P4ML: A Phased Performance-Based Pipeline Planner for Automated Machine Learning

Yolanda Gil, Ke-Thia Yao, Varun Ratnakar, Daniel Garijo, Greg Ver Steeg, Pedro Szekely, Rob Brekelmans, Mayank Kejriwal, Fanghao Luo and **I-Hui Huang**
[gil, kyao, varunr, dgarijo,gregv, pszekely, brekelma, kejriwal]@isi.edu
Information Sciences Institute, University of Southern California

Abstract

While many problems could benefit from recent advances in machine learning, significant time and expertise are required to design customized solutions to each problem. Prior attempts to automate machine learning have focused on generating multi-step solutions composed of primitive steps for feature engineering and modeling, but using already clean and featurized data and carefully curated primitives. However, cleaning and featurization are often the most time-consuming steps in a data science pipeline. We present a novel approach that works with naturally occurring data of any size and type, and with diverse third-party data processing and modeling primitives that can lead to better quality solutions. The key idea is to generate multi-step pipelines (or workflows) by factoring the search for solutions into phases that apply a different expert-like strategy designed to improve performance. This approach is implemented in the P4ML system, and demonstrates superior performance over other systems on a variety of raw datasets.

Keywords: Automating machine learning, planning, pipelines, workflows, AutoML

1. Introduction

Machine learning applications require significant expertise, tuning and effort. Research on automating machine learning (AutoML) focuses on developing approaches to automatically generate models for a given dataset, including any necessary featurization and data preparation steps. Early work in this area explored the use of artificial intelligence planning to generate multi-step pipelines (i.e., workflows) composed of data pre-processing and modeling steps (St. Amant and Cohen, 1998; Hauder et al., 2011). Given the ubiquity of data and the great interest in exploiting it, AutoML has been receiving increased attention. Auto-sklearn uses Bayesian optimization methods, and placed first in the ChaLearn AutoML challenge (Feurer et al., 2015a). Fusi et al. (2017) augmented that approach with probabilistic matrix factorization. A very different approach was used in TPOT, which relies on genetic algorithms to explore combinations of steps that lead to better performance (Olson et al., 2016b). However, there are still many open research topics in AutoML. Existing approaches focus on feature engineering and modeling, and assume that the data is already clean and that numerical features have already been generated. They use a carefully selected and well-curated set of pre-processing and modeling steps (or *primitives*) in order to make the search manageable. They also focus on classification tasks. Our goal is to design AutoML approaches that can accommodate any type of naturally occurring data, any collection of primitives, and any size of data. This paper presents a novel approach to AutoML that supports these goals and that has three key contributions:

1. Exploits expert strategies to structure the search for solutions in a machine learning problem into meaningful phases,

2. Uses knowledge about both datasets and primitives in order to design end-to-end pipelines that include data cleaning and featurization steps,

3. Explores the search space efficiently and returns the best solution found within a given time limit.

The paper begins articulating our goals and requirements, followed by an overview of our approach. We then describe the implementation of our approach in P4ML, a phased performance-based pipeline planner for automating machine learning. P4ML was populated with dozens of diverse third-party primitives, and the evaluations so far demonstrate superior performance on a variety of naturally occurring datasets.

2. Goals and Requirements

Our goal is to automate machine learning with approaches that will handle naturally occurring datasets, which leads us to several important requirements not addressed in prior research. First, we need to be able to generate solutions for any type of dataset. Our objective is to handle a wide variety of data such as images, audio, text, etc. There may be many ways to featurize these datasets, and the featurization approach matters for the quality of the solution. Second, we need to assume that the datasets are not necessarily clean. Raw data is typically full of errors, missing values, and other imperfections that make it of a less than acceptable quality to get reasonable results from models. Therefore, data cleaning steps should be part of the automatically generated pipeline. Third, we want to be able to incorporate a large number of diverse third-party modeling and other data processing primitives. A small number of primitives may be sufficient to generate some solution, but each algorithm works better for some datasets. Therefore, having a wide range and a large number of primitives is desirable. Our goal is to be able to incorporate into the pipeline generation process a large and diverse set of third party primitives and use them to generate the best solutions. Fourth, we need to be able to handle very large datasets within time constraints. At any time, complete pipelines should be available, so the best one can be returned as the answer.

3. P4ML: A Phased Performance-Based Pipeline Planner

There are three key aspects of our approach:

1. Exploit expert-like strategies to factor the search space. We use a hierarchical metadata-based planner that searches for solutions while being mindful of performance. The planner is given a time limit and outputs the best solution generated within that time.

2. Automatically annotate a catalog of primitive data processing and modeling steps. The annotations provide rich metadata about preconditions, performance, and other constraints of each primitive that are used during search.

3. Dynamically characterize, clean, and featurize datasets.

We first describe the pipeline generation process that uses metadata about primitives and datasets, and then discuss how that metadata is obtained or created.

3.1. Pipeline Generation

The pipeline generation process starts with a *task* (e.g., classification, regression, etc.), a *metric* (e.g., f1 macro, mean squared error, etc.), a *dataset* and a *time limit*. A catalog of primitives for modeling and data processing is used, and is described in Section 3.2. We divide the search into five distinct phases:

1. **Phase 1: Dataset characterization and featurization.** Tabular datasets are characterized in terms of the types of features (enumerated values, strings, numerical, etc.) and their ranges. Cleaning primitives are added to improve the overall quality of the dataset. Datasets that contain image, sound and video need to be featurized in order to extract relevant features that can be used by modeling primitives. Featurization primitives are selected depending on the type of data, and if several are available then multiple candidate pipelines are generated and passed on to the next phase.

2. **Phase 2: Pipeline skeleton design.** In this phase, we identify the modeling primitives in the catalog that are suitable for addressing the given task based on annotations of their functionality. This results in pipeline candidates for phase 3. Pipeline candidates are prioritized according to the diversity of the algorithms, i.e., if a Naive Bayes classifier has already been explored as a solution, we will prioritize other type of classifiers (e.g., based on random forest or decision trees).

3. **Phase 3: Requirement satisfaction.** Each primitive may have different requirements that need to be addressed prior to its execution. For example, a classifier may only work on datasets that have no missing values. Other primitives may only work on numerical data (i.e., not on categorical features that are strings), or on negative values. Given a candidate pipeline from Phase 2, Phase 3 analyzes the requirements of the pipeline primitives and adds additional primitives to create an executable pipeline. These requirements are addressed by profiling the dataset being analyzed, and adding primitives to address requirements when necessary. For any Phase 2 pipeline candidate, several Phase 3 pipelines may be generated, as there could be different strategies for addressing the requirements of a primitive. For example, different imputation primitives may be available to address missing values (e.g., fill in the most common value of a column, average values, etc.), each leading to different pipeline performance. While Phase 2 pipelines are just skeletons and are not executable, Phase 3 candidate pipelines can be run. A cross validation is performed to rank the candidate pipelines in a global table according to their performance in the given metric.

4. **Phase 4: Hyperparameter search.** Phase 3 pipelines provide the basic structure of the pipeline solutions. However, these pipelines use default hyperparameters for each of their primitives. In Phase 4 we select the highest ranked Phase 3 pipelines and perform a random search on their hyperparameters. The results are stored in the the global table as well.

5. **Phase 5: Ensemble generation.** An ensemble is created with the top performing pipelines. We carry out a greedy search over ensembles, beginning with the best performing pipeline and adding pipelines (with replacement) while the cross validation score is improving. Ensemble predictions are calculated and evaluated using a majority vote for classification or mean for regression.

In order to be able to explore a maximum number of solutions within the given time limit, phases 2 to 4 occur in parallel. When the given time limit is approaching, the best solution available in the global rank will be selected as the final output.

3.2. Primitive Catalog

We assume a catalog of primitives that contains metadata to describe their invocation and functionality. We clustered all primitives using bi-level rules that took into account both the 'primitive family' (e.g., Classification) and 'algorithm type' (e.g., KNN) annotations that accompany each primitive release. This rule-based clustering was found to closely mirror pre-built algorithmic hierarchies like scikit-klearn (Pedregosa et al., 2011), and we used it to build a primitive taxonomy to organize our search. We also performed an automated analysis on the primitive catalog to automatically annotate requirements of a primitive (e.g. inputs must have no missing values) by using a profiler[1]. These requirements are used in Phase 3 of our planner.

3.3. Characterizing, Cleaning, and Featurizing Datasets

Data profiling and cleaning are integral components in many real-world machine learning pipelines. We designed a data profiling primitive that accepts a tabular dataset as input and computes both basic profiles, examples being column data types (e.g., String, Date, Number) and the proportions of missing values in each column, but also more advanced profiles such as the language in text columns. The constructed profiles are used in Phase 2 when the system searches the pipeline skeleton design space.

We have also designed a suite of data cleaning primitives, the most important of which is *missing value imputation* (MVI). Our MVI primitive encapsulates various algorithmic options for imputing missing values, including simple algorithms (e.g., using column mean values), and advanced matrix-based algorithms. By default, a greedy search algorithm is used to automatically configure the primitive and decide which MVI algorithm to impute for each column containing missing values.

3.4. Implementation

We have implemented this approach in P4ML, an AutoML system that is organized in a modular architecture. P4ML is an initial prototype, and the pipeline generation phases can be easily extended as our research moves forward. As more metadata about primitives is added, the system can use that information to generate and prioritize candidates.

We build on a primitive catalog provided by third parties that participate in the DARPA Data Driven Discovery of Models (D3M) program. The version of the catalog used for this

[1] https://github.com/usc-isi-i2/dsbox-ta2/tree/master/python/dsbox/profiler/primitive

work includes 127 primitives that contain basic metadata describing their main functionality. Among these 127 primitives, 43 are scikit-learn primitives (classifiers, regressors and data processing primitives) (Pedregosa et al., 2011). We collaborated with D3M program participants to develop appropriate APIs for the primitives, as well as for ingestion of datasets and for testing and evaluation. Datasets are read from files, and Pandas DataFrames are generated to represent attribute matrices and target vectors. A development version of for our prototype is available in Github;[2] while a snapshot of the code with the version used in this paper is available in Zenodo (Yao et al., 2018).

4. Related Work

Recent development in AutoML has focused on the algorithm selection and hyperparameter optimization problems using Bayesian optimization and genetic programming approaches.

Auto-sklearn (Feurer et al., 2015a), the overall winner of the ChaLearn AutoML challenge (Guyon et al., 2016), extends the Bayesian optimization approach of the Auto-WEKA system. (Thornton et al., 2013) to a Python environment. In particular, Sequential Model-Based Algorithm Configuration (Hutter et al., 2011) (SMAC), is used not only for hyperparameter optimization, but also for algorithm selection by conditionally initializing relevant parameters. Importantly, (Feurer et al., 2015a) find marked improvement from using ensembles of models (Caruana et al., 2004) and warm-starting hyperparameter settings based on dataset meta-features (Feurer et al., 2015b), fitting within the wider tradition of meta-learning (Brazdil et al., 2008; Lemke et al., 2015). Meta-learning is also useful in finding a *surrogate model*, used by SMAC to predict performance for a given algorithm configuration and guide exploration toward promising models. While Feurer et al. (2015a) use random forests as a surrogate, recent advances have been achieved by using matrix factorization (Fusi et al., 2017) or scaling Gaussian process surrogates to large collections of metadata (Wistuba et al., 2018).

Tree-based genetic programming provides an alternative viable solution to the AutoML problem. Most notably, the TPOT system (Olson et al., 2016b,a) can construct arbitrarily long sequences of feature construction, feature selection, and classification operations via insertion, deletion, and sampling mutations.

It is important to note that these systems often draw from a restricted space of possible models. For example, Auto-sklearn is limited to 15 classifiers, 14 feature selectors, and 4 data preprocessors, while TPOT uses only decision tree and random forest based methods and Fusi et al. (2017) fixes the space of possible pipelines prior to training. Adapting and scaling AutoML methods to work over a fully flexible space of preprocessing, featurization, and modeling primitives remains a primary goal of our system.

5. Evaluation

For this evaluation we use datasets provided by DARPA's D3M program, specifically the LL0 datasets.[3] We note that the program will be releasing these datasets and other evaluation harness in the near future. The LL0 datasets consists of 384 individual datasets.

[2] https://github.com/usc-isi-i2/dsbox-ta2

[3] https://gitlab.datadrivendiscovery.org/d3m/datasets

	Classification		Regression	
	Success	Failure	Success	Failure
Auto-sklearn	124	93	61	16
P4ML	217	0	76	1

Table 1: Total number of datasets successfully handled by Auto-sklearn and P4ML.

	Better performance			
	P4ML	Ties	Auto-sklearn	Total problems
Classification	61	8	55	124
Regression	22	1	38	61

Table 2: Number of datasets where P4ML and Auto-sklearn had the best metric scores.

The datasets include text, images, audio, tables, and nested tables. Of the 384 datasets 217 datasets are classification problems, and the remaining 77 datasets are regression problems. Of the 217 classification datasets 120 are binary classification problems and 97 are multivariate classification problems. The F1 macro metric is used to evaluate classification problems, and the mean squared error is used for regression problems. The number of instances in the datasets range from 152 to 1,025,001, with a median of 751 instances. The number of attributes range from 4 to 10,938 with a median of 15 attributes.

We compare P4ML against Auto-sklearn version 0.3.0 (Auto-sklearn, 2018) with respect to coverage, that is, the ability to process any datasets including those that are not necessarily clean. We use Pandas DataFrames as inputs for both systems. Table 1 summarizes the results of the runs. Auto-sklearn was able to successfully complete 124 of the 217 classification datasets and 61 of the 77 regression datasets. Most of the failures were due to text attributes and to categorical attributes. P4ML was able to successfully complete all 217 of the classification datasets and 76 of the 77 regression datasets. The only dataset that P4ML failed to process was due to a bug in the D3M code used to read the input.

We also compare the performance with respect to the metric given in each problem. Table 2 shows that the metric scores of our P4ML prototype are comparable to Auto-sklearn. Of the datasets that Auto-sklearn was able to process, P4ML performed better in 61 of the 124 classification datasets, and Auto-sklearn did better in 38 of the 61 regression datasets. The results of the evaluation are available for inspection in (Yao, 2018).

6. Conclusions

We present a novel approach to automated machine learning that works with naturally occurring data of any type and can generate multi-step pipelines using third-party data processing and modeling primitives. The key idea is to generate multi-step pipelines by factoring the search for solutions into phases that apply a different expert-like strategy designed to improve performance. This approach is implemented in the P4ML system, and has been evaluated with a broad range of datasets. Future work includes the incorporation of feature engineering primitives and deep learning approaches, learning and reusing pipeline fragments, meta-learning from other datasets to prioritize primitives and pipelines, and supporting interactive problem definition by domain experts.

Acknowledgments. We gratefully acknowledge support from the Defense Advanced Research Projects Agency under award FA8750-17-C-0106.

References

Auto-sklearn. Release 0.3.0, January 2018. URL https://github.com/automl/auto-sklearn/releases/tag/v.0.3.0.

Pavel Brazdil, Christophe Giraud Carrier, Carlos Soares, and Ricardo Vilalta. *Metalearning: Applications to data mining*. Springer Science & Business Media, 2008.

Rich Caruana, Alexandru Niculescu-Mizil, Geoff Crew, and Alex Ksikes. Ensemble selection from libraries of models. In *Proceedings of the twenty-first international conference on Machine learning*, page 18. ACM, 2004.

Matthias Feurer, Aaron Klein, Katharina Eggensperger, Jost Springenberg, Manuel Blum, and Frank Hutter. Efficient and robust automated machine learning. In C. Cortes, N. D. Lawrence, D. D. Lee, M. Sugiyama, and R. Garnett, editors, *Advances in Neural Information Processing Systems 28*, pages 2962–2970. Curran Associates, Inc., 2015a.

Matthias Feurer, Jost Tobias Springenberg, and Frank Hutter. Initializing bayesian hyperparameter optimization via meta-learning. In *AAAI*, pages 1128–1135, 2015b.

Nicolo Fusi, Rishit Sheth, and Huseyn Melih Elibol. Probabilistic matrix factorization for automated machine learning. *arXiv preprint:1705.05355*, 2017.

Isabelle Guyon, Imad Chaabane, Hugo Jair Escalante, Sergio Escalera, Damir Jajetic, James Robert Lloyd, Núria Macià, Bisakha Ray, Lukasz Romaszko, Michèle Sebag, et al. A brief review of the chalearn automl challenge: any-time any-dataset learning without human intervention. In *Workshop on Automatic Machine Learning*, pages 21–30, 2016.

Matheus Hauder, Yolanda Gil, and Yan Liu. A framework for efficient data analytics through automatic configuration and customization of scientific workflows. In *E-Science (e-Science), 2011 IEEE 7th International Conference on*, pages 379–386. IEEE, 2011.

F. Hutter, H. H. Hoos, and K. Leyton-Brown. Sequential model-based optimization for general algorithm configuration. In *Proc. of LION-5*, page 507523, 2011.

Christiane Lemke, Marcin Budka, and Bogdan Gabrys. Metalearning: a survey of trends and technologies. *Artificial intelligence review*, 44(1):117–130, 2015.

Randal S. Olson, Nathan Bartley, Ryan J. Urbanowicz, and Jason H. Moore. Evaluation of a tree-based pipeline optimization tool for automating data science. In *Proceedings of the Genetic and Evolutionary Computation Conference 2016*, GECCO '16, pages 485–492, 2016a. ISBN 978-1-4503-4206-3.

Randal S. Olson, Ryan J. Urbanowicz, Peter C. Andrews, Nicole A. Lavender, La Creis Kidd, and Jason H. Moore. *Applications of Evolutionary Computation: 19th European Conference, EvoApplications 2016, Porto, Portugal, March 30 – April 1, 2016,* *Proceedings, Part I*, chapter Automating Biomedical Data Science Through Tree-Based Pipeline Optimization, pages 123–137. Springer International Publishing, 2016b. doi: 10.1007/978-3-319-31204-0_9.

F. Pedregosa, G. Varoquaux, A. Gramfort, V. Michel, B. Thirion, O. Grisel, M. Blondel, P. Prettenhofer, R. Weiss, V. Dubourg, J. Vanderplas, A. Passos, D. Cournapeau, M. Brucher, M. Perrot, and E. Duchesnay. Scikit-learn: Machine learning in Python. *Journal of Machine Learning Research*, 12:2825–2830, 2011.

Robert St. Amant and Paul R Cohen. Intelligent support for exploratory data analysis. *Journal of Computational and Graphical Statistics*, 7(4):545–558, 1998.

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

Martin Wistuba, Nicolas Schilling, and Lars Schmidt-Thieme. Scalable gaussian process-based transfer surrogates for hyperparameter optimization. *Machine Learning*, pages 1–36, 2018.

Ke-Thia Yao. P4ML Evaluation Results, May 2018. URL https://doi.org/10.5281/zenodo.1251317.

Ke-Thia Yao, Varun Ratnakar, Rob Brekelma, Daniel Garijo, Fanghao Luo, I-hui Huang, and Pedro Szekely. P4ML source code release, May 2018. URL https://doi.org/10.5281/zenodo.1251320.