Autonomous Predictive Modeling via Reinforcement Learning

Udayan Khurana
IBM Research AI
ukhurana@us.ibm.com

Horst Samulowitz
IBM Research AI
samulowitz@us.ibm.com

ABSTRACT

Building a robust predictive model requires an array of steps such as data imputation, feature transformations, estimator selection, hyper-parameter search, ensemble construction, amongst others. Due to this vast, complex and heterogeneous space of operations, off-the-shelf optimization methods offer infeasible solutions for realistic response time requirements. In practice, much of the predictive modeling process is conducted by experienced data scientists, who selectively make use of available tools. Over time, they develop an understanding of the behavior of operators, and perform serial decision making under uncertainty, colloquially referred to as educated guesswork. With an unprecedented demand for application of supervised machine learning, there is a call for solutions that automatically search for a suitable combination of operators across these tasks while minimize the modeling error. We introduce a novel system called *APRL (Autonomous Predictive modeler via Reinforcement Learning)*, that uses past experience through reinforcement learning to optimize sequential decision making from within a set of diverse actions under a budget constraint. Our experiments demonstrate the superiority of the proposed approach over known AutoML systems that utilize Bayesian optimization or genetic algorithms.

CCS CONCEPTS

• **Computing methodologies** → **Supervised learning**.

KEYWORDS

automated machine learning; data science automation; reinforcement learning

ACM Reference Format:
Udayan Khurana and Horst Samulowitz. 2020. Autonomous Predictive Modeling via Reinforcement Learning. In *Proceedings of the 29th ACM International Conference on Information and Knowledge Management (CIKM '20), October 19–23, 2020, Virtual Event, Ireland*. ACM, New York, NY, USA, 4 pages. https://doi.org/10.1145/3340531.3417448

1 INTRODUCTION

In recent years, we have witnessed an increased demand for automated machine learning (*AutoML*) techniques that computationally solve some or all of the steps involved in the applied machine learning process. This has been fueled by a significant supply deficit for demand of data scientists. AutoML as an optimization problem is challenging. It involves a vast and diverse space of options, making it computationally prohibitive to run optimization solvers for obtaining solutions in realistic time frame using affordable resources. For instance, feature transformations for a single feature, using a modest number of unary functions, say 30, for combination depth of 3, itself leads to 3^{30} configurations to choose from; or consider Sklearn's Random Forest classifier that provides 17 hyper-parameters with varying range of possible values, continuous or discrete; selecting the optimal ensemble base models from a set of many constructed models, is yet another combinatorial problem, and so on. One direction is the use of historical information obtained from model building exercises, called *metalearning*, i.e., learning to learn. Different works recently have been able to learn useful clues from historical information to short-circuit the search on the vast space. For instance, in Bayesian Learning, priors on various options are learned, in reinforcement learning-based techniques, pruning or search strategies are tuned using prior runs, or associating feature distribution characteristics with suitable actions, etc. Another direction for AutoML is based on the use of approximations and heuristics for specific tasks such as feature engineering, without considering other aspects such as estimator selection. In this context, it should be noted that the choice of learning algorithm often greatly influences the choice of effective feature engineering transformations, and vice versa. While the collective use of such individual optimizers may not aim towards a joint global optimal solution, they are used effectively by practitioners. Since these techniques are usually based on task specific heuristics, it is difficult to integrate them in a unified framework. In this paper we propose a novel approach called *APRL (Autonomous Predictive modeler via Reinforcement Learning)*, which dwells on the idea of combining the two – historical information and task-specific heuristics algorithms, unified through a reinforcement learning framework.

Usually, in the process of trying different options such as a particular feature transformation, using a particular estimator or certain choice of hyper-parameters, produce several transformed versions of the data which lead to related but slightly varying models. Amongst several such trials on the way, only one or a few models constructed are generally deemed winners and adopted. However, upon carefully selecting a subset of these subpar models, simple but effective ensembles can be created that outperform the impact of the best one alone, owing to their mutual diversity besides the individual strength. Given the tremendous success of ensembles in predictive modeling, APRL has been designed to explicitly optimize for ensembles. APRL goes a step further and not only makes ensembles from the subpar models obtained in the process, but also explicitly aims to create models that would maximize the performance of a final ensemble. This is in contrast to other AutoML systems where the focus is first to find the best individual pipelines first and later ensemble them. We use reinforcement learning (RL)

CIKM '20, October 19–23, 2020, Virtual Event, Ireland

ACM ISBN 978-1-4503-6859-9/20/10…$15.00
https://doi.org/10.1145/3340531.3417448

to device an exploration strategy to take actions that minimize the *ensemble generalization error (EGE)* of a proposed final ensemble. Generally, performance driven AutoML methods are especially prone to overfitting on training data, leading to generalization issues. Overfit pipelines have high variance and low bias. APRL uses ensembles as a central method to drive the AutoML process because it helps lower the overall variance, leading to more generalizable solutions, even though the individual member pipelines have high variance and tend to overfit by themselves. In this paper, we describe the core ideas behind our early work on APRL (please refer here [12] for more details), which includes modeling the action space, exploration process, rewards through ensemble generalization and policy learning through RL. APRL builds on top of our prior work of using RL for feature engineering alone [13].

2 RELATED WORK

Amongst AutoML systems, Auto-sklearn [8] and Auto-WEKA [15] use sequential parameter optimization based on Bayesian Optimization to determine effective predictive modeling pipelines by combining data pre-processors, transformers and estimators. Auto-sklearn performs ensemble selection using a greedy heuristic method [4], once the base pipelines are created. In APRL, however, we propose the reward function so as to create pipelines in a way that aims to optimize a final ensemble. Genetic algorithms have been used as an alternate automated ML method. TPOT [18] is one such example. While it does not create ensembles, it could compose them based on derived models as the authors point out. Bhowan et al. [2] use multi-objective genetic programming to evolve a set of accurate and diverse models through biasing the fitness function. While it does not optimize various steps of ML pipeline, it does however, build ensembles for imbalanced classification problems. Fusi et al. [9] present a recommender system that suggests from a set of pre-existing pipelines. Alpha3DM [7] models pipeline generation as a single-player game and builds pipelines iteratively by selecting among a set of actions which are insertion, deletion, replacement of pipeline parts.

Besides end-to-end AutoML systems, automation of individual tasks such as feature engineering (FE), hyper-parameter optimization (HPO), algorithm selection have been an active area of research recently. Due the lack of space, we refer the interested reader to surveys on AutoML [20], [1], and FE [10]. In this paper, we build upon our prior work on FE based on RL [11, 13, 14, 17]. It is based on trial or exploration of different transformation functions and finding sequences with high returns based on initial feedback. Their trials are organized in a hierarchical, directed acyclic graph and the goal is the minimization of model error. The exploration policy is trained through RL on historical ML problems. In this paper, we extend this approach of a hierarchical structure for exploration, however, with a much diverse space of actions including ensembles, hyper-parameters, estimator selection. In addition, our reward mechanism reflects the need to optimize for ensemble goals of both – high accuracy and diversity instead of model accuracy alone for a single pipeline. Hence, we demonstrate a complete AutoML system based on reinforcement learning.

Model ensembles are used extensively in machine learning to aggregate the output of several weak predictors into a single strong predictor [6]. There exists a body of work on generating diversity for ensembles, surveyed by Brown et al. [3].

3 APRL OVERVIEW

APRL is presented with a classification or regression problem in terms of feature vectors (X) and a target vector (y), and the time constraint (t_{max}) in which to build a model. Figure 1 illustrates APRL's approach. The cornerstone of APRL is an agent that iteratively decides the action to perform, based on two factors – the performance of various actions until that moment and time remaining until when the problem must be solved. In each iteration, the agent must decide to perform an action from the choice of many available ones. It can apply one of the following, to either the given data (X, y) or a version of it that has been derived in the course of this exploration: (a) feature transformations; (b) estimator selection and model building; (c) HPO for a model and transformation. The process of taking such actions is called *data science exploration* or simply *exploration*, which is illustrated in Figure 2. Due to the hierarchical nature of the abstract representation, it is called an *exploration tree*.

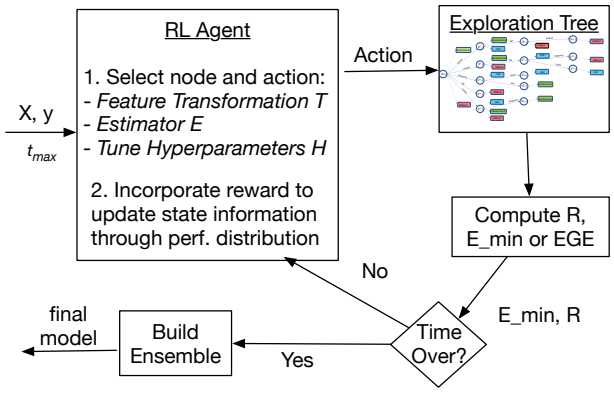

Figure 1: APRL approach Overview. Given a supervised ML problem (X, y) and a time constrain t_{max}, the system generates an ensemble after iterative exploration through multiple transforms, estimators and hyper-parameters.

Similar to [13], applying a feature transformation to a feature-set means generating new features by applying the corresponding function on all possible input features (or feature combinations for non-unary functions), and appending them to the original feature set. For example, applying a *logarithm* transform, $\log(X)$ means creating new features by $\log(x_i) \forall x_i \in X$, such that feature x_i contains only positive values. Similarly, *frequency* transform function, that creates a new feature by counting the number of occurrences of a value is done only for discreet values transforms. In this paper, we present results using a set of 10 different transformation functions, based on experience: *frequency counts, Principal Component Analysis, Rounding, Min-max-scaler, Hyperbolic tangent, Groupby+stddev, Cube root, Sigmoid, Standard scaler, Logarithm*, and a special *feature selection* transformation (Sklearn's univariate FS) that removes unnecessary features. Please note that the method is independent of the actual transformations used and additional or fewer transformations may be used, including domain specific ones.

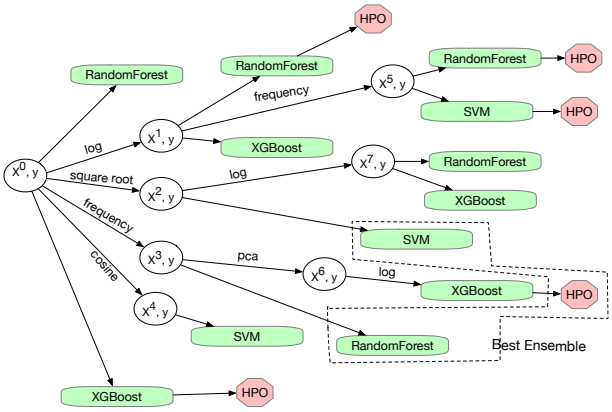

Figure 2: Illustration of an Exploration Tree for a given problem of X^0, y. Circular nodes depict data versions owing to transformations on edges. Rectangular nodes correspond to selected estimators. Hexagonal red blocks represent hyper-parameter optimization, and dotted cluster is an ensemble.

The second category of actions is the choice of a particular estimator, such as *Random Forest*, *Gradient Boosted Trees*, *K-nearest neighbors*, etc., to be chosen on a transformed (or original) data with its default hyper-parameters. The third category of actions is that of a HPO routine to be run on for data node in the tree. We use a hyper-parameter optimization routine based on black-box optimization using radial basis functions [5]. The action specifies a data node, the estimator, hyper-parameter optimization, and the allocated time for it. Additionally, the result of each hyper-parameter optimization step is treated as a starting value set for the next action belonging to the same estimator type. At each iteration, the agent enumerates all actions possible from the current state of the tree (without repetition) and ranked as per the expected long term benefit. The action with the highest expected benefit is chosen and applied. From one state of the tree to the next, the list of possible actions changes. Additionally, the perceived benefit of the actions change as well. The task of taking the best action, given the state of the exploration is performed by the agent based upon a policy, Π that it learns historically on other learning problems using reinforcement learning. Further details on the agent's policy learning, definition of states, and such provided in Section 3.

At each node with an estimator, plain or with hyper-parameter optimization, an prediction of the labels is performed through 5-fold cross validation. The predictions thus generated are required to compute the ensemble generalization error (EGE) for a potential ensemble that may contain that node. It is further used to compute E_{min}, a quantity that reflects the minimum value of EGE for an ensemble from any subset of the nodes of a given tree. This quantity is used as a feedback reward for the agent to learn its policy.

Exploration Policy Learning: We model the exploration process as a Markov Decision Process (MDP), where a state of the system contains all the required information to take the next action at that point. The *state* at any iteration is described by a combination of the exploration tree state (entire tree up to that point with all details such as gain in model accuracy due to transforms, estimators, EGE, etc.) and the fraction of remaining time $(t_{max} - t)/t_{max}$ where $t = 0$ at the beginning. Let the entire set of states be S. An action for this MDP is the application of a particular transformer, estimator or HPO on a node of the tree at that state. Let the entire set of actions be C. A policy, $\Pi : S \rightarrow C$, guides which action should be taken for a given state. We first learn Π iteratively through RL on a set of many prediction problems, and apply it to use for an unseen data.

The "remaining time budget fraction" is a part of the state definition because it is used to determine whether to explore more (i.e., experiment with different transformations and estimators, usually at the beginning of a run) or exploit more (i.e., focus more towards steps that will likely give a desired result based on the exploration so far). It decides whether to focus on exploiting gains (depth) or exploring (breadth) or a compromise, in different regions of the graph, at different steps. Overall, the policy selects the action with the highest expected long-term reward contribution; however, upon evaluating a new node's actual immediate contribution, the expectations are often revised and explore/exploit gears are (implicitly) calibrated through the policy. For example, upon finding an exceptionally high improvement at a node during early stages, the (breadth) exploration can be temporarily localized under that node instead of the same level as it. At step i, the occurrence of an action results in an estimator node, n_i, and the best ensemble error $E_{min}(n_i)$ for a subset of of nodes $\{0, 1 \ldots n_i\}$. To each step, we attribute an immediate scalar reward (with a slight abuse of notation): $r_i = \frac{E_{min}(nodes(\mathcal{T}_{i-1})) - E_{min}(nodes(\mathcal{T}_i))}{E_{min}(\{X_0\})}$, with $r_0 = 0$, by definition, and $E_{min}(\{X_0\})$ is the 5-fold CV performance using any particular estimator. The cumulative reward over time from state s_i onwards is defined as, $R(s_i) = \sum_{j=0}^{N} \gamma^i . r_{i+j}$, where $\gamma \in [0, 1)$ is a discount factor, which prioritizes earlier rewards over the later ones. We find the optimal policy Π^* that maximizes cumulative reward. We use the basic framework of Q-learning [19] to learn action-value Q-function, for each state, $s \in S$ and action, $c \in C$, with respect to policy Π is: $Q(s, c) = r(s, c) + \gamma R^{\Pi}(\delta(s, c))$, where $\delta : S \times C \rightarrow S$ is a hypothetical transition function, and $R^{\Pi}(s)$ is the cumulative reward after state s. The optimal policy is: $\Pi^*(s) = \arg\max_c [Q(s, c)]$. Please refer to [12] for details on our Q-function approximation as well as the choice of input parameters for state description.

Ensemble Generalization Error: We now describe the criterion to compute the ensemble generalization error (EGE) for a given set of base models and an algorithm to select a subset from a given set of models thats maximizes the EGE. While it is used to create a final ensemble, more importantly it is also used to compute a reward for the RL training and runtime stages as described in the previous section. Krogh et al. [16] provide a useful expression for computing the ensemble generalization error. For a number of base models and the output of model, α on input, x, be $V^{\alpha}(x)$. Let a weighted average ensemble output on x be, $\overline{V}(x) = \sum_{\alpha} w_{\alpha} V^{\alpha}(x)$. The ambiguity on input, x of a single member of the ensemble is defined as, $a^{\alpha} = (V^{\alpha}(x) - \overline{V}(x))^2$. Then, the overall ambiguity of the ensemble on input x is: $\bar{a}(x) = \sum_{\alpha} w_{\alpha} a^{\alpha}(x) = \sum_{\alpha} w_{\alpha} (V^{\alpha}(x) - \overline{V}(x))^2$.

We propose a simple yet effective greedy algorithm (E_{min}) to find the suitable subset of ensemble constituents in an efficient manner. For more details on that and further analysis on ensemble generalization error, we refer the interested reader to [12].

	Base Estimator AUC		AUC after 30 mins optimization			Reduction in error (1-AUC) over Random Forest (%age)			Reduction in error (1-AUC) over XGBoost (%age)		
Classification data	RF	XGB	ASKL	TPOT	APRL	ASKL	TPOT	APRL	ASKL	TPOT	APRL
pc2	0.5000	0.5000	0.8464	0.8241	**0.8823**	69.29	64.82	76.46	69.29	64.82	76.46
numerai28.6	0.5079	0.5174	0.5292	0.5281	**0.5605**	4.33	4.10	10.69	2.44	2.22	8.92
mc1	0.5789	0.5263	0.8211	0.7638	**0.8475**	57.51	43.91	63.79	62.23	50.14	67.81
Hyperplane_10_1E-3	0.6618	0.6949	0.7344	0.7644	**0.7812**	21.47	30.34	35.31	12.94	22.78	28.29
bank-marketing	0.6634	0.6678	0.9272	0.9330	**0.9512**	78.37	80.10	85.49	78.08	79.83	85.30
CreditCardSubset	0.6665	0.6667	0.7383	**0.8159**	0.7272	21.53	44.79	18.20	21.50	44.75	18.17
BNG(credit-g)	0.7043	0.7242	0.8730	0.8182	**0.8952**	57.06	38.52	64.57	53.96	34.08	62.01
bank32nh	0.7061	0.7809	0.8980	0.8988	**0.9169**	65.28	65.57	71.73	53.43	53.81	62.08
puma32H	0.7898	0.8808	0.9617	0.9650	**0.9883**	81.76	81.78	94.42	67.84	67.87	90.15
kin8nm	0.8055	0.7829	0.9664	0.9478	**0.9741**	82.72	82.72	86.68	84.52	84.52	88.07
Mean error reduction (above ten datasets)						53.93	53.67	**60.73**	50.62	50.48	**58.73**
Median error reduction (above ten datasets)						57.29	54.81	**64.18**	53.70	48.89	**62.05**
Mean error reduction (all 56 datasets)						64.98	62.51	**71.10**	62.07	59.84	**68.4**
Median error reduction (all 56 datasets)						68.32	65.09	**73.54**	66.15	61.07	**76.59**

Table 1: Comparing the accuracy of Random Forest, XGBoost Classifier with AutoSklearn and APRL ran for 30 minutes each. The fractional improvement is reported over reduction in error (1 - AUC) over the corresponding base model's.

4 EVALUATION

We evaluated on 56 binary classification problems from OpenML, which are different from those used to learn the policy. We summarize all the results, while providing details for a subset of 10 amongst them (with lowest base performance). Our evaluation through out is based on splitting using a fixed seed (1) through Sklearn's train-test split. All the candidates were only provided with 67% of the rows and all numbers reported were based on evaluation on the unseen 33%, using the metric Area under ROC curve. We report the relative reduction in AUC error compared to a base estimator as a measure of comparing across different models and aggregating results across different datasets. It is described as: $\frac{(1-AUC_{base})-(1-AUC_{opt})}{1-AUC_{base}}$.

When compared to Auto-sklearn and TPOT binary classification problems, APRL performed better given the same duration of runtime (Table 1). Similar overall trend can be seen in regression problems, in terms of relative reduction of root mean squared error [12]. APRL also fared better when compared to the RL-based approach of feature engineering [13], alone, or when followed by an independent ensemble step, when run for same time duration. These extended results, training parameters, and relative behavior of different approaches over varying time can be found in [12].

Conclusion and Future Work: In this paper, we presented a novel AutoML framework called APRL, that uses reinforcement learning to produce effective ensembles resulting from combinations of diverse optimizers for various ML steps. At the heart of APRL, an autonomous agent strategizes time-aware contextual decision making of balancing various data science operations. The strategy is learnt and optimized over historical problems in ML and transferred to new unseen problems. We demonstrated the impact of our method by comparison with two well-known openly available AutoML methods – Auto-Sklearn and TPOT, amongst others. On an average, APRL reduces error over strong base estimators such as Random Forest, XGBoost by approximately 70%. Further, we are exploring Deep Q-learning to improve the contextual strategy, along with much larger training datasets. Our other focus is to incorporate a much larger space space of operations and methods for different ML steps.

REFERENCES

[1] Charu Aggarwal, Djallel Bouneffouf, Horst Samulowitz, Beat Buesser, Thanh Hoang, et al. 2020. How can AI Automate End-to-End Data Science? *IJCNN*.
[2] Urvesh Bhowan et al. 2013. Evolving Diverse Ensembles Using Genetic Programming for Classification With Unbalanced Data. *Trans. Evol. Comp* (2013).
[3] Gavin Brown, Jeremy Wyatt, Rachel Harris, and Xin Yao. 2005. Diversity creation methods: A survey and categorisation. *Information Fusion* 6, 1 (2005), 5–20.
[4] Rich Caruana, Alexandru Niculescu-Mizil, Geoff Crew, and Alex Ksikes. 2004. Ensemble Selection from Libraries of Models. In *ICML*.
[5] Alberto Costa and Giacomo Nannicini. 2018. RBFOpt: an open-source library for black-box optimization with costly function evaluations. *Mathematical Programming Computation* 10, 4 (2018).
[6] Thomas G. Dietterich. 2000. Ensemble Methods in Machine Learning. *Multiple Classifier Systems* 1857 (2000). arXiv:arXiv:1011.1669v3
[7] Iddo Drori, Yamuna Krishnamurthy, Remi Rampin, Kyunghyun Cho, et al. 2018. AlphaD3M. *International Workshop on AutoML* (2018).
[8] Matthias Feurer, Aaron Klein, et al. 2015. Efficient and robust automated machine learning. In *Advances in neural information processing systems*.
[9] Nicolo Fusi, Rishit Sheth, and Melih Elibol. 2018. Probabilistic Matrix Factorization for Automated Machine Learning. In *NeurIPS*. 3352–3361.
[10] Udayan Khurana. 2018. Transformation-based Feature Engineering in Supervised Learning: Strategies toward Automation. In *Feature Engineering for Machine Learning and Data Analytics*, Guozhu Dong and Huan Liu (Eds.). Chapter 9.
[11] Udayan Khurana, Fatemeh Nargesian, et al. 2016. Automating Feature Engineering. *NIPS workshop on Artificial Intelligence for Data Science* (2016).
[12] Udayan Khurana and Horst Samulowitz. 2019. Automating Predictive Modeling Process using Reinforcement Learning. *arXiv preprint arXiv:1903.00743* (2019).
[13] Udayan Khurana, Horst Samulowitz, and Deepak Turaga. 2018. Feature Engineering for Predictive Modeling using Reinforcement Learning. In *AAAI*.
[14] Udayan Khurana, Deepak Turaga, Horst Samulowitz, et al. 2016. Cognito: Automated Feature Engineering for Supervised Learning. In *IEEE ICDM*.
[15] Lars Kotthoff et al. 2017. Auto-WEKA 2.0: Automatic Model Selection and Hyperparameter Optimization in WEKA. *J. Mach. Learn. Res.* (2017).
[16] A Krogh and J Vedelsby. 1995. Neural network ensembles, cross validation, and active learning. *Advances in neural network processing systems* 7 (1995), 8–231.
[17] Fatemeh Nargesian, Udayan Khurana, Tejaswini Pedapati, et al. 2018. Dataset Evolver: An Interactive Feature Engineering Notebook.. In *AAAI*.
[18] Randal S Olson and Jason H Moore. 2016. TPOT: A tree-based pipeline optimization tool for automating machine learning. In *Workshop on AutoML*.
[19] Christopher JCH Watkins and Peter Dayan. 1992. Q-learning. *Machine learning* 8, 3-4 (1992).
[20] Marc-André Zöller and Marco F Huber. 2019. Survey on automated machine learning. *arXiv preprint arXiv:1904.12054* 9 (2019).