Policy-gradient Methods for Decision Trees

Aurélia Léon and Ludovic Denoyer

Sorbonne Universités, UPMC Univ Paris 06, UMR 7606, LIP6, F-75005, Paris, France

Abstract. We propose a new type of decision trees able to learn at the same time how inputs fall in the tree and which predictions are associated to the leaves. The main advantage of this approach is to be based on the optimization of a global loss function instead of using heuristic-based greedy techniques, while keeping the good characteristics of decision trees. The learning algorithm is inspired by reinforcement learning and based on gradient-descent based methods, allowing a fast optimization. Moreover the algorithm is not limited to (mono-label) classification task and can be used for any predictive problem while a derivable loss function exist. Experimental results show the effectiveness of the method w.r.t baselines.

1 Introduction

Decision Trees are popular machine learning methods based on a divide-and-conquer strategy for prediction. They are usually learned with greedy algorithms based on different heuristics that aim at building the structure of the tree on a training set, resulting in sub-optimal learned models.

We propose a new family of decision trees called **Reinforced Decision Trees (RDTs)** which aims at keeping the advantages of decision trees – i.e fast inference speed, high performance – with the ability of being easily learned on different problems by using fast gradient-based descent approaches. The main idea of RDT is to consider the classification problem as a **sequential decision process** where a learned policy guides any input x in a tree structure from the root to one leaf. The learning algorithm is inspired from policy-gradient methods [1] which allows us to act on both the way an input x falls into the tree and on the predictions associated with the leaves of this tree. The difference with classical Decision Trees learning techniques is that our algorithm aims at directly solving a global (derivable) objective function without using heuristic-based greedy techniques. When comparing to classical classification techniques, our model benefits from the tree architecture which greatly reduces the computation complexity during prediction. A set of experiments over different datasets shows the efficiency of our approach.

2 Notations and Model

We consider the multi-class classification problem where each input $x \in \mathbb{R}^n$ has to be associated with one of the C possible categories[1]. Let us denote $y \in \mathbb{R}^C$ the label of x, such that $y_i = 1$ if x belongs to class i and $y_i = -1$ elsewhere. We will denote $\{(x^1, y^1), ..., (x^N, y^N)\}$ the set of N training examples.

[1]Note that the model naturally handles other predictive problems when a derivable loss function exists, and is not restricted to classification.

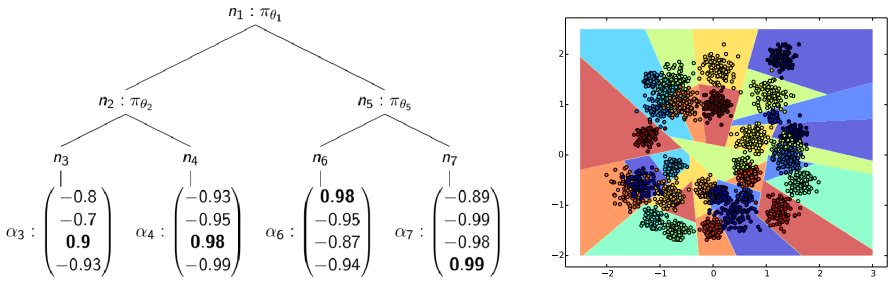

Fig. 1: (left) An example of RDT. The π_θ functions and the α values (the vectors in the leaves of the tree) have been learned in one integrated step following a policy-gradient based method. Bold values correspond to predicted categories at the leaves level. (right) Decision frontier learned by the model on a 2D dataset with 32 different categories.

2.1 Reinforced Decision Trees Architecture

The *Reinforced Decision Tree (RDT)* architecture shares common points with decision trees. Let us denote $\mathcal{T}_{\theta,\alpha}$ such a tree with parameters θ and α that will be defined later: (i) The tree is composed of a set of nodes denoted $nodes(\mathcal{T}_{\theta,\alpha}) = \{n_1, ..., n_T\}$ where T is the number of nodes of the tree. (ii) n_1 is the root of the tree. (iii) $parent(n_i) = n_j$ means that node n_j is the parent of n_i. Each node but n_1 has only one parent, n_1 has no parent since it is the root of the tree. (iv) $leaf(n_i) = true$ if and only if n_i is a leaf of the tree.

Note that we do not have any constraint concerning the number of leaves or the topology of the tree. Each node of the tree is associated with its own set of parameters: (i) A node n_i is associated with a set of parameters denoted θ_i if n_i is an internal node i.e $leaf(n_i) = false$. (ii) A node n_i is associated with a set of parameters denoted $\alpha_i \in \mathbb{R}^C$ if n_i is a leaf of the tree.

Parameters $\theta = \{\theta_i\}$ are the parameters of the policy that will guide an input x from the root node to one of the leaves of the tree. Parameters α_i correspond to the prediction that will be produced when an input reached the leaf n_i. Our architecture is thus very close to classical decision trees, the major difference being that the prediction associated with each leaf is a set of parameters that will be learned during the training process, allowing the model to choose how to match the leaves of the tree with the categories. The model will thus be able to simultaneously learn which path an input has to follow based on the θ_i parameters, but also how to organize the categories in the tree based on the α_i parameters. An example of RDT is given in Figure 1.

2.2 Inference Process

Let us denote H a trajectory, $H = (n_{(1)}, ..., n_{(t)})$ where $n_{(i)} \in nodes(\mathcal{T}_{\theta,\alpha})$ and (i) is the index of the i-th node of the trajectory. H is thus a sequence of nodes

where $n_{(1)} = n_1$, $\forall i > 1, n_{(i-1)} = parent(n_{(i)})$ and $leaf(n_{(t)}) = true$.

Each internal node n_i is associated with a function (or policy) π_{θ_i}: for a given input x, $\pi_{\theta_i}(x, n) = P(n|n_i, x)$ is the probability that x moves from the node n_i to the node n (with $P(n|n_i, x) = 0$ if $n \notin children(n_i)$). In our experiments, π_{θ_i} is a linear function followed by a soft-max. Note that the difference between RDT and DT is that the decision taken at each node is stochastic, which will allow us to use gradient-based learning algorithms. The probability of a trajectory $H = (n_{(1)}, ..., n_{(t)})$ given an input x can be written as:

$$P(H|x) = \prod_{i=1}^{t-1} \pi_{\theta_{(i)}}(x, n_{(i+1)})$$

Once a trajectory has been sampled, the prediction produced by the model depends on the leaf $n_{(t)}$ reached by x. The model directly outputs $\alpha_{(t)}$ as a prediction, $\alpha_{(t)}$ being a vector in $\mathbb{R}^C$ (one score per class) as explained before, also learned on the training set. The model thus produces one score for each possible category, but the inference complexity of this step is $\mathcal{O}(1)$ since it just corresponds to returning the value $\alpha_{(t)}$.

After training, the output is the vector $\alpha_{(t)}$ associated to the most probable leaf.

2.3 Learning Process

The goal of the learning procedure is to simultaneously learn both the *policy functions* π_{θ_i} and the *output parameters* α_i in order to minimize a given learning loss Δ which corresponds here to a classification loss (e.g square loss or hinge loss). Our learning algorithm is based on an extension of **policy gradient techniques** inspired from the Reinforcement Learning literature and similar to [2]. More precisely, our learning method is close to the methods proposed in [1] with the difference that, instead of considering a reward signal which is usual in reinforcement learning, we consider a loss function Δ. This function computes the quality of the system, providing a richer feedback information than simple rewards since it can be derivated, and thus gives the direction in which parameters have to be updated. The performance of our system is denoted $J(\theta, \alpha)$:

$$J(\theta, \alpha) = E_{P_\theta(x,H,y)}[\Delta(F_\alpha(x, H), y)]$$

where $F_\alpha(x, H)$ is the prediction made following trajectory H - i.e the sequence of nodes chosen by the π-functions. The optimization of J can be made by gradient-descent techniques and we need to compute the gradient of J:

$$\begin{aligned}
\nabla_{\theta,\alpha} J(\theta, \alpha) &= \int \nabla_{\theta,\alpha} \left(P_\theta(H|x) \Delta(F_\alpha(x, H), y) \right) P(x, y) dH dx dy \\
&= \int P_\theta(H|x) \nabla_{\theta,\alpha} \left(log P_\theta(H|x) \right) \Delta(F_\alpha(x, H), y) P(x, y) dH dx dy \\
&+ \int P_\theta(H|x) \nabla_{\theta,\alpha} \Delta(F_\alpha(x, H), y) P(x, y) dH dx dy
\end{aligned}$$

Using the Monte Carlo approximation of this expectation by taking M trail histories over the N training examples, and given that $\Delta(F_\alpha(x^i, H), y) = \Delta(\alpha_{(t)}, y)$, we obtain:

$$\nabla_{\theta,\alpha} J(\theta, \alpha) = \frac{1}{N}\frac{1}{M}\sum_{i=1}^{N}\sum_{k=1}^{M}\left[\sum_{j=1}^{t-1}\left(\nabla_{\theta,\alpha}\left(\log \pi_{\theta_{(j)}}(x^i)\right)\Delta(\alpha_{(t)}, y)\right) + \nabla_{\theta,\alpha}\Delta(\alpha_{(t)}, y)\right]$$

Intuitively, the gradient is composed of two terms: the first term aims at penalizing trajectories with high loss, and the second term is responsible about how to allocate the categories in the leaves of the tree. We use a classical stochastic gradient descent-based algorithm with mini-batches.

The complexity of the inference process is linear with the depth of the tree. Typically, in a multi-class classification problem, the depth of the tree can be proportional to $\log C$, resulting in a very high speed inference process. Note that different functions topologies can be used for π. In the following we have used **simple linear functions**, but more sophisticated ones can be tested like neural networks. Moreover, in our model, there is no constraints upon the α parameters, nor about the loss function Δ which only has to be derivable. It means that our model can also be used for other tasks like multivariate regression, or even for producing continuous outputs at a low price.

3 Experiments

In this Section, we provide a set of experiments that have been made on a toy dataset used for visualization, and real datasets. Our model has been compared with two baseline models: linear one-versus-all SVMs and Decision Trees. Hyper-parameters (learning rates, size of mini-batches and number of iterations) have been tuned on a validation set, and the results have been averaged on five different runs. For Decision Trees, we used the implementation in scikit-learn of an optimized version of the CART algorithm.

3.1 Results

A first set of experiments has been performed on different multi-class UCI datasets - see Table 1. One can see that the best performance is obtained with RDT and a depth of 10. Note that the inference complexity of RDT with depth D is $\mathcal{O}(D)$ which is lower or equals to the complexity obtained with one-versus-all SVM, resulting in faster models during inference. In comparison to classical decision trees, our model takes more time to train but obtains better results. We think that it is due to the fact that we optimize a global objective without using greedy-based learning techniques, and that the splits are linear and not necessarily parallel to axis. Figure 1 (right) illustrates the model obtained on a 2D dataset with 32 categories, and allows us to better understand the nature of the learned decision frontier. RDTs also outperform Decision Trees on this dataset.

	D	pendigits	optdigits	letter	mnist
#train samples		3745	1911	6661	50000
#val samples		3748	1908	6652	10000
#test samples		3497	1796	6687	10000
#features		16	64	16	784
#classes		10	10	26	10
Linear SVM		17.38 ± 3.62	6.84 ± 0.46	29.98 ± 0.01	8.47 ± 0.02
Decision Tree	5	22.61 ± 0.08	31.47 ± 0.13	63.24 ± 0.01	32.53 ± 0.01
	8	11.40 ± 0.20	18.89 ± 0.21	37.22 ± 0.08	18.44 ± 0.02
	10	9.99 ± 0.24	16.63 ± 0.35	30.73 ± 0.09	14.09 ± 0.06
	(*)	9.74 ± 0.35	16.00 ± 0.24	17.90 ± 0.14	12.87 ± 0.04
RDT	5	5.35 ± 1.41	7.24 ± 0.83	30.89 ± 1.13	7.73 ± 0.50
	8	4.45 ± 1.25	5.86 ± 0.26	15.81 ± 0.75	6.25 ± 0.57
	10	3.43 ± 0.34	5.35 ± 0.71	12.34 ± 0.44	5.41 ± 0.44

Table 1: Classification error (%) of different models and RDTs, where each node has exactly two children. D is the depth of the built trees. In the Decision Tree (*), nodes are expanded until all leaves are pure or until all leaves contain less than 2 samples.

4 Related Work

Hierarchical models have been developed for two cases: (i) a first one where a hierarchy of category is already known; in that case, the hierarchy of classifiers is mapped on the hierarchy of classes. (ii) A second approach closer to ours consists in automatically building a hierarchy from the training set. This is usually done in a preliminary step by using for example clustering techniques like spectral clustering on the confusion matrix [3], using probabilistic label tree [4] or even partitioning optimization [5]. Facing these approaches, RDT has the advantage to learn the hierarchy and the classifier in an integrated step only guided by a unique loss function. The closest works are [6] where a hierarchical mixture of experts is learned using EM algorithm, [7] that constructs globally optimal decision trees and can optimize existing decision trees, [8] which discovers the hierarchy using online learning algorithms, the construction of the tree being made during learning, or more recently [9] where the split functions and the leaf parameters of the decision tree are optimized together using stochastic gradient descent and a global objective. Other families of methods have been proposed like error-correcting codes [10, 11, 12], sparse coding [13] or even using representation learning techniques, representations of categories being obtained by unsupervised models [14, 3].

At last, the use of sequential learning models, inspired by reinforcement learning, in the context of classification or regression has been explored recently for different applications like features selection [15] or image classification [16, 17]. Our model belongs to this family of approaches.

5 Conclusion and Perspectives

We have presented *Reinforced Decision Trees* which is a learning model able to simultaneously learn how to allocate categories in a hierarchy and how to classify inputs. RDTs are sequential decision models where the prediction over one input can be made using $\mathcal{O}(\log C)$ classifiers, making this method suitable for problems with large number of categories. Moreover, the method can be easily adapted to any learning problem like regression or ranking, by changing the loss function. RDTs are learned by using a policy gradient-inspired method. Experimental results show the effectiveness of this approach. Future work mainly involves the use of this model for continuous outputs problems.

Acknowledgements

This work was performed within the Labex SMART supported by French state funds managed by the ANR within the Investissements d'Avenir programme under reference ANR-11-IDEX-0004-02.

References

[1] J. Baxter and P. Bartlett. Direct gradient-based reinforcement learning, 1999.

[2] Ludovic Denoyer and Patrick Gallinari. Deep sequential neural network. *arXiv preprint arXiv:1410.0510 - Workshop Deep Learning NIPS 2014*, 2014.

[3] Samy Bengio, J Weston, and D. Grangier. Label embedding trees for large multi-class tasks. *Adv. Neural Inf. Process. Syst.*, (1):163–171.

[4] Baoyuan Liu, Fereshteh Sadeghi, Marshall Tappen, Ohad Shamir, and Ce Liu. Probabilistic label trees for efficient large scale image classification. *Comput. Vis. Pattern Recognit.*, pages 843–850, 2013.

[5] Jason Weston, A Makadia, and Hector Yee. Label partitioning for sublinear ranking. *Int. Conf. Mach. Learn.*

[6] Michael I. Jordan and Robert A. Jacobs. Hierarchical mixtures of experts and the em algorithm. *Neural computation*, 6(2):181–214, 1994.

[7] Kristin P Bennett and J Blue. Optimal decision trees. *Rensselaer Polytechnic Institute Math Report*, 214, 1996.

[8] Anna Choromanska and John Langford. Logarithmic Time Online Multiclass prediction. pages 1–13, 2014.

[9] Mohammad Norouzi, Maxwell Collins, Matthew A Johnson, David J Fleet, and Pushmeet Kohli. Efficient non-greedy optimization of decision trees. In *Advances in Neural Information Processing Systems*, pages 1720–1728, 2015.

[10] TG Dietterich and Ghulum Bakiri. Solving multiclass learning problems via error-correcting output codes. *arXiv Prepr. cs/9501101.*

[11] RE Schapire. Using output codes to boost multiclass learning problems. *ICML*, (1):1–9.

[12] M Cissé, T Artières, and P Gallinari. Learning compact class codes for fast inference in large multi class classification. *Eur. Conf. Mach. Learn.*

[13] Bin Zhao and Eric P. Xing. Sparse output coding for large-scale visual recognition. *Comput. Vis. Pattern Recognit.*, pages 3350–3357, 2013.

[14] Kilian Weinberger and Olivier Chapelle. Large margin taxonomy embedding with an application to document categorization. *Adv. Neural Inf.*, pages 1–8.

[15] Gabriel Dulac-arnold, Ludovic Denoyer, Philippe Preux, and Patrick Gallinari. Datum-wise classification. A sequential Approach to sparsity. *ECML/PKDD*, pages 375–390.

[16] Gabriel Dulac-Arnold, Ludovic Denoyer, Nicolas Thome, Matthieu Cord, and Patrick Gallinari. Sequentially generated instance-dependent image representations for classification. *Internation Conference on Learning Representations - ICLR 2014*, 2014.

[17] Karol Gregor, Ivo Danihelka, Alex Graves, and Daan Wierstra. DRAW: A recurrent neural network for image generation. *CoRR*, abs/1502.04623, 2015.