JMLR: Workshop and Conference Proceedings 1:1–9, 2015 12th European Workshop on Reinforcement Learning

A Reinforcement Learning Approach to Online Learning of Decision Trees

Abhinav Garlapati* ABHINAVG@CSE.IITM.AC.IN
Aditi Raghunathan* ADITIRAG@CSE.IITM.AC.IN
Vaishnavh Nagarajan* VAISH@CSE.IITM.AC.IN
Balaraman Ravindran RAVI@CSE.IITM.AC.IN
Department of Computer Science and Engineering
Indian Institute of Technology, Madras

Abstract

Online decision tree learning algorithms typically examine all features of a new data point to update model parameters. We propose a novel alternative, Reinforcement Learning-based Decision Trees (RLDT), that uses Reinforcement Learning (RL) to actively examine a minimal number of features of a data point to classify it with high accuracy. Furthermore, RLDT optimizes a long term return, providing a better alternative to the traditional *myopic* greedy approach to growing decision trees. We demonstrate that this approach performs as well as batch learning algorithms and other online decision tree learning algorithms, while making significantly fewer queries about the features of the data points. We also show that RLDT can effectively handle concept drift.

Keywords: Decision Trees, Online Learning, Reinforcement Learning

1. Introduction

Decision trees sequentially test the values of various features of a data point to report a class label. The feature to be inspected at a stage is a decision that depends on the results of the tests conducted previously. It is natural to perceive this problem as a sequential decision-making task, which can be solved using Reinforcement Learning (RL). More importantly, when compared to traditional decision tree learning, we will see how the different constructs in RL offer better control over multiple aspects of the learning algorithm.

In this work, we present RLDT, an RL-based online decision tree algorithm for classification.Our objective is to design a learner that can achieve high accuracy while simultaneously minimizing the amount of information about data points used for both predicting *and* learning. We show that learning such a minimal tree is equivalent to solving for the optimal policy of an unknown Markov Decision Process (MDP).

A challenge in searching over the space of decision trees is the combinatorial complexity of the problem.Traditional algorithms address this by taking a greedy decision while splitting any node (e.g., splitting along feature with highest information gain) to grow the tree. As shown by Garey and Graham (1974), this can lead to sub-optimal solutions. The decision making problem could be solved better by maximizing a *long-term return* rather than a short term gain - which is a typical goal in RL.

* The authors contributed equally

Sometimes decision trees can have very few training points in their leaf nodes. This overfitting is usually avoided by pruning the tree. RLDT, however, does not have an explicit pruning stage.The RL formulation provides a neat way of striking a balance between pruning for generalization and growing deeper trees for higher accuracy.

We are interested in a learner that asks as few questions as possible about the features of the data point since data can be expensive. Furthermore, different features could have different costs associated with their extraction. We would want to selectively and efficiently examine values subject to these parameters. We will see that the reward formulation of the MDP can be naturally tuned to learn in this setup.

Our interest in studying an RL based approach is also motivated by the fact that online classification models face the challenge of concept drift, where the optimal tree changes with time. Online RL algorithms efficiently handle such situations with non-stationary optimal solutions.

Finally, we note that the above formulation is rich in that any of the vast variety of RL algorithms can be used to learn the optimal policy which forms our decision tree. Thus RLDT promises different techniques for online decision tree learning within a single framework.

The rest of the paper is organized as follows. In Section 2, we discuss related work in online decision tree learning. Section 3 formally defines the problem while Section 4 and 5 present the MDP formulation in RLDT. We discuss experiments in Section 6 after which we discuss possible directions for future work.

2. Earlier Work

Typically, online decision tree learning algorithms such as VFDT (Domingos and Hulten, 2000) and ID4 (Utgoff, 1989) try to incrementally learn the same tree that batch training algorithms like C4.5 learn. They examine all the features of the input data point to update their models. However RLDT uses a minimal subset of these features to both classify a new data point and to update its parameters.

Our work is closely related to that of Hoi et al. (2012) in which the authors study an online linear classification algorithm that examines a fixed number of features to classify an input point. As the authors note, the constraints of this algorithm are harder than that in sparse online learning (Duchi and Singer, 2009; Langford et al., 2009) that has no upper bound on the number of features examined. RLDT, in fact, imposes a harder combination of the constraints as the learner is encouraged to examine the fewest number of features, subject to an upper bound.

We must note how RLDT differs from earlier studies of RL-based decision tree learning. Bonet and Geffner (1998) model the process of learning decision trees as a *known* partially observable MDP where the states of the underlying MDP are the training sample points. The learner decides to query a feature/report a label based on its belief over these states; the belief encodes the knowledge about the point that has been acquired so far. Preda (2007) approach the problem by designing an MDP with states corresponding to every possible partition of the training data. The action defined at a state is any kind of test (a feature query) that can be conducted on one of the subsets in that partition of the sample points.

A serious drawback of the above approaches is that the solutions are very rigid with respect to the training data. The algorithms fail to work in an incremental setup because the state spaces change completely by the addition of a new point. RLDT on the other hand is different in that it uses a state space representation that allows learning from new data incrementally. While the algorithm presented by Hwang et al. (2006) uses a similar representation, it is designed for learning from an offline batch of data points and an extension to online learning is not obvious. Furthermore, Hwang et al. (2006) propose a reward parametrization that is not concerned with minimizing the number of queries.

3. Online Learning for Classification with Query Costs

We now formally define the problem of learning to classify online by querying on features selectively. We assume that the input space is $\mathcal{X} = \mathbb{Z}^d$ and the output space is a set of labels $\mathcal{Y} = \{1, 2, \ldots K\}$. We discuss an extension to continuous features in the next section. At every timestep $t = 1, 2, \ldots$, the environment draws an input $\boldsymbol{x}_t = (x_1, x_2, \ldots x_d) \in \mathcal{X}$ and output $y_t \in \mathcal{Y}$ according to an underlying distribution $\mathcal{D}$. The learner is initially not informed of any of the attributes of $\boldsymbol{x}_t$. Instead, the learner interacts with the environment and *actively* informs itself about the point in two ways. First, it can either ***query*** the environment about the value of a feature with index j – which is denoted as executing action $\mathcal{F}_j$. Secondly, the learner can ***report*** to the environment an answer $\hat{y}_t \in \mathcal{Y}$ which is denoted as action $R_{\hat{y}_t}$. The environment then reveals the true label y_t, provides a reward depending on whether $\hat{y}_t = y_t$ or not, and considers the episode (timestep) terminated. The objective of the learner involves a) achieving a good performance in classification and b) making few queries.

4. MDP Formulation

We formulate the above problem as a Markov Decision Process M such that each episode of the MDP processes a new input point. The state space $\mathcal{S}$ of the MDP consists of all possible states that encode partial information about the data point. This partial information consists of the values of the features that are considered to be known at that state. Thus, every state $s \in \mathcal{S}$ corresponds to a set of known features $f(s)$ which have a configuration unique to s. Clearly, for the initial state s_0 of the MDP, $f(s_0) = \phi$.

The action space $\mathcal{A}$ consists of the feature query actions $\mathcal{F}_i$ for $i = 1, 2, \ldots d$ and the report actions $\mathcal{R}_i$ for $i = 1, 2, \ldots K$. However, not all query actions are allowed on all states. Consider feature $j \in f(s)$. $\mathcal{F}_j$ is disallowed at s. Furthermore, on taking action $\mathcal{F}_i$ at this state ($i \notin f(s)$) a transition is made to another state s' that includes all partial information present in s and also the information about the value of feature i. Thus $f(s') = f(s) \cup \{i\}$. On the other hand, taking action $\mathcal{R}_k$ leads us to a state where the label of the point is known regardless of which report action was taken. While here we consider only queries that result in as many splits as the number of values the feature can take, note that one could also consider comparison queries that result in binary splits.

The rewards on the query actions $\mathcal{F}_i$ are negative and are proportional to the cost $-C_i$ of querying feature i of the point $\boldsymbol{x}_t$. The reward on the report action $\mathcal{R}_i$ is however

dependent on the actual label y_t and the reported label i. If $y_t = i$, the agent assumes that the environment reinforces it with a positive reward, and a negative reward otherwise.

In order to discourage the agent from asking many queries we have to set non-zero query costs ($C_i > 0$) *or/and* lower values of γ, the discount factor. While γ discourages querying independent of the feature being queried, query costs provide us more flexibility to encode the actual cost associated with accessing a feature.

5. Solving the MDP

We employ Q-learning to solve the MDP M formulated as above. We chose Q-learning (off-policy method) over SARSA (on-policy method) because we perform multiple path updates (described in Section 5.2). The details of the process is described below and experimental results follow.

5.1. The reward for classifications and misclassification

When the true label is revealed after a report action, the value estimate of the report action for the true label is updated with a target of R_+ while the report action for each of the other labels is updated with a target of R_-. At some state s, for a class k, assume that $\mathcal{D}$ is such that any point that is consistent with the partial information at that state belongs to class k with probability p. Thus, we know that $Q(s, \mathcal{R}_k)$ converges to the expected return of

$$pR_+ + (1-p)R_-.$$

Consider a query action $\mathcal{F}_i$ at s for which state t is one of the possible subsequent states. Given state t, it is likely that there exists a class k such that its density p_k in state s, is less than its density p'_k in t, since we have more information about the point in t. Thus, $p'_k > p_k$. Taking action $\mathcal{F}_i$ and reporting k in state t results in an expected return of

$$-C_i + \gamma(p'_k R_+ + (1-p'_k)R_-).$$

The amount by which this expected return differs from the expected reward on directly taking $\mathcal{R}_k$ at state s would be

$$-C_i + (\gamma p'_k - p_k)R_+ + (-1 + \gamma - \gamma p'_k + p_k)R_-.$$

The above quantity can be made both positive and negative depending on the parameters γ, C_i, R_+, R_-. This is precisely how the tradeoff between accuracy and number of queries is addressed in RLDT. Furthermore, carefully setting the values R_+ and R_- to be a function of the label being correctly/wrongly reported, we can enforce cost-sensitive learning, and handle class imbalance. Class imbalance can also be addressed by setting the value of α to be lower for the majority class thereby under-sampling the class.

5.2. Speeding up Convergence

We say that RLDT converges when it has seen sufficiently many points beyond which the algorithm's performance does not change in expectation as it has converged to the optimal policy/decision tree. In this section, we discuss two techniques that will help in faster convergence.

Multiple Path Updates The updates after every episode correspond to a specific order in which the queries were asked. This order corresponds to a single 'path' down the MDP. However, it is crucial to note that we could have made the same set of queries in any other order and effected these updates in other paths. We could have also taken shorter paths that made only a subset of these queries before reporting the label. As an example, in Figure 1, while the red edges correspond to the path that was taken, value updates can be performed on all the action-values that have been shown. These updates must be done exactly once per state-action pair to avoid bias. We will see that this makes convergence significantly faster. We note that to avoid enumerating the exponential number of paths, we could sample a constant number of random paths from the space of paths to perform the updates. This would still be speed up convergence when compared to the naive single path update.

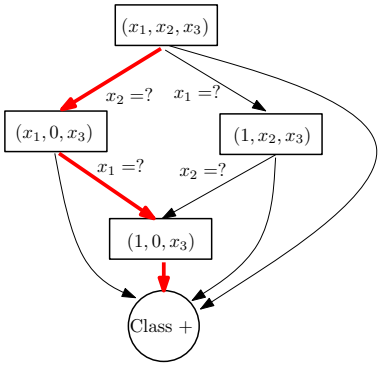


Figure 1: Paths along which updates must be made

This technique however results in more frequent updates in states where fewer features about the point are considered to be known i.e., when $|f(s)|$ is small. Thus the value of the best report actions at these states converge faster than possibly better query actions. Since ϵ-greedy exploration may not sufficiently explore these query actions, we use optimistic initial values for all the query actions.

Truncated Exploration Assume that we skip taking an exploratory report action $\mathcal{R}_i$ at a state s and choose to make a query instead. At the end of the episode we would always take a report action and know the true label of the input point. By knowing the true label, we would also know the outcome of action $\mathcal{R}_i$ that we skipped at state s. Thus, the report actions at s can be updated without exploring any of them. Hence, we restrict exploratory steps to only the query actions.

6. Experiments

Since it is sufficient to either have a low value of γ or a strictly positive query cost, we choose to set $\gamma = 0.8$ and all query costs zero. The report rewards are set as $R_+ = 5$ and $R_- = -5$. Apart from an exploration rate $\epsilon = 0.01$, we initialize all query action values to an optimistic value of 8 (an arbitrarily chosen value that is slightly higher than the maximum reward of 5). We allow the learner to make at most three queries in order

to truncate the state space of the MDP. We show that under this restriction RLDT can perform better than other algorithms.

Figure 2 shows the graphs corresponding to the performance of RLDT averaged across 50 runs on the Mushroom Data Set [1]. Note that it is the return, which is a function of both accuracy and the number of queries, that the agent tries to optimize. We see that the learner attains a high accuracy after examining at most 3 out of 22 features of the first 1000 points. Beyond this, the number of queries the agent makes drops to 1.1 queries per point in expectation while maintaining an accuracy of $97.74\% \pm \mathbf{3.01}$. This demonstrates that the discount factor discourages querying without affecting the accuracy undesirably. Note that all values are reported with their 95% confidence intervals.

On the Nursery Data Set [2] we only present the moving averages to observe the effect of the techniques discussed in Section 5.2. We see that multiple path updates speed up convergence significantly. Also, the *truncated exploration* technique results in many queries being asked on the first few points as we would expect. However, a significant difference in the convergence rate is not noticeable in this case. On this data set, we also show that increasing query cost reduces the number of queries made during learning (Figure 4a), although reducing accuracy from $92.53 \pm \mathbf{1.22}\%$ to $88.5 \pm \mathbf{0.88}\%$. We also observe that when the number of allowed queries is limited, RLDT can perform better than even a batch learning algorithm like C4.5 that learns from all features. On performing 5-fold cross validation, RLDT achieves an accuracy of $92.53 \pm \mathbf{1.22}\%$ with 2 feature queries per point in the testing phase. When C4.5 trees are restricted to a depth of 2 using Reduced Error Pruning, the accuracy on the same five folds turns out to be $85.33 \pm \mathbf{0.87}\%$.

We use the Electric Power Consumption Data Set (Harries and Wales, 1999) [3] to examine the performance of RLDT under concept drift. As seen in Figure 4b, a sufficient learning rate α is required to maintain a tree that adapts to a changing distribution. RLDT performs better than most popular online decision tree learning algorithms (Table 1), while making significantly fewer queries.

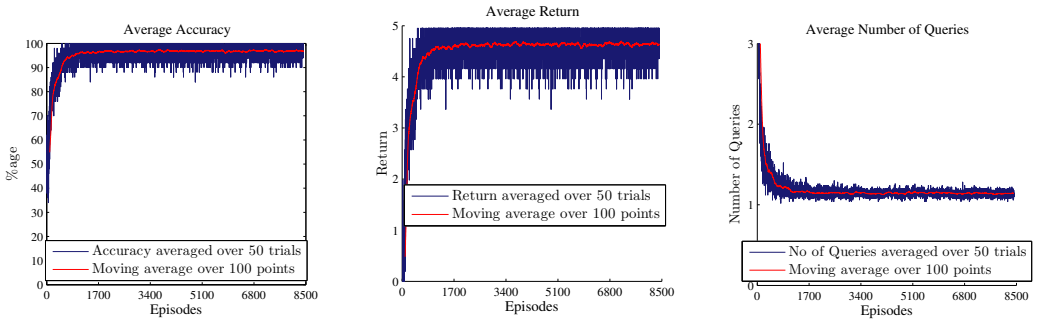
Method	Final Accuracy(%)	Mean Accuracy(%)
StARMiner Tree(ST)	81.3	78.93
Automatic StARMiner Tree(AST)	80.1	79.15
VFDT	75.8	74.64
VFDTcNB	77.3	77.16
RLDT	**83.26**	**81.15**

Table 1: Performance of standard algorithms on Electric Power Consumption Data Set

7. Future Extensions

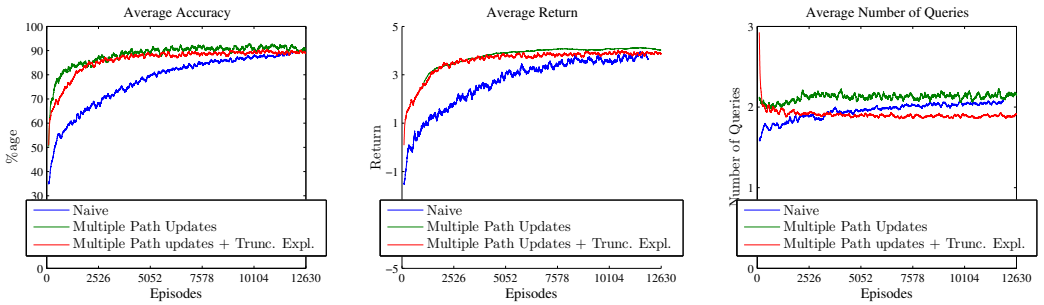
In this section, we discuss possible ideas for developing RLDT to work with high-dimensional and continuous feature spaces.

1. Available at https://archive.ics.uci.edu/ml/datasets/Mushroom
2. Available at https://archive.ics.uci.edu/ml/datasets/Nursery
3. Continuous attributes were discretized.



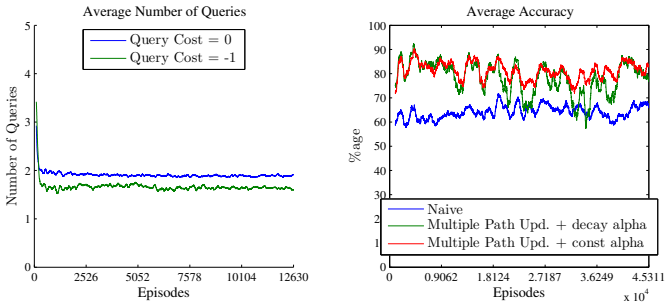
(*a*) Average Accuracy (*b*) Average Return (*c*) Average no. of Queries

Figure 2: Performance on Mushroom Data Set with Multiple Path Updates



(*a*) Average Accuracy (*b*) Average Return (*c*) Average no. of Queries

Figure 3: Performance on Nursery Data Set



(*a*) Varying Query Costs on Nursery Data Set (*b*) Average Accuracy on Electricity Data Set

Figure 4

Dealing with large state and action spaces The MDP corresponding to high dimensional data will have large state and action spaces. Limiting the number of queries is one solution to truncating a large state space. We could also dynamically prune the query ac-

tions at a state if we observe that the actions lead to 'equivalent' states where the best action is the same report action. The idea is to explicitly avoid exploration when not required.

On the other hand, we could use function approximation by encoding the states in terms of the features that are known/unknown at that state. However, efficient generalization can be done only with appropriate assumptions on the data distribution. For two states s and s', if neither of $f(s)$ and $f(s')$ is a subset of the other, the distribution in the part of the input space corresponding to s and s' may differ too significantly that we may not be able to draw experience from s to s' or vice versa. Nevertheless, we could still generalize if one of $f(s)$ or $f(s')$ is a subset of the other. That is, we can always draw experience from 'sub-states' where more feature values are known, to 'super-states' where less values are known. Two challenges arise here. First, we will have to determine how much weight experiences from different sub-states are given. Second, we would want to design a function approximation architecture that can (roughly) capture the overlapping experience between sub-states and super-states.

It is also important for the method to scale to large *action spaces* which might otherwise hinder techniques like Q-learning. Policy gradient versions of RLDT with appropriate policy parametrization to handle this would be an interesting direction for future work.

Continuous attribute values One way to handle continuous attribute values would be to discretize the feature space into finitely many bins. Alternatively, we could directly employ RL techniques that work with continuous action spaces. The value of splitting a continuous feature as a function of the split point is likely to be a continuous multimodal distribution which can be estimated online using a Gaussian Mixture Model (Agostini and Celaya, 2010).

Another alternative would be to maintain a finite number of split points per feature, each of which is dynamically updated as described by Chickering et al. (2001). As the allowed set of actions changes gradually, the value estimates for the current set of actions can be effectively used to estimate values of a new set of actions.

8. Conclusion

We have presented RLDT, an RL-based online algorithm that actively chooses to know very few features of a data point to *both* classify it and to learn a better model. The framework of the algorithm is rich enough to handle different settings such as class imbalance and non-uniformly expensive features using appropriate parameter values. Furthermore, since we can employ any RL algorithm to learn the optimal policy, RLDT promises multiple solution methods to online decision tree learning that can be explored in the future.

References

Alejandro Agostini and Enric Celaya. Reinforcement learning with a gaussian mixture model. In *International Joint Conference on Neural Networks, IJCNN 2010*, pages 1–8, 2010.

Blai Bonet and Hector Geffner. Learning sorting and decision trees with pomdps. In Jude W. Shavlik, editor, *Proceedings of the Fifteenth International Conference on Machine Learning (ICML 1998)*, pages 73–81, 1998.

David Maxwell Chickering, Christopher Meek, and Robert Rounthwaite. Efficient determination of dynamic split points in a decision tree. In *Proceedings of the 2001 IEEE International Conference on Data Mining,*, pages 91–98, 2001.

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*, KDD '00, pages 71–80, 2000.

John Duchi and Yoram Singer. Efficient online and batch learning using forward backward splitting. *J. Mach. Learn. Res.*, 10:2899–2934, December 2009.

M. R. Garey and R. L. Graham. Performance bounds on the splitting algorithm for binary testing. *Acta Inf.*, 3(4):347–355, December 1974. ISSN 0001-5903.

Michael Harries and New South Wales. Splice-2 comparative evaluation: Electricity pricing, 1999.

Steven CH Hoi, Jialei Wang, Peilin Zhao, and Rong Jin. Online feature selection for mining big data. In *Proceedings of the 1st international workshop on big data, streams and heterogeneous source mining: Algorithms, systems, programming models and applications*, pages 93–100. ACM, 2012.

Kao-Shing Hwang, Tsung-Wen Yang, and Chia-Ju Lin. Self organizing decision tree based on reinforcement learning and its application on state space partition. In *Proceedings of the IEEE International Conference on Systems, Man and Cybernetics, Taipei, Taiwan, October 8-11, 2006*, pages 5088–5093. IEEE, 2006.

John Langford, Lihong Li, and Tong Zhang. Sparse online learning via truncated gradient. *J. Mach. Learn. Res.*, 10:777–801, June 2009.

Mircea Preda. Adaptive building of decision trees by reinforcement learning. In *Proceedings of the 7th Conference on 7th WSEAS International Conference on Applied Informatics and Communications - Volume 7*, AIC'07, pages 34–39, 2007.

Paul E. Utgoff. Incremental induction of decision trees. *Mach. Learn.*, 4(2):161–186, November 1989.