GramML: Exploring Context-Free Grammars with Model-Free Reinforcement Learning

Hernan C. Vazquez
MercadoLibre, Inc
hernan.vazquez@mercadolibre.com

Jorge A. Sánchez
MercadoLibre, Inc
jorge.sanchez@mercadolibre.com

Rafael Carrascosa
MercadoLibre, Inc
rafael.carrascosa@mercadolibre.com

Abstract

One concern of AutoML systems is how to discover the best pipeline configuration to solve a particular task in the shortest amount of time. Recent approaches tackle the problem using techniques based on learning a model that helps relate the configuration space and the objective being optimized. However, relying on such a model poses some difficulties. First, both pipelines and datasets have to be represented with meta-features. Second, there exists a strong dependence on the chosen model and its hyperparameters. In this paper, we present a simple yet effective model-free reinforcement learning approach based on an adaptation of the Monte Carlo tree search (MCTS) algorithm for trees and context-free grammars. We run experiments on the OpenML-CC18 benchmark suite and show superior performance compared to the state-of-the-art.

1 Introduction

Automated machine learning (AutoML) aims at automating the process of applying machine learning to real-world problems (11). In particular, designing and finding the best machine learning pipelines is a labor and time intensive task which requires extensive experimentation and expertise (7). Although different solutions have been proposed for this problem (8), most successful techniques are model-based (9), i.e. rely on a model trained to help explore the solution space efficiently. Learning this model requires a description of the task and data via meta-features as well as setting up an additional set of hyper-parameters. In this work, we present GramML, an approach that combines context-free grammars (19) and a model-free reinforcement learning strategy (13). We propose a grammar that generates pipeline configurations (e.g. python dictionaries) and that allow us to effectively decouple the pipeline generation process from the technology in which they are instantiated. We also propose a model-free Monte-Carlo tree search (13) algorithm specifically adapted to our problem in the sense that it exploits the tree-structured search space induced by the grammar. Experiments on the OpenML-CC18 benchmark (1) show improved performance compared to state-of-the-art approaches.

2 Related Work

This section briefly reviews previous work on AutoML. AutoSklearn is the main representative for solving this kind of problems (5). AutoSklearn is an approach that relies on Bayesian optimization and surrogate models techniques and it is based on the SMAC (10) algorithm. More recently, other model-based approaches have shown good performance compared to AutoSklearn. For instance, MOSAIC (17) combines the surrogate model formulation with a Monte Carlo Tree Search (MCTS) strategy for exploring the hypothesis space. AlphaD3M (4) also builds upon MCTS and propose to use of neural networks as surrogate model. In this case, the model explore sequences of actions over pipeline parts instead of whole configurations. Recently, (3) explored the use of AlphaD3M with a grammar but relying on a pre-trained model to guide the search. PIPER (14) uses a context-free grammar and a greedy heuristic showing competitive results against other techniques. Other approaches, like those based on genetic programming (16; 18) and hierarchical planning (15; 12) are omitted due to space limitations.

3 GramML

Searching for optimal pipeline configurations involves two main problems: 1) how to define the configuration space, and 2) how to explore it efficiently. In the first case, we choose to use context-free grammars (19) as they provide a flexible description of the configuration space and allow for tree-like navigation. We augment such grammar by allowing certain symbols to have properties or operations associated with them. For the second, we use a model-free variation of the MCTS/UCT (13) algorithm. We provide a detailed description below.

3.1 Pipeline Configuration Grammar

Context-free grammars have shown to be well suited for defining a configuration space (3; 14; 12). They are well aligned with our goal of generating configuration objects (python dictionaries or JSON objects) instead of explicit pipeline code. Working with such objects has several advantages, namely: configurations are extensible, human-readable, and easily portable to different formats, helping improve interoperability in service architectures. We call this family of grammars *Pipeline Configuration Grammars*, PCG. A PCG is defined as $< G, g_0, T, R >$, where $G = \{g, g_{node}, g_{step}\}$ is the set of non-terminal symbols, with g a regular non-terminal symbol and g_{node} and g_{step} two augmented symbols that we use to simplify the construction. g_{node} is a g symbol that ends with "_NODE". When reduced, it generates the content between braces ({}). Similarly, g_{step} is a g symbol that ends with "_STEPS". When reduced, it encloses the content in square brackets ([]). g_0 is a g_{node} non-terminal symbol and denotes the initial symbol of the grammar. $T = \{t, t_{component}, t_{column}\}$ is the set of terminal symbols, with t a string and $t_{component}$, t_{column} two augmented symbols. $t_{component}$ is a t symbol that ends with ".component" and represents a pipeline component, e.g. a random forest classifier. When the production algorithm comes across a $t_{component}$ symbol, it automatically loads the component definition. t_{column} is a t symbol that ends with "*type*.columns" where *type* is any of the column types supported, e.g. "numerical.columns". When the production algorithm comes across a t_{column} symbol, it automatically loads the index of the columns of the required type. Finally, R is the set of production rules. Fig. 1 show a simplified example of grammar rules and production. The complete grammar we use in our experiments is based on the same set of components as in

```
ROOT_NODE := PIPELINE & ROOT_STEPS
ROOT_STEPS := PREPROCESS_NODE &
              ESTIMATOR_NODE
PREPROCESS_NODE := NUM_TRANSFORM &
                   NUM_COLUMNS
NUM_TRANSFORM := Normalizer.component |
                 StandardScaler.component
NUM_COLUMNS := "col_index": numerical.columns
ESTIMATOR_NODE := LogisticRegression.component |
                  RandomForest.component
```

(a) Grammar Rules

```
{"class": "Pipeline",
 "steps": [{
    "class": "StandardScaler",
    "col_index": [1,2,3]
   },
   {"class": "RandomForest",
    "hyperparams": [ {"name": "max_depth", "value": 10,
                      "hptype": "int"}],
   },
]}
```

(b) Grammar Production

Figure 1: Simplified examples of grammar rules for pipeline configurations production.

AutoSklearn and it accounts for over 22k different pipeline configurations.

3.2 Model-Free MCTS for PCG

Exploring the space induced by the grammar can be done efficiently with tree search algorithms. In our case, we use MCTS/UCT (13), a bandit-based Monte Carlo tree search algorithm for trees. We choose UCT due to its simplicity and generality. For PCG the game is to generate productions of the grammar $p \in P$ (i.e. pipeline configurations) from the available actions $A(s, g)$ that represent the options given by the production rules for a particular non-terminal symbol $g \in G$ given a partial production $s \in S$, with S is the space of possible partial productions. The algorithm can be stopped at any time and provide the currently best pipeline configuration $p*$, as:

$$p^* = \arg\max_{\lambda \in \Lambda} \mathit{score}(\lambda), \tag{1}$$

where λ is an evaluated pipeline, $score(\lambda)$ denotes the empirical score of evaluating λ in a training set sample, and $\Lambda \subseteq P$ is the set of evaluated pipelines. The longer the algorithm is run, the more productions will be generated and evaluated, and the more likely will be that the recommended pipeline is indeed the optimal one. The process of generating grammar productions and pipeline configuration evaluations consist on the following steps:

Selection. It starts from the root node that represents the initial symbol g_0 and, at each level, selects the next node according to the selection policy. Our selection policy is UCT (13):

$$a^* = \arg\max_{a \in A(s,g)} \left\{ Q(s, a) + C\sqrt{\frac{ln[N(s)]}{N(s, a)}} \right\}, \tag{2}$$

where $A(s, g)$ is a set of actions available for the partial production s and the symbol g ; $Q(s, a)$ denotes the average score of taking action of reducing g with the symbol $a \in (G \cup T)$ to continue the partial production s ; $N(s)$ is the number of evaluated pipeline configuration that start with the partial production s and $N(s, a)$ the number of times symbol a has been sampled after partial production s. The constant C controls the balance between exploration and exploitation.

Expansion. It opens a new path in the tree by taking an action unless the selection phase has reached a terminal state. In our case, a terminal state corresponds to a full pipeline. This stage should never ends in a terminal state.

Simulation. It performs a complete random walk through the grammar to produce a complete pipeline configuration and evaluates it. This is the only phase in which a pipeline configuration is evaluated. All evaluated pipelines λ are stored. The best pipeline is computed as in Eq. 3.2 once the algorithm finishes.

Back-propagation. It propagates the results (scores) back to all nodes along the path from the leaf up to the root. The statistics for Q and N are updated.

4 Evaluation

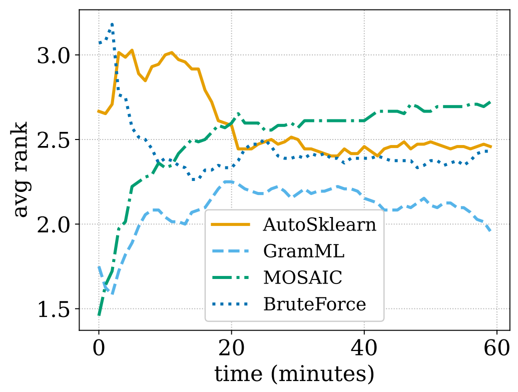
In what follows, we present an experimental evaluation of GramML. We compare our approach with AutoSklearn and MOSAIC as they allow for a one-to-one assessment using the same set of ML components. We also consider a baseline system consisting of expanding the configuration tree induced by the grammar and using a random sampling heuristic, i.e. sampling without replacement over the set of expanded pipelines. We term this model `BruteForce` as it corresponds to a complete evaluation of the configuration space ($\sim$24k different pipelines) if allowed to run long enough. As mentioned above, our model spans the same configuration space as AutoSklearn. Hyperparameters are set to their default values.

4.1 Experimental setup

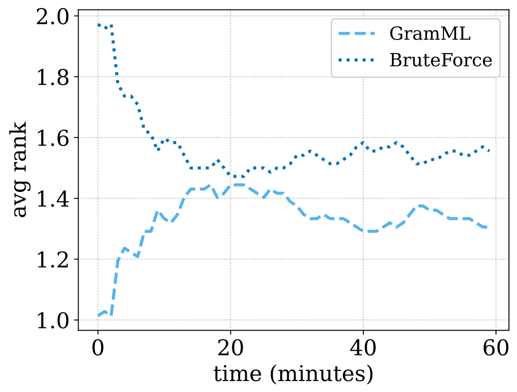
For evaluation, we consider the OpenML-CC18 benchmark (20), a collection of 72 binary and multi-class classification datasets carefully curated from the thousands of datasets on OpenML (1). We use standard train-test splits, as provided by the benchmark suite. Testing performance is measured on the test set after 1-hour execution (time budget of 1 hour) on an Amazon EC2 R5 spot instance (8 vCPU and 64GB of RAM). For each task, we rank the performance obtained by all systems. We report the average of such ranking at each time step.

4.2 Results

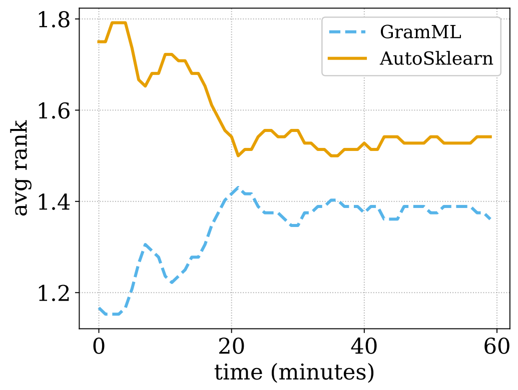
Comparative results are shown in Figure 2. Figure 2a shows the average rank (lower is better) across time. It can be seen a strong prevalence of GramML compared to MOSAIC and AutoSklearn[1]. During the first few minutes, MOSAIC shows the best performance among the three systems but it becomes surpassed by GramML shortly after. An interesting phenomenon can be observed around minute 20, where the performance of MOSAIC and AutoSklearn crosses. This might be attributed to the time it takes for the surrogate model in AutoSklean to stabilize. After this point, the performance of AutoSklearn improves over MOSAIC steadily. Interestingly, BruteForce seems to be a strong baseline and its performance is similar to that of AutoSklearn. Figures 2c, 2d, and 2b show pairwise comparisons of GramML with AutoSklearn, MOSAIC, and BruteForce, respectively. Note how the performance gap between GramML and MOSAIC increases over time. In addition, GramML shows a consistently better average ranking over AutoSklearn and BruteForce, especially in the first 15 minutes. After that, the difference tends to stabilize.



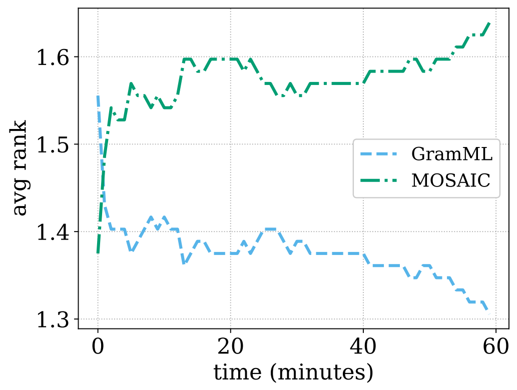
(a) All



(b) GramML vs BruteForce



(c) GramML vs AutoSklearn



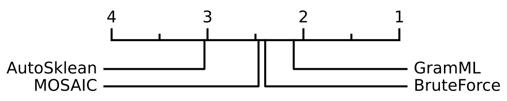
(d) GramML vs MOSAIC

Figure 2: Results of the approaches on OpenML-CC18 benchmark.

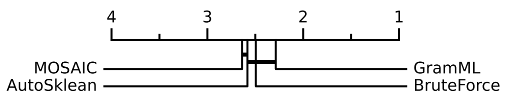
In order to check the presence of statistically significant differences in the average rank distributions, Figure 3 shows the results using critical difference (CD) diagrams (2) for 15, 30, 45 and 60 minutes. We use a non-parametric Friedman test at $p < 0.05$ and a Nemenyi post-hoc test to find which pairs differ (6). If the difference is significant, the vertical lines for the different approaches appear as

[1]We use AutoSklearn `v0.7.0` and MOSAIC `v0.1b0`, both of which rely on scikit-learn `v0.22.2.post1`

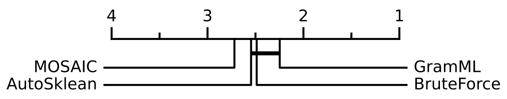
well separated. If the difference is not significant, they are joined by a thick horizontal line. The diagrams show statistically significant differences at 15 minutes between all approaches. At 30 minutes there is a significant difference between MOSAIC and GramML but neither approach shows any critical difference with AutoSklearn. Finally, the difference becomes statistically significant again between all approaches after 1 hour. Interestingly, there seems to be no statistical difference between Autosklearn and BruteForce baseline after 60 minutes of execution



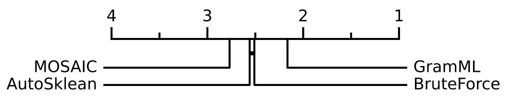
(a) 15 minutes



(b) 30 minutes



(c) 45 minutes



(d) 1 hour

Figure 3: CD plots with Nimenyi post-hoc test

5 Conclusions

In this work, we explore the use of model-free MCTS with context-free grammars to generate pipeline configurations. Experiments on a standard benchmark show statistically significant improvements compared to other approaches. Although more experiments are needed for a thorough evaluation (different machines and problem types), the results are encouraging. Future work will consider horizontal scalability and distributed search strategies.

Acknowledgments

We thank all anonymous reviewers and area chair for their insightful comments and suggestions.

References

[1] Bischl, B., Casalicchio, G., Feurer, M., Hutter, F., Lang, M., Mantovani, R.G., van Rijn, J.N., Vanschoren, J.: Openml benchmarking suites. arXiv:1708.03731v2 [stat.ML] (2019)

[2] Demšar, J.: Statistical comparisons of classifiers over multiple data sets. The Journal of Machine learning research **7**, 1–30 (2006)

[3] Drori, I., Krishnamurthy, Y., Lourenco, R., Rampin, R., Cho, K., Silva, C., Freire, J.: Automatic machine learning by pipeline synthesis using model-based reinforcement learning and a grammar. arXiv preprint arXiv:1905.10345 (2019)

[4] Drori, I., Krishnamurthy, Y., Rampin, R., Lourenco, R.d.P., Ono, J.P., Cho, K., Silva, C., Freire, J.: Alphad3m: Machine learning pipeline synthesis. arXiv preprint arXiv:2111.02508 (2021)

[5] Feurer, M., Klein, A., Eggensperger, K., Springenberg, J., Blum, M., Hutter, F.: Efficient and robust automated machine learning. Advances in neural information processing systems **28** (2015)

[6] Gijsbers, P., Bueno, M.L., Coors, S., LeDell, E., Poirier, S., Thomas, J., Bischl, B., Vanschoren, J.: Amlb: an automl benchmark. arXiv preprint arXiv:2207.12560 (2022)

[7] Gijsbers, P., LeDell, E., Thomas, J., Poirier, S., Bischl, B., Vanschoren, J.: An open source automl benchmark. arXiv preprint arXiv:1907.00909 (2019)

[8] He, X., Zhao, K., Chu, X.: Automl: A survey of the state-of-the-art. Knowledge-Based Systems **212**, 106622 (2021)

[9] Huang, Q.: Model-based or model-free, a review of approaches in reinforcement learning. In: 2020 International Conference on Computing and Data Science (CDS). pp. 219–221. IEEE (2020)

[10] Hutter, F., Hoos, H.H., Leyton-Brown, K.: Sequential model-based optimization for general algorithm configuration. In: International conference on learning and intelligent optimization. pp. 507–523. Springer (2011)

[11] Karmaker, S.K., Hassan, M.M., Smith, M.J., Xu, L., Zhai, C., Veeramachaneni, K.: Automl to date and beyond: Challenges and opportunities. ACM Computing Surveys (CSUR) **54**(8), 1–36 (2021)

[12] Katz, M., Ram, P., Sohrabi, S., Udrea, O.: Exploring context-free languages via planning: The case for automating machine learning. In: Proceedings of the International Conference on Automated Planning and Scheduling. vol. 30, pp. 403–411 (2020)

[13] Kocsis, L., Szepesvári, C.: Bandit based monte-carlo planning. In: European conference on machine learning. pp. 282–293. Springer (2006)

[14] Marinescu, R., Kishimoto, A., Ram, P., Rawat, A., Wistuba, M., Palmes, P.P., Botea, A.: Searching for machine learning pipelines using a context-free grammar. In: Proceedings of the AAAI Conference on Artificial Intelligence. vol. 35, pp. 8902–8911 (2021)

[15] Mohr, F., Wever, M., Hüllermeier, E.: Ml-plan: Automated machine learning via hierarchical planning. Machine Learning **107**(8), 1495–1515 (2018)

[16] Olson, R.S., Moore, J.H.: Tpot: A tree-based pipeline optimization tool for automating machine learning. In: Workshop on automatic machine learning. pp. 66–74. PMLR (2016)

[17] Rakotoarison, H., Sebag, M.: Automl with monte carlo tree search. In: Workshop AutoML 2018@ ICML/IJCAI-ECAI (2018)

[18] de Sá, A.G., Pinto, W.J.G., Oliveira, L.O.V., Pappa, G.L.: Recipe: a grammar-based framework for automatically evolving classification pipelines. In: European Conference on Genetic Programming. pp. 246–261. Springer (2017)

[19] Segovia-Aguas, J., Jiménez, S., Jonsson, A.: Generating context-free grammars using classical planning. In: Proceedings of the Twenty-Sixth International Joint Conference on Artificial Intelligence (IJCAI-17); 2017 Aug 19-25; Melbourne, Australia. Melbourne: IJCAI; 2017. p. 4391-7. IJCAI (2017)

[20] Vanschoren, J., Van Rijn, J.N., Bischl, B., Torgo, L.: Openml: networked science in machine learning. ACM SIGKDD Explorations Newsletter **15**(2), 49–60 (2014)