Learning Decision Trees with Reinforcement Learning

Zheng Xiong [*1] **Wenpeng Zhang** [*2] **Wenwu Zhu** [1,2]
([*] Equal Contribution)
[1]Tsinghua-Berkeley Shenzhen Institute [2]Department of Computer Science and Technology
Tsinghua University
[1]Shenzhen, [2]Beijing, China
xiongz17@mails.tsinghua.edu.cn zhangwenpeng0@gmail.com wwzhu@tsinghua.edu.cn

Abstract

Decision trees are usually learned by heuristic methods like greedy search, which only considers immediate information gain at the current splitting node and may find sub-optimal solutions in a constrained search space. In this paper, we train a RNN controller with reinforcement learning to automatically search for the splitting strategy with optimal long-term payoff in the global search space. Empirically, decision trees generated by our method slightly outperform those generated by commonly used greedy search methods under the same hyper-parameter setting.

1 Introduction

Decision trees are powerful machine learning models which are widely applied in real-world applications. They are white-box models based on the "if-else" conditions and are easy to interpret. They can handle numerical and categorical data simultaneously, and require little data preprocessing. Furthermore, decision trees can be easily integrated into ensemble frameworks like random forest [8] and gradient boosting [6] to further improve their performance.

However, learning an optimal decision tree is known to be NP-complete. As the search space of tree induction is too huge to be explored comprehensively, different heuristic methods have been proposed to learn a decision tree. Among these methods, greedy search is most commonly used as it is easy to understand and implement, meanwhile yields acceptable results on various tasks. However, greedy search only considers short-term information gain in the current step and makes local optimal decision at each node, which may lead it to sub-optimal solutions in a constrained search space.

In this paper, we propose a reinforcement learning approach to automatically search for potential better splitting strategies in the global search space based on evaluations of long-term payoff. As the key point of learning a decision tree is to decide on the feature used for each split, which can be modeled as a sequential decision process in a finite time-invariant action space, we employ a RNN controller to predict the splitting feature for each non-leaf node orderly. The RNN controller is trained with reinforcement learning to search for decision trees with better performance. It is expected to learn the optimal sampling distribution in the action space and provide a set of satisfying splitting strategies after training.

We name the decision tree learned by our method as *RDT*, which is the abbreviation of both "reinforcement learning based decision tree" and "RNN generated decision tree". The effectiveness of RDT is tested on a binary classification task using 6 UCI datasets, which cover different sizes of search space ranging from 10^{13} to 10^{45}. We find that RDT yields better performance than those generated by commonly used greedy search methods, such as CART [5], under the same hyper-parameter setting.

31st Conference on Neural Information Processing Systems (NIPS 2017), Long Beach, CA, USA.

2 Related Work

Our work belongs to the category of research called meta-learning [15], a general framework to learn (and optimize) the performance of learning methods based on data.

An important research area in meta-learning is to find the optimal design of a learning model, such as hyper-parameter configuration and the design of optimization algorithm. The choice of design can be regarded as a sampling problem in a huge action space with complicated interactions between different dimensions. Since exhaustive search is infeasible due to unaffordable computation cost, many researchers have been focusing on heuristic algorithms. However, human-designed methods require a lot of expertise and engineering, and are usually restricted in a constrained search space, as the whole search space is too complicated to analyze thoroughly.

Several automatic sampling methods have been proposed to settle this problem. Bayesian optimization refines its choices sequentially via Bayesian posterior updating as more data is observed, which has been widely applied in hyper-parameter optimization [4, 9, 13, 16]. Another method trains a meta-learner, like a neural network, to approximate and optimize the algorithm used in learning tasks. For example, [1, 12] replaced human-designed optimization algorithms with direct numerical updates generated by an LSTM optimizer, which is jointly trained with the learner network.

The line of work that is closely related to our method is to explore for the optimal design with reinforcement learning [7], a method that has been proved effective and efficient for finding satisfying solutions in a huge action space [14]. Recent works proposed to predict the choice for a learning task with the sequential output of a RNN controller. The key insight is to train this RNN controller with reinforcement learning to maximize the expected reward it gets from applying the chosen strategy on a given task. This framework has acquired promising solutions on a variety of problems, like neural architecture search [18], neural optimizer search [3], combinatorial optimization [2] and device placement [10].

3 Methodology

To learn a decision tree, we need to choose the splitting feature at each non-leaf node and the splitting value of this feature. As the set of possible splitting values varies with the feature chosen and the sample distribution at the current node, it is hard to predict the splitting value directly. To simplify the problem, we use the RNN controller only to predict the splitting feature at each node. Once the feature is selected, the feature value which can maximize the information gain of the current split will be chosen as the splitting value.

To fix the sequence length of the RNN controller, we assume the decision tree to be a complete binary tree with depth k. So the total number of non-leaf nodes in the decision tree is $(2^k - 1)$, and they are indexed sequentially from left to right layer by layer. The splitting feature at the ith node is chosen according to the controller's prediction in the ith step.

Based on these two assumptions, our controller is implemented as a RNN which runs for $(2^k - 1)$ steps, with the output vector corresponding to the feature set element-wisely. The training process is shown in Figure 1. The RNN controller samples a feature according to the softmax output distribution and uses this one-hot encoded prediction as input vector for the next step. Once the controller has run for $(2^k - 1)$ steps, the sequence of selected features will be used to build the decision tree.

The decision tree is built from the root recursively. If the current node satisfies pre-set early stop conditions, it turns into a leaf node, and the predictions for its subtree remain unused. Otherwise, the splitting value of this feature is selected to maximize the information gain respect to gini index. Once the tree has been established, its performance score on the validation set will be fed back as reward signal to update the controller's parameters θ. The training objective of the controller is to maximize its expected reward, represented by $J(\theta)$:

$$J(\theta) = E_{P(a_1:a_T;\theta)}[R] \tag{1}$$

Since the reward signal R is non-differentiable respect to θ, we use REINFORCE [17], a policy gradient method, to calculate the gradient. An empirical approximation with baseline function is:

$$\nabla_\theta J(\theta) = \frac{1}{m} \sum_{k=1}^{m} \sum_{t=1}^{T} \nabla_\theta \log P(a_t | a_{(t-1):1}; \theta)(R_k - b) \tag{2}$$

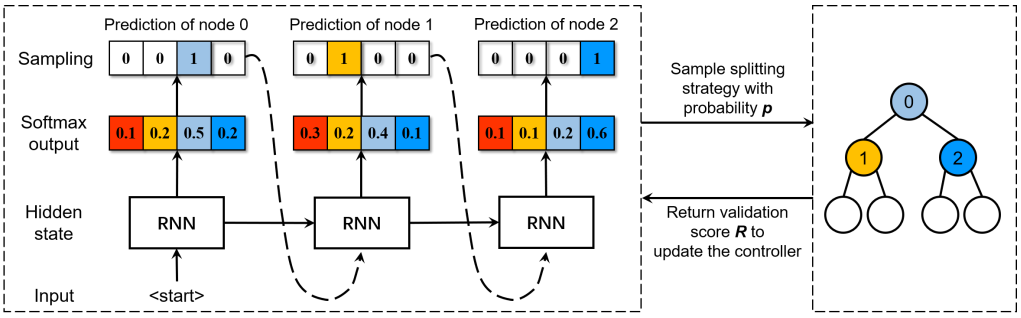

Figure 1: Model framework. Each color represents a unique feature. The feature used at the ith node is predicted stochastically according to the softmax output of the RNN in the ith step. We do not directly select the feature with the maximal probability in order to aid in exploration. The initial input vector <start> is updated as parameters of the RNN.

where m is the number of different strategies that the controller samples in one batch; T is the number of splitting features to be predicted; b is the exponential moving average of reward signal, which is used as a baseline to reduce the variance of the estimation.

4 Experiments

4.1 Experimental Setup

We experiment on a binary classification task to evaluate the performance of our proposed method. We use 6 datasets selected from the UCI[1] repository with different sizes of search space, which ranges from 10^{13} to 10^{45}. We remove the samples with missing values, and transfer categorical features into one-hot encoding. We use AUC(area under the curve) as the performance score and evaluation metric, as it is indifferent to class imbalance and provides decent evaluation of classifier performance. The summary of the datasets is illustrated in Table 1.

Table 1: Summary of the 6 binary-class datasets.

Datasets	Heart	Breast	Pima	German	HTRU	Credit
# instances	270	569	768	1000	17898	30000
# features	20	30	8	24	8	23

We split each dataset into training, validation and test set in proportion of $50\%, 25\%, 25\%$. The training set is used to build the decision tree, the validation set is used to evaluate the performance of the selected classifier, and the test set is used to evaluate the performance of the final model.

As the feature number and data distribution of each dataset are different, we train a RNN controller on each dataset separately. We firstly train a decision tree with CART algorithm using the scikit-learn library [11]. We use gini index as the split criterion, fine-tune tree depth and min_samples_leaf (the minimum number of samples required to split an internal node) to maximize the classifier's AUC score on the validation set. Once the optimal CART model has been found, we set it as the baseline classifier, and use its tree depth to determine the sequence length of our RNN controller.

The decision tree sampled by our controller is trained with the same hyper-parameters as the baseline model, and the performance score is fed back as reward signal. During the training process, we keep a list of the top-k sampled decision trees with the highest reward. After training, we select the top-1 model as the final model, and calculate its AUC score on the test set to compare with CART baseline.

Across all the experiments, our controller RNN is trained with the ADAM optimizer, and the learning rate is set to 0.0005. The controller is a single-layer RNN, and its weights are initialized uniformly at random between -0.08 and 0.08. The hidden state size is set to 2^m, where m is the minimum integer

[1] http://archive.ics.uci.edu/ml/datasets

that satisfies $2^m \geq \#feature$. To reduce the influence of random sampling on our experimental results, we repeat the training process for 5 independent epochs on each dataset. The mean value and standard deviation of the AUC score of the final model in each epoch are calculated to test the average performance and stability of the RNN controller.

4.2 Experimental Results

Figure 2 shows the AUC score of RDT as a function of the number of training iterations (due to the page limit, we only put the results of 3 datasets here. Please refer to the supplementary material for the complete results). The results illustrate that the RNN controller starts from random sampling, gradually discovers decision trees with increasing performance, and finally outperform baseline models after training for a period of time.

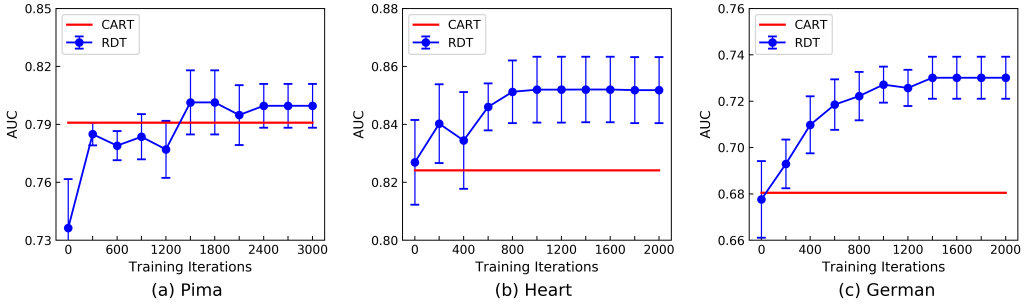

Figure 2: RDT's performance on the test set.

Table 2 shows the final AUC score of RDT and CART on validation and test set. *RDT Avg* is the average AUC score of the 5 independent epoches, and *RDT Top* is the AUC score of the final model with the best performance on the validation set in all 5 epoches.

Table 2: Final AUC score of RDT and CART.

Datasets	Validation Set			Test Set			Search	Iters
	CART	RDT Avg	RDT Top	CART	RDT Avg	RDT Top	Space	
Pima	80.47	$\mathbf{80.93 \pm 0.44}$	81.54	79.09	$\mathbf{79.96 \pm 1.70}$	78.90	10^{13}	3000
Heart	91.04	$\mathbf{94.73 \pm 1.09}$	96.98	82.41	$\mathbf{85.18 \pm 1.71}$	87.19	10^{19}	2000
Breast	99.18	$\mathbf{99.78 \pm 0.27}$	99.99	95.15	$\mathbf{95.22 \pm 2.06}$	95.93	10^{45}	2000
German	68.59	$\mathbf{74.84 \pm 0.96}$	76.19	68.05	$\mathbf{73.01 \pm 1.36}$	74.43	10^{42}	2000
HTRU	96.91	$\mathbf{96.92 \pm 0.17}$	97.09	96.12	$\mathbf{96.70 \pm 0.23}$	97.03	10^{13}	2000
Credit	75.79	$\mathbf{75.88 \pm 0.07}$	75.98	75.22	$\mathbf{75.23 \pm 0.29}$	75.44	10^{42}	3000

The results demonstrate that RDT slightly outperforms CART baseline on all the datasets. We also notice that RDT becomes less advantageous compared to greedy search method as the data size and feature number increase. One possible reason is that the search space has become too huge and complicated to be explored effectively by the primitive algorithms, like REINFORCE, used in our work. Thus, future works are expected to introduce more advanced RNN architectures and reinforcement learning algorithms to further improve the efficiency and scalability of RDT.

5 Conclusion

In this work, we proposed a proof-of-concept framework for learning decision trees with reinforcement learning. A RNN controller is utilized to predict the splitting feature at each node, and trained with reinforcement learning to search for decision trees with better performance. Empirical results show that decision trees learned by our method slightly outperform those generated by commonly used greedy search methods like CART. Potential directions for future work include introducing more advanced algorithms to improve the effectiveness and scalability of RDT, and applying this framework to more powerful models like random forest and gradient boosting.

References

[1] Marcin Andrychowicz, Misha Denil, Sergio Gomez, Matthew W Hoffman, David Pfau, Tom Schaul, and Nando de Freitas. Learning to learn by gradient descent by gradient descent. In *Advances in Neural Information Processing Systems*, pages 3981–3989, 2016.

[2] Irwan Bello, Hieu Pham, Quoc V Le, Mohammad Norouzi, and Samy Bengio. Neural combinatorial optimization with reinforcement learning. *arXiv preprint arXiv:1611.09940*, 2016.

[3] Irwan Bello, Barret Zoph, Vijay Vasudevan, and Quoc V Le. Neural optimizer search with reinforcement learning. In *International Conference on Machine Learning*, pages 459–468, 2017.

[4] James Bergstra, Daniel Yamins, and David Cox. Making a science of model search: Hyperparameter optimization in hundreds of dimensions for vision architectures. In *International Conference on Machine Learning*, pages 115–123, 2013.

[5] Leo Breiman, Jerome Friedman, Charles J Stone, and Richard A Olshen. *Classification and regression trees*. CRC press, 1984.

[6] Jerome H Friedman. Greedy function approximation: a gradient boosting machine. *Annals of statistics*, pages 1189–1232, 2001.

[7] Ke Li and Jitendra Malik. Learning to optimize. In *International Conference on Learning Representations*, 2017.

[8] Andy Liaw, Matthew Wiener, et al. Classification and regression by randomforest. *R news*, 2 (3):18–22, 2002.

[9] Hector Mendoza, Aaron Klein, Matthias Feurer, Jost Tobias Springenberg, and Frank Hutter. Towards automatically-tuned neural networks. In *Workshop on Automatic Machine Learning*, pages 58–65, 2016.

[10] Azalia Mirhoseini, Hieu Pham, Quoc V Le, Benoit Steiner, Rasmus Larsen, Yuefeng Zhou, Naveen Kumar, Mohammad Norouzi, Samy Bengio, and Jeff Dean. Device placement optimization with reinforcement learning. In *International Conference on Machine Learning*, pages 2430–2439, 2017.

[11] Fabian Pedregosa, Gaël Varoquaux, Alexandre Gramfort, Vincent Michel, Bertrand Thirion, Olivier Grisel, Mathieu Blondel, Peter Prettenhofer, Ron Weiss, Vincent Dubourg, et al. Scikit-learn: Machine learning in python. *Journal of Machine Learning Research*, 12(Oct):2825–2830, 2011.

[12] Sachin Ravi and Hugo Larochelle. Optimization as a model for few-shot learning. In *International Conference on Learning Representations*, 2017.

[13] Bobak Shahriari, Kevin Swersky, Ziyu Wang, Ryan P Adams, and Nando de Freitas. Taking the human out of the loop: A review of bayesian optimization. *Proceedings of the IEEE*, 104(1): 148–175, 2016.

[14] D Silver, J Schrittwieser, K Simonyan, I Antonoglou, A Huang, A Guez, T Hubert, L Baker, M Lai, A Bolton, et al. Mastering the game of go without human knowledge. *Nature*, 550 (7676):354, 2017.

[15] Sebastian Thrun and Lorien Pratt. *Learning to learn*. Springer Science & Business Media, 2012.

[16] Ziyu Wang, Frank Hutter, Masrour Zoghi, David Matheson, and Nando de Feitas. Bayesian optimization in a billion dimensions via random embeddings. *Journal of Artificial Intelligence Research*, 55:361–387, 2016.

[17] Ronald J Williams. Simple statistical gradient-following algorithms for connectionist reinforcement learning. *Machine learning*, 8(3-4):229–256, 1992.

[18] Barret Zoph and Quoc V Le. Neural architecture search with reinforcement learning. In *International Conference on Learning Representations*, 2017.