# ML-Plan for Unlimited-Length Machine Learning Pipelines

**Marcel Wever** MARCEL.WEVER@UPB.DE
**Felix Mohr** FELIX.MOHR@UPB.DE
**Eyke Hüllermeier** EYKE@UPB.DE
*Heinz Nixdorf Institute, Paderborn University, Paderborn, Germany*

## Abstract

In automated machine learning (AutoML), the process of engineering machine learning applications with respect to a specific problem is (partially) automated. Various AutoML tools have already been introduced to provide out-of-the-box machine learning functionality. More specifically, by selecting machine learning algorithms and optimizing their hyperparameters, these tools produce a machine learning pipeline tailored to the problem at hand. Except for TPOT, all of these tools restrict the maximum number of processing steps of such a pipeline. However, as TPOT follows an evolutionary approach, it suffers from performance issues when dealing with larger datasets. In this paper, we present an alternative approach leveraging a hierarchical planning to configure machine learning pipelines that are unlimited in length. We evaluate our approach and find its performance to be competitive with other AutoML tools, including TPOT.

**Keywords:** automated machine learning, complex pipelines, hierarchical planning

## 1. Introduction

The demand for machine learning functionality is increasing quite rapidly these days, not least because of recent impressive successes in practical applications. Since users in different application domains are normally not machine learning experts, a suitable support in terms of tools that are easy to use is required. Ideally, (nearly) the whole process including inducing models from data, data preprocessing, the choice of a model class, the training and evaluation of a prediction, etc. would be automated (Lloyd et al., 2014). This has triggered the field of *automated machine learning* (AutoML), which has developed into an important branch of machine learning research in the last couple of years.

Various state-of-the-art AutoML tools (Thornton et al., 2013; Komer et al., 2014; Feurer et al., 2015; Olson and Moore, 2016; de Sá et al., 2017) have shown impressive results in selecting machine learning algorithms and optimizing their hyperparameters to form a machine learning pipeline (ML pipeline). These approaches can be divided into two main categories. First, the AutoML problem is designed as an optimization problem with a fixed number of decision variables, which then is solved via standard (Bayesian) optimization tools such as SMAC (Hutter et al., 2011). Typically, these approaches, such as auto-sklearn and Auto-WEKA, have one variable for a pre-processing algorithm, one variable for the learning algorithm, and one variable for each parameter of each algorithm. However, a relaxation of the length restriction is not straight-forward. Approaches of the second category organize the AutoML search space in terms of a formal grammar. The advantage of this formalism is that it naturally allows for recursive structures, thereby supporting more flexible ML

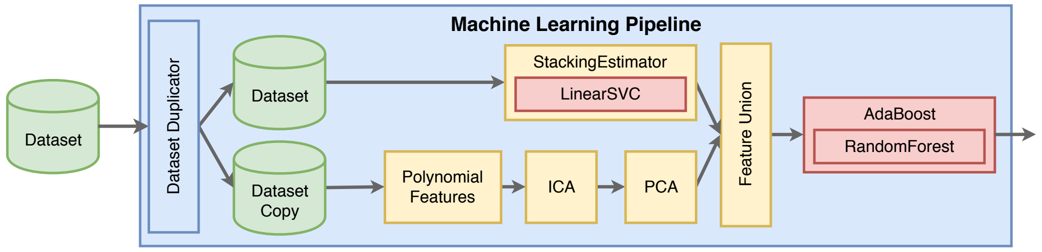

Figure 1: Possible workflow of an ML pipeline

pipelines. While de Sá et al. (2017) propose a grammar-based approach and allow for multiple preprocessing steps, the maximum length of an ML pipeline is still fixed. To the best of our knowledge, TPOT as originally introduced by Olson et al. (2016) is the only AutoML tool with no upper bound on the length of a pipeline. However, while the more flexible pipelines as composed by TPOT often perform particularly well, we also observed severe scalability issues of TPOT for more complex datasets (e.g., large number of features and/or instances).

In this paper, we present an alternative approach for composing ML pipelines that are unlimited in length, using a grammar-based formalism. More specifically, we show how ML-Plan (Mohr et al., 2018), an AutoML tool based on an AI planning technique called hierarchical task networks, can be extended for this purpose. In our evaluation, we find that our approach performs competitive to TPOT and furthermore improves on the scalability issues.

## 2. AutoML and Hierarchical Planning

AutoML seeks to automatically *compose* and *parametrize* machine learning algorithms into ML pipelines with the goal to optimize a given metric, e.g., predictive accuracy. Figure 1 shows an example of such a pipeline, which also illustrates that pipelines are by no means only sequences of atomic algorithms but can have parallel flows and nested structures as well. For example, StackingEstimator has a LinearSVC and AdaBoost uses RandomForest as a base learner. Especially, when it comes to meta methods, such recursive definitions of algorithms incorporating a base learner (and other components) constitute a very frequent pattern. In general, complete pipelines can be viewed as a hierarchical composition structure as in the example shown on the right-hand side of Figure 2. Furthermore, machine learning algorithms usually have hyperparameters that need to be chosen specifically for this algorithm. Thus, a hierarchical view of a machine learning pipeline represents its natural structure particularly well.

One interesting approach for creating such structure is hierarchical planning, a concept from the field of AI planning (Ghallab et al., 2004). In essence, it is about iteratively breaking down an initially given complex task into new sub-tasks, which may also be complex or simple (no further refinement required). Complex tasks are recursively decomposed until only simple tasks remain. This procedure is comparable, for example, to deriving a sentence from a context-free grammar. In that sense, complex tasks correspond to non-terminals and

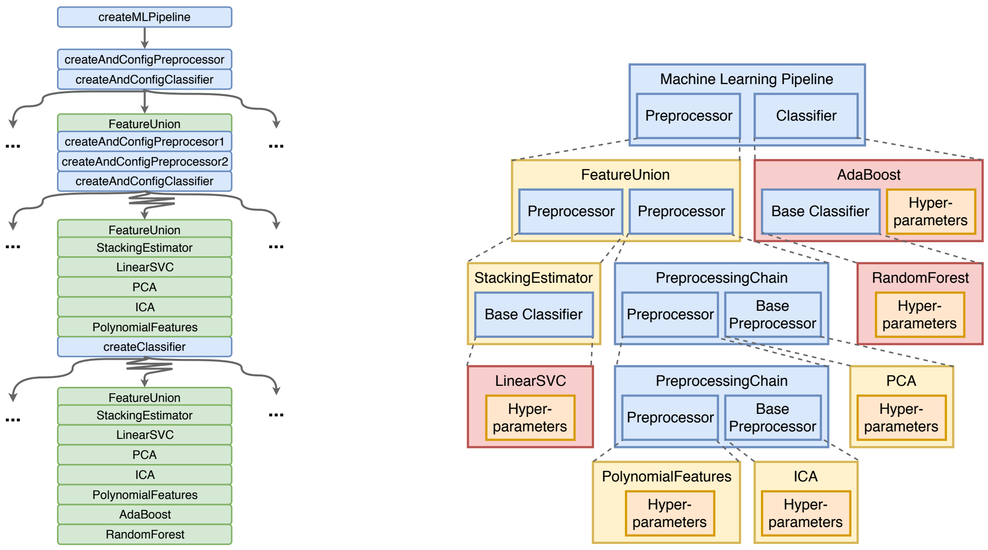

Figure 2: Excerpt of the search graph (left) listing the tasks of the planning problem at each node and its hierarchical representation in the form of an ML pipeline (right).

simple tasks match terminal symbols. An example is shown on the left-hand side in Figure 2 where complex tasks are displayed in blue and simple tasks in green.

We are aware of four approaches to AutoML using hierarchical planning or related techniques. The first approach is related to optimization within the RapidMiner framework based on hierarchical task networks (HTN) (Nguyen et al., 2014; Kietz et al., 2012). They conduct a beam search (hill-climbing in the most extreme case), where the beam is selected based on a *ranking* of alternative choices obtained from a meta-learning module, which compares the current dataset with previous ones and choices taken back then. The most recent representative of this line of research is Meta-Miner (Nguyen et al., 2014). While these approaches do not execute candidates during search to observe their performance, approaches of extensive evaluation is presented in RECIPE (de Sá et al., 2017) and TPOT (Olson and Moore, 2016). TPOT and RECIPE create pipelines using a grammar-based genetic programming algorithm; the pipeline candidates are evaluated in the course of computing their fitness. Last, ML-Plan (Mohr et al., 2018) recognizes the value of executing pipelines during search, but also observes that the extensive evaluation conducted in TPOT and RECIPE is infeasible for larger datasets. It reduces the number of evaluations by only considering candidates obtained from completions of currently best candidates. Like Meta-Learner, it is based on HTN planning.

Of course, other AutoML solutions such as Auto-WEKA or auto-sklearn can be extended to multiple pre-processing steps. It is clear that one can flatten any hierarchical structure into a vector as long as the allowed structures are bound in length. However, it is rather unclear how to represent ML pipelines that are unlimited in length.

In this paper, we extend ML-Plan to deal with unlimited-length ML pipelines, which is our approach for pipelines including a single preprocessor and a learner. In the following section, we give a brief overview of ML-Plan and explain how it is extended.

## 3. ML-Plan for Unlimited-Length ML Pipelines

### 3.1. ML-Plan

As briefly sketched above, ML-Plan is a hierarchical planner designed for AutoML problems (Mohr et al., 2018). Standard hierarchical planners such as SHOP2 (Au et al., 2011) lack some fundamental requirements of AutoML, e.g., to evaluate candidate solutions during search, which was a main motivation for developing ML-Plan.

The search technique adopted by ML-Plan is a best-first graph search. Like other planners, ML-Plan reduces the planning problem to the problem of finding a path to an optimal goal node in a graph. The graph is represented by a distinguished root node, a function for generating successors of a given node, and a function for testing whether a node is a goal node. In a nutshell, the best-first search algorithm assumes that every node in the explored part of the graph is associated with a score (in $\mathbb{R}$), and, in each iteration of the search, the leaf node with the lowest score is chosen for expansion. In contrast to A*, there is no assumption that the node score can be computed from edge costs; instead, there is just a function that returns the score without being related to the score of other nodes.

The node evaluation in ML-Plan is based on random path completion as also used in Monte Carlo Tree Search (Browne et al., 2012). To obtain the evaluation of a node, this strategy draws a fixed number of path completions, builds the corresponding pipelines and evaluates them against a validation set. The score assigned to the node is the *minimal* score that was observed over these validations in order to estimate the best solution that can be obtained when following paths under the node.

Intuitively, ML-Plan formalizes the HTN problem in a way that the resulting search graph is split into an algorithm selection region (upper region) and an algorithm configuration region (lower region). The main motivation for this strategy lies in the node evaluation we want to apply, which is based on random completions. Since algorithm selections usually constitute a much more significant change to the performance of a pipeline than parameter settings, we consider all solutions under a node that has all algorithms fixed as a kind of neighborhood, and random samples drawn in that sub-region yield more reliable estimates.

With the idea of a two-phased search graph in mind, the HTN definition of ML-Plan is as follows[1]. The initial task `createClassifier` can be broken down into a chain of the three tasks `createRawPP`, `setupClassifier`, `refinePP`. The first task is meant to choose the algorithms used for pre-processing *without* parametrizing them, the second task is meant to choose and configure the multi-label classifier, and the third step parametrizes the previously chosen pre-processors. The second task `setupClassifier` can, for each classifier, be decomposed into two sub-tasks. First, `<classifier>:create` is a simple task indicating the creation of a new classifier of the respective class, e.g. `RandomForestClassifier:create`. Second, `<classifier>:configure` is a complex task meant to configure the parameters of the classifier.

As an additional remark, ML-Plan comes with a built-in strategy to prevent over-fitting. This strategy apportions the assigned timeout for the whole search process among two phases. The first phase covers the actual search in the space. The second phase takes a collection of identified solutions and selects the one that minimizes the estimated general-

1. Since we have not formally introduced HTN planning, we describe the problem definition in a rather intuitive way. The formal definition can be found in the implementation published with this paper

ization error. Roughly speaking, the collection used for selection in phase 2 corresponds to the $k$ best candidates and $k$ random candidates that are not significantly worse than the best candidate. The time allocated at time step $t$ for the second phase is flexible and corresponds to the accumulated time that was required in phase 1 to evaluate the classifiers that would be chosen at time step $t$ for the selection process.

### 3.2. Extending ML-Plan for Unlimited-Length Pipelines

In order to compose ML pipelines with more complex pre-processing workflows, TPOT allows for chaining pre-processing algorithms and to fuse datasets taking the union of the respective features. Extending ML-Plan to operate on the same space of solution candidates, we add two complex and one simple tasks. First, we add a complex task `createFeatureUnion`, which may be resolved to the simple task `FeatureUnion` and two complex tasks to create the preceding pre-processing steps. Second, we add a complex task `createFeaturePreprocessorChainItem`, which may be resolved to one complex task for selecting a concrete pre-processing algorithm (e.g., PCA, Polynomial Features, etc.) and another complex task for creating any kind of pre-processing. In particular, the latter includes the building blocks for feature union and chaining pre-processing algorithms. We refer to this extension as ML-Plan-UL.

## 4. Experimental Evaluation

In our experimental evaluation, we focus on comparing ML-Plan-UL to TPOT, which both operate on the same solution space; besides, to the best of our knowledge, TPOT is the only AutoML tool supporting ML pipelines of unlimited length. As additional references, we also evaluate auto-sklearn and Auto-WEKA. All the tools are evaluated on a selection of 20 data sets from the `openml.org` (Vanschoren et al., 2013) repository, all of which were previously used to evaluate AutoML approaches (Thornton et al., 2013; Feurer et al., 2015).

The implementation of ML-Plan-UL, the evaluation code that produced the results shown in this section, and the used datasets are publicly available to assure reproducibility[2].

Results were obtained by carrying out 20 runs on each dataset with a timeout of one day per run. The timeout for the internal evaluation of a single solution was set to 20m for all the candidates. In each run, we used 70% of a stratified split of the data for the respective AutoML tool and 30% for testing. Note that we used the *same* splits for *all* tools. The computations were executed on 100 Linux machines in parallel, each of them equipped with 8 cores (Intel Xeon E5-2670, 2.6Ghz) and 32GB memory; every experiment used one machine at a time. The accumulated time of all experiments was over 300k CPU hours (over 34 CPU years).

Runs that did not adhere to the time or resource limitations (plus a tolerance threshold) were canceled without considering their results. That is, algorithms were canceled if they did not terminate within 110% of the predefined timeout or if they consumed more resources (memory or CPU) than allowed.

2. `https://github.com/fmohr/ML-Plan`

| Dataset | abalone | amazon | car | cifar10small | cifar10 | convex | credit-g | dexter | dorothea | gisette |
|---|---|---|---|---|---|---|---|---|---|---|
| ML-Plan-UL | **73.11** | **18.28** | 0.78 | 57.06 | - | 14.17 | 24.13 | **5.56** | **5.85** | 2.23 |
| TPOT | 73.35 | - | 0.40 | - | - | - | **23.53** | - | - | - |
| auto-sklearn | - | 22.0 ● | 1.64 ● | **55.08** ○ | - | **12.16** ○ | - | 7.30 ● | 6.04 | **2.22** |
| Auto-WEKA | 73.46 | 50.28 ● | **0.24** ○ | 62.09 ● | **64.06** | 45.7 ● | 26.44 ● | 9.82 ● | 11.01 ● | 4.18 ● |

| Dataset | krvskp | madelon | mnistrot | mnist | secom | semeion | shuttle | waveform | winequality | yeast |
|---|---|---|---|---|---|---|---|---|---|---|
| ML-Plan-UL | **0.65** | 15.55 | 49.5 | 3.36 | 6.71 | **5.51** | 0.02 | 13.61 | 33.17 | 38.94 |
| TPOT | 1.08 | 15.26 | - | - | **6.50** | 6.06 | **0.01** | 13.1 | **32.66** | **38.75** |
| auto-sklearn | 0.74 | **14.7** | **43.49** ○ | **2.90** | 6.57 | 6.29 | 0.02 | 13.6 | 36.25 ● | 39.27 |
| Auto-WEKA | 4.02 ● | 20.34 ● | 74.3 ● | 5.39 ● | 6.60 | 8.42 ● | 0.13 ● | **13.05** | 33.58 | 39.8 |

Table 1: Mean 0/1-losses [in %] for a timeout of one day

As TPOT did not return even a single result for any of the datasets within the timeout, we configured TPOT to log intermediate solutions and considered the most recent one of these to compute the respective value in the results table.

The results of the experiments are summarized in Table 1, with best performances highlighted in bold. To determine significance for differences in performance, we applied a t-test with $p = 0.05$. A significant improvement of ML-Plan-UL over another tool is indicated by • and a significant degradation is highlighted by ◦.

The overall image is that ML-Plan-UL performs competitive to TPOT as best performances vary among the datasets and there are neither significant improvements nor degradations. While ML-Plan-UL does not return any result for *cifar10* only (due to exceeding memory usage), TPOT does not manage to return anything (not even an intermediate solution) for nearly half of the datasets. Thus, despite the substantially larger search space ML-Plan-UL manages to find feasible solutions even for larger datasets as compared to TPOT. Furthermore, ML-Plan-UL also performs particularly well compared to auto-sklearn and Auto-WEKA. In comparison to auto-sklearn, we observe 4 significant improvements and 3 significant degradations. For the reference Auto-WEKA, we observe 13 significant improvements and only a single degradation.

## 5. Conclusion

We have presented ML-Plan-UL as an extension of ML-Plan to deal with ML pipelines of unlimited length, i.e., allowing for more complex pre-processing worklflows. To this end, we slightly adapted the search space description by additional tasks that allow for sequential and tree-shaped pre-processing workflows. In our experimental evaluation, we have shown that ML-Plan-UL performs competitive to TPOT and, in contrast to the latter, even manages to return solutions for larger datasets.

## Acknowledgement

This work was partially supported by the German Research Foundation (DFG) within the Collaborative Research Center "On-The-Fly Computing" (SFB 901).

## References

Tsz-Chiu Au, Okhtay Ilghami, Ugur Kuter, J. William Murdock, Dana S. Nau, Dan Wu, and Fusun Yaman. SHOP2: an HTN planning system. *CoRR*, abs/1106.4869, 2011. URL http://arxiv.org/abs/1106.4869.

Cameron Browne, Edward Jack Powley, Daniel Whitehouse, Simon M. Lucas, Peter I. Cowling, Philipp Rohlfshagen, Stephen Tavener, Diego Perez Liebana, Spyridon Samothrakis, and Simon Colton. A survey of monte carlo tree search methods. *IEEE Trans. Comput. Intellig. and AI in Games*, 4(1):1–43, 2012. doi: 10.1109/TCIAIG.2012.2186810. URL https://doi.org/10.1109/TCIAIG.2012.2186810.

Alex Guimarães Cardoso de Sá, Walter José G. S. Pinto, Luiz Otávio Vilas Boas Oliveira, and Gisele L. Pappa. RECIPE: A grammar-based framework for automatically evolving classification pipelines. In *Genetic Programming - 20th European Conference, EuroGP 2017, Amsterdam, The Netherlands, April 19-21, 2017, Proceedings*, pages 246–261, 2017. doi: 10.1007/978-3-319-55696-3_16. URL https://doi.org/10.1007/978-3-319-55696-3_16.

Matthias Feurer, Aaron Klein, Katharina Eggensperger, Jost Springenberg, Manuel Blum, and Frank Hutter. Efficient and robust automated machine learning. In *Proc. NIPS 2015, Advances in Neural Information Processing Systems*, pages 2962–2970, 2015.

Malik Ghallab, Dana Nau, and Paolo Traverso. *Automated Planning: Theory and Practice*. Elsevier, 2004.

Frank Hutter, Holger H Hoos, and Kevin Leyton-Brown. Sequential model-based optimization for general algorithm configuration. *LION*, 5:507–523, 2011.

Jörg-Uwe Kietz, Floarea Serban, Abraham Bernstein, and Simon Fischer. Designing KDD-workflows via HTN-planning. In *ECAI 2012, 20th European Conference on Artificial Intelligence*, pages 1011–1012, 2012. doi: 10.3233/978-1-61499-098-7-1011. URL https://doi.org/10.3233/978-1-61499-098-7-1011.

Brent Komer, James Bergstra, and Chris Eliasmith. Hyperopt-sklearn: Automatic hyperparameter configuration for scikit-learn. In *ICML Workshop on AutoML*, 2014.

James Robert Lloyd, David K. Duvenaud, Roger B. Grosse, Joshua B. Tenenbaum, and Zoubin Ghahramani. Automatic construction and natural-language description of non-parametric regression models. In *Proceedings of the Twenty-Eighth AAAI Conference on Artificial Intelligence, Québec City, Québec, Canada*, pages 1242–1250, 2014.

Felix Mohr, Marcel Wever, and Eyke Hüllermeier. ML-Plan: Automated machine learning via hierarchical planning. *Machine Learning*, Jul 2018.

P. Nguyen, Melanie Hilario, and Alexandros Kalousis. Using meta-mining to support data mining workflow planning and optimization. *J. Artif. Intell. Res.*, 51:605–644, 2014. doi: 10.1613/jair.4377. URL https://doi.org/10.1613/jair.4377.

Randal S. Olson and Jason H. Moore. TPOT: A tree-based pipeline optimization tool for automating machine learning. In *Workshop on Automatic Machine Learning*, pages 66–74, 2016.

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 2016 Genetic and Evolutionary Computation Conference, Denver, CO, USA, July 20 - 24, 2016*, pages 485–492, 2016. doi: 10.1145/2908812.2908918. URL http://doi.acm.org/10.1145/2908812.2908918.

Chris Thornton, Frank Hutter, Holger H. Hoos, and Kevin Leyton-Brown. Auto-WEKA: Combined selection and hyperparameter optimization of classification algorithms. In *The 19th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining, KDD 2013, Chicago, IL, USA*, pages 847–855, 2013.

Joaquin Vanschoren, Jan N. van Rijn, Bernd Bischl, and Luis Torgo. OpenML: Networked science in machine learning. *SIGKDD Explorations*, 15(2):49–60, 2013. doi: 10.1145/2641190.2641198. URL http://doi.acm.org/10.1145/2641190.2641198.