the new synthetic features with the raw dataset results in some features that are deemed important for prediction accuracy being implicitly assigned a greater weight. Still, unlike in regular stacking, the raw dataset is always retained, to avoid disproportional influence of the primitives in each individual layer on the final outcome. Owing to this strategy, the effects of bias stemming from individual primitives and noise arising from the raw dataset can be mitigated.

The hyperparameter space in Autostacker consists of the following four components:

$$H = \begin{cases} \text{type of each primitive} \\ \text{each model hyperparameter within each primitive} \\ \text{number of layers in each pipeline} \\ \text{number of nodes in each layer} \end{cases} \quad (4)$$

The following attributes of Autostacker can be configured by the user based on their computational resources and/or time constraints:

- I and J : These parameters respectively denote the maximum number of layers and the maximum number of nodes corresponding to each layer.
- The primitive types: In this work, a dictionary of primitives that serves solely as a search space is provided for brevity. However, additional primitives can be added by the user as needed.

It is also noteworthy that Autostacker provides two ways of specifying I and J . In the default mode, users simply specify their respective maximum ranges, which must be positive integers. This is not only sufficient for Autostacker to explore different configurations, but also yields the following benefits: (1) This mode does not impose any constraints on the system, thereby allowing the discovery of further potentially beneficial and innovative pipelines; (2) The search process is significantly expedited, as shown in the Experiment section.

Alternatively, users can explicitly denote the value of I and J . This allows systems to build pipelines with a specific number of

layers and number of nodes per layer based on the available computational power and time.

The search algorithm for finding the appropriate hyperparameters is described in the next section.

Algorithm 1 Basic EA Search

```

1:  $N = 200$ 
2:  $M = 10$ 
3:  $iter\_init = Random(N)$ 
4: for  $iter$  in  $M$  do
5:   Randomly sperate  $iter\_init$  into two equal parts, we get
6:    $iter\_init\_1$  and  $iter\_init\_2$ .
7:    $new\_gen\_1 = MUTATION(iter\_init\_1)$ 
8:    $new\_gen\_2 = CROSSOVER(iter\_init\_2)$ 
9:    $new\_gen = new\_gen\_1 + new\_gen\_2$ 
10:   $eva\_pip = iter\_init \cup new\_gen$ 
11:   $eva\_result = EVALUATE(eva\_pip)$ 
12:   $sel\_pip = SELECT(eva\_pip, eva\_result, N)$ 
13:   $iter\_init = sel\_pip$ 
14: end for
15: Return  $sel\_pip$ 
16: function  $MUTATION(list\_pip)$ 
17:   for each integer  $i$  in length of  $list\_pip$  do
18:      $list\_pip[i] = list\_pip[i]$  with one change in set (4)
19:   end for
20:   Return  $list\_pip$ 
21: end function
22: function  $CROSSOVER(list\_pip)$ 
23:   for each pair  $(pip\_1, pip\_2)$  in  $list\_pip$  do
24:     Randomly separate  $pip\_1$  into two parts.
25:     Randomly sperate  $pip\_2$  into two parts.
26:     Combine the 1st part of  $pip\_1$  with the 2nd part of  $pip\_2$ .
27:     Combine the 1st part of  $pip\_2$  with the 2nd part of  $pip\_1$ .
28:     Update  $pip\_1$  and  $pip\_2$  in  $list\_pip$ 
29:   end for
30:   Return  $list\_pip$ 
31: end function
32: function  $EVALUATE(list\_pip)$ 
33:   Train the  $list\_pip$ 
34:   for each integer  $i$  in length of  $list\_pip$  do
35:      $eva\_result[i] = CV(list\_pip[i])$ 
36:   end for
37:   Return  $eva\_result$ 
38: end function
39: function  $SELECT(eva\_pip, eva\_result, N)$ 
40:    $sel\_pip =$  the  $N$   $pips$  with highest  $eva\_result$ 
41:   Return  $sel\_pip$ 
42: end function

```

3.3 Search Algorithm

In this work, a basic evolutionary algorithm (EA) is chosen as the search algorithm for identifying the group of hyperparameters H that generate superior model pipelines. The EA adopted in the present study involves mutation and cross-over only, and does not incorporate any sophisticated techniques. As will be shown later,

even with this straightforward baseline algorithm, the proposed system can already achieve significantly better performance. The steps comprising the proposed algorithm are shown in Algorithm 1.

As can be seen from the flowchart, N completed pipelines are initially generated by randomly selecting the hyperparameters. Next, a one-step mutation is run on the upper half of these pipelines to obtain additional $N/2$ pipelines, whereby the candidates for mutation are chosen randomly. In the following step, further $N/2$ pipelines are selected to run the cross-over, resulting in N new pipelines in total.

The one-step mutation results in a random change in one of the hyperparameters in H as in set (4). This change could pertain to, for example, the number of estimators in a Random Forest Classifier, or result in replacing an SVM classifier with a logistic regression classifier. Cross-over, on the other hand, results in a partial exchange of topology between two pipelines. For example, the first half of the layers in one pipeline and the second half of the layers in another pipeline can be chosen to formalize a new pipeline.

Once the above steps are completed, the generated $2N$ pipelines are trained and evaluated through cross-validation. As a result, N pipelines with the highest validation accuracies are selected as the seed pipelines for the next generation of mutation and cross-over. Once the seed pipelines are completed, they are subjected to a further one-step mutation and cross-over, followed by evaluation and selection. This process is repeated until all the iterations are executed. It should be noted that the number of iterations M can be specified by the user.

3.4 Training and Testing Process

This section is designated for the training and testing procedure. In the proposed approach, training is performed in the evaluation step, as shown above. Corresponding to the hierarchical framework employed here, the pipeline is trained layer by layer. Within each layer, each primitive is also trained independently using the same dataset. The next layer is trained on an expanded dataset, obtained by concatenating the previous dataset with the prediction results from the previously trained layer. Similarly, the validation and testing processes share the same mechanism with validation and test sets, respectively.

Once all pipelines are trained and validated, the first ten pipelines with the highest validation accuracies are selected as the final Autostacker output. This choice is based on the premise that these ten pipelines can provide better baselines for human experts when aiming to solve a specific problem. Providing the user with a range of modeling options, rather than a single best-performing pipeline, allows for greater flexibility in the modeling process. In addition, the effect of small, unbalanced datasets is considered, given that, in such cases, it is difficult to guarantee that the validation process performance can fully represent that obtained on the test set. For example, two pipelines with the same validation results might behave very differently when applied to the same test dataset. Hence, it is necessary to provide a set of candidates that can be guaranteed to perform adequately, on average, allowing human experts to refine the pipelines output by Autostacker.

3.5 Scaling and Parallelization

Another significant advantage of the proposed approach stems from the system flexibility, allowing it to scale up and parallelize, as these aspects are inherent in EA. Throughout the sequence of steps, comprising of the initial generation, one-step mutation, one-step cross-over, training, validation, and finally evaluation, each pipeline runs independently, allowing individual pipelines to be refined independently of others.

Moreover, the proposed approach avoids the need for frequent communication or sequential decision making among processes applied to individual pipelines. The only requirements imposed on these processes is that validation results pertaining to individual pipelines be shared, in order to be ranked at the end of each iteration. A single selection, based on the validation accuracy, is subsequently applied on the outputs of the parallel processes described above. More specifically, in terms of the Algorithm 1 described above, `Random()`, `MUTATION()`, `CROSSOVER()`, and `EVALUATE()` functions are easily parallelized at runtime.

4 EXPERIMENTS

4.1 Dataset and Preprocessing

To demonstrate the performance of the system described in the preceding sections, 15 datasets were selected from the benchmark dataset [20]—comprising of public data resources, such as OpenML [32] and UCI [16], among others—and served as the sample experimental data for testing Autostacker. Specifically, nine datasets claimed to produce better results in TPOT when compared to Random Forest Classifier were chosen arbitrarily, along with four datasets that were noted for their inferior performance in TPOT, and two datasets that yielded the same performance when subjected to Random Forest Classifier in TPOT. Although the experimental sample comprised of only 15 datasets, they were representative of all cases used in TPOT. Moreover, these datasets pertain to different problem domains and target different machine learning tasks, including binary classification and multi-class classification.

In its current design, Autostacker does not require any data or feature preprocessing. However, the system is sufficiently flexible to permit these tasks, which can be incorporated as another building block or a hyperparameter in Autostacker. Nevertheless, as the main objective of the present work is to demonstrate the modeling process as the key contribution to the architecture and automation process of the proposed framework, these additional features are not examined in the subsequent sections. It is also worth noting that, prior to commencing each set of experiments, the dataset was shuffled and partitioned into training (80%) and testing (20%) data. The first 80% of the dataset was used in the cross-validation conducted during the selection stage.

4.2 Baseline Comparison

The goal of Autostacker is to automatically provide a better baseline pipeline for data scientists. Thus, the baseline used for a comparison should represent the prediction ability of pipelines resulting from the initial trials performed by data scientists. For these reasons, Random Forest Classifier was chosen as the baseline pipeline, as it includes 500 estimators as ensemble learning models, which have

Figure 3: Test Accuracy and Time Cost Comparison. The first two rows show the test accuracy computed with balanced accuracy among Autostacker and other compared frameworks. Results on 15 benchmark datasets have been shown here. The third row shows the time cost comparison between TPOT and Autostacker.

Figure 4: Autostacker outperforms two other architectures on average. The y-axis shows the number of datasets that each architecture outperforms all the other architectures. The measurement is based on the average test accuracy.

been shown to work well on average in the context of multi-model predictions. The results yielded by Autostacker are further compared to those produced by the TPOT [21] model, one of the more recent and highly popular AutoML systems, as well as AutoSklearn [7], which won 1st place in the final phase of 2016 AutoML challenge. As authors of both TPOT and AutoSklearn have provided their systems as an open-source, many subsequent improvements have subsequently been made by other researchers and practitioners in the AutoML community. In this experiment, TPOT v0.9.0 and AutoSklearn v0.3.0 were utilized.

Table 1: Primitive List in Autostacker

Perceptron	AdaBoostClassifier
LogisticRegression	XGBClassifier
SVC	MLPClassifier
DecisionTreeClassifier	BernoulliNB
KNeighborsClassifier	MultinomialNB
RandomForestClassifier	GradientBoostingClassifier
BaggingClassifier	ExtraTreesClassifier

Currently, the primitives used in Autostacker are sourced from the scikit-learn [23] and XGBoost [4] libraries, as shown in Table 1. In Autostacker, users are allowed to plug in any primitives for which the function signatures are consistent with the current Autostacker code base. In terms of the basic structure (number of layers and number of nodes per layer) of the candidate pipelines, as previously noted, two types of settings are provided in Autostacker. In this section, the performance yielded by Autostacker in the default mode is discussed, which pertains to dynamic configurations. In addition, 5 is set as the maximum number of layers, while no more than 3 nodes per layer are permitted.

4.3 Results

In this section, the test accuracy and time cost results are presented. In addition, Autostacker performance is compared with that of the

Random Forest, TPOT and AutoSklearn. The test accuracy is calculated using balanced accuracy [33]. For each of the presented frameworks, the reported results pertain to 20 to 30 pipelines generated by running multiple experiments with same 80/20 splitting strategy indicated above. In terms of running time, Autostacker was run for 3 full generations, while 100 generations were used for TPOT, as pointed by the original paper. For AutoSklearn, the time limitation for each round was set to one hour. The points in these plots represent outlier values, calculated using Tukey’s fence method [31]. When outliers are shown on the top or bottom, the notches represent the upper or lower fences respectively. If no points are shown on one end, the notch no longer represents a fence but rather an extreme. The experiments were run using 24 CPU machines with 40GB of memory.

The first two rows in Figure 3 show test accuracy comparisons among the examined frameworks when applied to the previously described 15 sample datasets. Several key observations can be made based on the results:

- Autostacker achieves better test accuracy when compared with Random Forest Baselines in all 15 cases, while its accuracy is comparable or superior to that of TPOT and AutoSklearn in **12 out of 15** datasets.
- Autostacker is robust, as it provides a good baseline when applied on all 15 datasets. On the other hand, Random Forest fails to produce meaningful results on the parity5 and parity5+5 datasets, whereas TPOT fails to provide better baselines than Random Forest Classifier on the pima, ecoli and wine-recognition datasets after multiple hours of computation. AutoSklearn also fails to outperform any model on heart-h and wine-recognition.

The third row in Figure 3 shows the time cost of TPOT and Autostacker. AutoSklearn is excluded from these comparisons because it is mandatory to specify the time limitation a priori when using AutoSklearn; otherwise 1 hour is set as a default time limitation. This default setting was adopted in all experiments performed in this work, as noted earlier. As can be seen from the findings, Autostacker significantly reduces the time usage (up to **6 times**) compared to TPOT. Autostacker also requires less time than AutoSklearn when applied to 11 datasets. Interestingly, AutoSklearn outperforms Autostacker in both time and accuracy on three datasets (Hill Valley, allhypo, and vehicle). While these findings could suggest that AutoSklearn performs better than other frameworks when applied on larger datasets, it should be noted that Autostacker had the highest test accuracy on the largest dataset (dis, 3772 samples). Moreover, Autostacker seems to have an inherent advantage when applied to smaller datasets.

The experimental results are summarized in Figure 4, where it can be seen that Autostacker output improves the baseline pipeline sufficiently to benefit human experts in their subsequent work. Moreover, Autostacker outperforms all the baseline systems on average on all the sample datasets.

5 DISCUSSION

Despite great performance on the fifteen dataset benchmark, Autostacker still has several limitations, as noted below, along with possible solutions that could be implemented in the future:

- Many modern approaches and architectures achieve excellent results on large, high-dimensional datasets and multi-task problems. Deep learning, for example, has become the dominant approach in a wide range of fields, such as computer vision and natural language processing [15]. The primitive library and modeling structure currently adopted in Autostacker do not scale well to these problems. One direction for future work in this field could thus be to incorporate more advanced primitives into Autostacker’s catalog and use them as necessary.
- Autostacker efficiency can be improved by adopting better search algorithms. Many variants of evolutionary algorithms expand on the basic version used in this work. Experimenting with different algorithms may result in Autostacker performing the search faster or finding better pipelines. A rigorous statistical analysis is also likely to yield a better understanding of the Autostacker output. In particular, it would be beneficial to elucidate why certain architectures are chosen, or how those architectures evolve over time.

6 CONCLUSION

In this work, Autostacker was introduced. The architecture of this AutoML system was inspired by stacking, cascading, and evolutionary algorithms. Despite the lack of data preprocessing and feature selection phases, Autostacker still outperforms competing AutoML systems on a wide variety of datasets in both accuracy and speed. It is the authors’ hope that this proposed framework will serve as a new benchmark in AutoML, which bears the potential for incorporating additional primitives and preprocessing techniques.

ACKNOWLEDGMENTS

The authors would like to thank Chad DeChant for helping with proofreading and useful discussions.

This research was supported in part by the US Defense Advanced Research Project Agency (DARPA) *Data-Driven Discovery of Models (D3M)* Program, grant FA8750-17-2-0124.

REFERENCES

- [1] James Bergstra, Dan Yamins, and David D Cox. 2013. Hyperopt: A python library for optimizing the hyperparameters of machine learning algorithms. In *Proceedings of the 12th Python in Science Conference*. 13–20.
- [2] Leo Breiman. 1996. Stacked regressions. *Machine learning* 24, 1 (1996), 49–64.
- [3] Rich Caruana, Alexandru Niculescu-Mizil, Geoff Crew, and Alex Ksikes. 2004. Ensemble selection from libraries of models. In *Proceedings of the twenty-first international conference on Machine learning*. ACM, 18.
- [4] Tianqi Chen and Carlos Guestrin. 2016. XGBoost: A Scalable Tree Boosting System. *CoRR abs/1603.02754* (2016). arXiv:1603.02754 <http://arxiv.org/abs/1603.02754>
- [5] Corinna Cortes and Vladimir Vapnik. 1995. Support-vector networks. *Machine learning* 20, 3 (1995), 273–297.
- [6] Agoston E Eiben, James E Smith, et al. 2003. *Introduction to evolutionary computing*. Vol. 53. Springer.
- [7] Matthias Feurer, Aaron Klein, Katharina Eggensperger, Jost Springenberg, Manuel Blum, and Frank Hutter. 2015. Efficient and robust automated machine learning. In *Advances in Neural Information Processing Systems*. 2962–2970.
- [8] Yoav Freund. 1995. Boosting a weak learning algorithm by majority. *Information and computation* 121, 2 (1995), 256–285.
- [9] H2O.ai. [n. d.]. H2O-3. ([n. d.]). <https://github.com/h2oai/h2o-3>
- [10] Mark Hall, Eibe Frank, Geoffrey Holmes, Bernhard Pfahringer, Peter Reutemann, and Ian H Witten. 2009. The WEKA data mining software: an update. *ACM SIGKDD explorations newsletter* 11, 1 (2009), 10–18.
- [11] Jeremy Heitz, Stephen Gould, Ashutosh Saxena, and Daphne Koller. 2009. Cascaded classification models: Combining models for holistic scene understanding. In *Advances in Neural Information Processing Systems*. 641–648.
- [12] Frank Hutter, Holger H Hoos, and Kevin Leyton-Brown. 2011. Sequential Model-Based Optimization for General Algorithm Configuration. *LION* 5 (2011), 507–523.
- [13] Michael Kearns. 1988. Learning Boolean formulae or finite automata is as hard as factoring. *Technical Report TR-14-88 Harvard University Aikem Computation Laboratory* (1988).
- [14] Lars Kotthoff, Chris Thornton, Holger H Hoos, Frank Hutter, and Kevin Leyton-Brown. 2016. Auto-WEKA 2.0: Automatic model selection and hyperparameter optimization in WEKA. *Journal of Machine Learning Research* 17 (2016), 1–5.
- [15] Yann LeCun, Yoshua Bengio, and Geoffrey Hinton. 2015. Deep learning. *nature* 521, 7553 (2015), 436.
- [16] M. Lichman. 2013. UCI Machine Learning Repository. (2013). <http://archive.ics.uci.edu/ml>
- [17] James Lloyd. 2015. Freeze Thaw Ensemble Construction. (2015). <https://github.com/jamesrobertlloyd/automl-phase-2>
- [18] Gregory Morse and Kenneth O. Stanley. 2016. Simple Evolutionary Optimization Can Rival Stochastic Gradient Descent in Neural Networks. In *Proceedings of the Genetic and Evolutionary Computation Conference 2016 (GECCO '16)*. ACM, New York, NY, USA, 477–484. <https://doi.org/10.1145/2908812.2908916>
- [19] Randal S. Olson, Nathan Bartley, Ryan J. Urbanowicz, and Jason H. Moore. 2016. Evaluation of a Tree-based Pipeline Optimization Tool for Automating Data Science. In *Proceedings of the Genetic and Evolutionary Computation Conference 2016 (GECCO '16)*. ACM, New York, NY, USA, 485–492. <https://doi.org/10.1145/2908812.2908918>
- [20] Randal S Olson, William La Cava, Patryk Orzechowski, Ryan J Urbanowicz, and Jason H Moore. 2017. PMLB: A Large Benchmark Suite for Machine Learning Evaluation and Comparison. *arXiv preprint arXiv:1703.00512* (2017).
- [21] Randal S Olson, Ryan J Urbanowicz, Peter C Andrews, Nicole A Lavender, Jason H Moore, et al. 2016. Automating biomedical data science through tree-based pipeline optimization. In *European Conference on the Applications of Evolutionary Computation*. Springer, 123–137.
- [22] David W Opitz and Richard Maclin. 1999. Popular ensemble methods: An empirical study. *J. Artif. Intell. Res.(JAIR)* 11 (1999), 169–198.
- [23] Fabian Pedregosa, Gaël Varoquaux, Alexandre Gramfort, Vincent Michel, Bertrand Thirion, Olivier Grisel, Mathieu Blondel, Peter Prettenhofer, Ron Weiss, Vincent Dubourg, et al. 2011. Scikit-learn: Machine learning in Python. *Journal of Machine Learning Research* 12, Oct (2011), 2825–2830.
- [24] F. Petroski Such, V. Madhavan, E. Conti, J. Lehman, K. O. Stanley, and J. Clune. 2017. Deep Neuroevolution: Genetic Algorithms Are a Competitive Alternative for Training Deep Neural Networks for Reinforcement Learning. *ArXiv e-prints* (Dec. 2017). arXiv:1712.06567
- [25] Robi Polikar. 2006. Ensemble based systems in decision making. *IEEE Circuits and systems magazine* 6, 3 (2006), 21–45.
- [26] Lior Rokach. 2010. Ensemble-based classifiers. *Artificial Intelligence Review* 33, 1-2 (2010), 1–39.
- [27] Robert E Schapire. 1990. The strength of weak learnability. *Machine learning* 5, 2 (1990), 197–227.
- [28] Jasper Snoek, Hugo Larochelle, and Ryan P Adams. 2012. Practical bayesian optimization of machine learning algorithms. In *Advances in neural information processing systems*. 2951–2959.
- [29] Jost Tobias Springenberg, Aaron Klein, Stefan Falkner, and Frank Hutter. 2016. Bayesian optimization with robust Bayesian neural networks. In *Advances in Neural Information Processing Systems*. 4134–4142.
- [30] Chris Thornton, Frank Hutter, Holger H Hoos, and Kevin Leyton-Brown. 2013. 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*. ACM, 847–855.
- [31] J.W. Tukey. 1977. *Exploratory Data Analysis*. Addison-Wesley Publishing Company. <https://books.google.com/books?id=UT9dAAAAIAAJ>
- [32] Joaquin Vanschoren, Jan N. van Rijn, Bernd Bischl, and Luis Torgo. 2013. OpenML: Networked Science in Machine Learning. *SIGKDD Explorations* 15, 2 (2013), 49–60. <https://doi.org/10.1145/2641190.2641198>
- [33] Digna R Velez, Bill C White, Alison A Motsinger, William S Bush, Marylyn D Ritchie, Scott M Williams, and Jason H Moore. 2007. A balanced accuracy function for epistasis modeling in imbalanced datasets using multifactor dimensionality reduction. *Genetic epidemiology* 31, 4 (2007), 306–315.
- [34] Paul Viola and Michael Jones. 2001. Rapid object detection using a boosted cascade of simple features. In *Computer Vision and Pattern Recognition, 2001. CVPR 2001. Proceedings of the 2001 IEEE Computer Society Conference on*, Vol. 1. IEEE, I–I.
- [35] David H Wolpert. 1992. Stacked generalization. *Neural networks* 5, 2 (1992), 241–259.