Decision Trees for Tactic Prediction in Coq [*]

Liao Zhang[1,3], Lasse Blaauwbroek[1,2], Bartosz Piotrowski[1,4], Cezary Kaliszyk[3,4], and Josef Urban[1]

[1] Czech Technical University, Prague, Czech Republic
[2] Radboud University, Nijmegen, The Netherlands
[3] University of Innsbruck, Austria
[4] University of Warsaw, Poland

1 Introduction

We present ongoing work of comparing several different decision tree learning methods on predicting tactics for proof states in Coq [5] by grasping knowledge from preceding human-written proofs. The work is based on Tactician[1], a plugin enabling proof automation for Coq. The authors use Tactician to extract features from the abstract syntax trees of proof states in the form of term walks of length one and two. We apply Tactician to record all the proof states $(253,538)$ and respective tactics in the Coq standard library and randomly choose 1% $(2,530)$ as the evaluation data. Three algorithms are implemented: random forests and gradient boosted trees working on binary learning with negative examples and multilabel regression. Finally, we compare the results of the decision tree models and the Tactician's original method, k-nearest neighbors (k-NN).

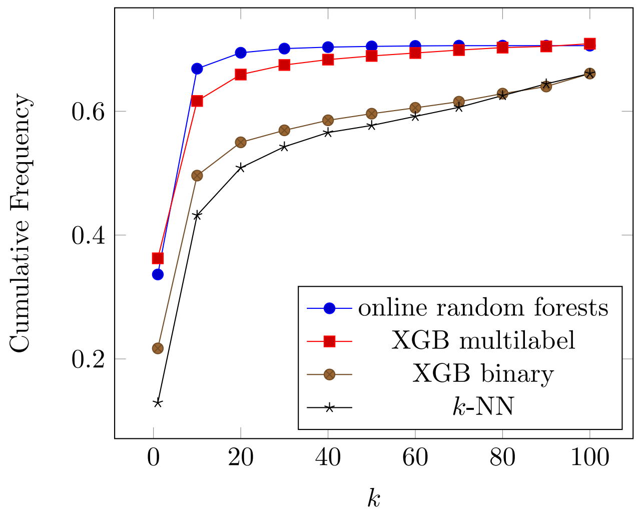
2 Tactic Prediction Models

After establishing the dataset of state features and tactics as outlined above, we can apply machine learning algorithms for tactic prediction. The naive k-NN simply computes the similarity between proof state features by the Jaccard index [4] to find likely tactics.

Random forests, a popular machine learning method, provide predictions by voting from multitude decision trees. Typically, random forests will not update the structure after training. Since grasping local knowledge is essential for Tactician, we take the inspiration from [3] and implement an online version here [6] capable of incremental optimization. When a new training instance is passed to a leaf of the decision tree, we split the leaf and create a new decision node with two successive leaves. Under a certain size limitation, a new tree grows in the forests with probability $\frac{1}{n}$ where n is the current tree number. To make predictions for a given example, it should be passed to the respective leaves (denoting a tactic to be predicted) of all the trees by the decision rules, and the tactics are ranked by their occurring frequency in the pointed leaves.

Gradient boosted trees apply boosting techniques, appropriate combinations of weak base learners, to construct a powerful learner, and XGBoost[2] is such a popular library. On the binary regression setting, XGBoost receives (f, t) where f denotes the features of the proof state, and t represents the hash of the corresponding tactic in the Coq library. The input is labeled 0 or 1 according to the tactic being advantageous or not for the proof state. A tactic for a proof state gets a positive label if it is exactly the one applied to the state in the library. In contrast,

[*]This work was supported by the ERC grant no. 714034 *SMART*, by the European Regional Development Fund under the project AI&Reasoning (reg. no. CZ.02.1.01/0.0/0.0/15_003/0000466), and by the Ministry of Education, Youth and Sports within the dedicated program ERC CZ under the project *POSTMAN* no. LL1902.



negative tactics are the elements in the tactic space and differ from the positive instance. We apply k-NN to predict tactics for a state in the library by learning from the preceding states and arbitrarily select a subset from the best-100 predictions as the negative instances. While predicting tactics for a certain state, we first preselect the top-100 k-NN predictions. Then, we use the XGBoost model to obtain scores for each tactic and return tactics by the order of the corresponding scores. The input to XGBoost on multilabel regression is simply f with the respective positive tactic of the state as the label, which builds an individual regressor for each label. Every label's regressor performs binary regression on two kinds of data: the proof states with the same label (tactic) applied and the others. For a given example, each tree model calculates a score, and we predict the tactics related to the models in score order.

3 Experimental Evaluation

The online random forests learn from all the previous states and dynamically update the structure during the prediction. Since the introduced XGBoost algorithms are offline, we build an individual model for each state while performing experiments. To speed up the XGBoost evaluation, we merely provide the previous $1,000$ proof states encountered before the current proof state for training.

Figure 3 depicts the accuracy of the presented machine learning methods. The k-NN always has the lowest accuracy rates. In the beginning, the XGBoost's multilabel regression performs best, while the random forests surpass others later. The binary regression setting is much worse than the other decision tree approaches for all the k. This indicates that we need better semantic characterizations of tactics instead of the naive hash.

References

[1] Lasse Blaauwbroek, Josef Urban, and Herman Geuvers. The tactician. In *International Conference on Intelligent Computer Mathematics*, pages 271–277. Springer, 2020.

[2] Tianqi Chen and Carlos Guestrin. Xgboost: A scalable tree boosting system. In *Proceedings of the 22nd acm sigkdd international conference on knowledge discovery and data mining*, pages 785–794, 2016.

[3] Pedro Domingos and Geoff Hulten. Mining high-speed data streams. In *Proceedings of the sixth ACM SIGKDD international conference on Knowledge discovery and data mining*, pages 71–80, 2000.

[4] Paul Jaccard. The distribution of the flora in the alpine zone. 1. *New phytologist*, 11(2):37–50, 1912.

[5] The Coq Development Team. The Coq proof assistant, version 8.11.0, Oct 2019.

[6] Liao Zhang, Lasse Blaauwbroek, Bartosz Piotrowski, Prokop Černỳ, Cezary Kaliszyk, and Josef Urban. Online machine learning techniques for coq: A comparison. *arXiv preprint arXiv:2104.05207*, 2021.