Adaptive Non-Deterministic Decision Trees: General Formulation and Case Study

HEMERSON PISTORI[1]
JOÃO JOSÉ NETO[2]
MAURO CONTI PEREIRA[1]

[1]UCDB - Universidade Católica Dom Bosco
GPEC - Grupo de Pesquisa em Engenharia e Computação,
Av. Tamandaré, 6000 Jd. Seminário, CEP 79117-900, Campo Grande (MS), Brasil
(pistori,mauro)@ucdb.br
[2]USP - Universidade de São Paulo, Escola Politécnica
LTA - Laboratório de Linguagens e Técnicas Adaptativas
Av. Prof. Luciano Gualberto. Trav. 3 N. 158, CEP 05508-900, São Paulo (SP), Brasil
joao.jose@poli.usp.br

Abstract. This paper introduces the adaptive non-deterministic decision tree, a formal device derived from adaptive device theory. ANDD-tree is a new framework for the development of supervised learning techniques. The general formulation of this framework, a case study and some experimental results are also presented.

Keywords: Rule-driven Adaptive Devices, Machine Learning, Decision Trees

(Received July 21, 2005 / Accepted January 06, 2006)

1 Introduction

Decision trees are widely used as a knowledge representation tool in machine learning, mainly for their clean graphical representation, which captures, some aspects of the human decision process. Algorithms for inducing decision trees from examples, a major problem in machine learning, include ID3, C4.5 [19], CART [3] and ITI [14], the last one being an incremental inducer.

Rule-driven adaptive devices [11], RDAD for short, were first introduced as a formal framework to solve some important problems in the domain of compiler construction, in the late 80's [10]. The scope of this theory has grown since then, embracing works in areas as diverse as grammatical inference [12], automatic music composition [1], natural language processing [13], robotics [4], natural computing [15] and computer vision [16]. A RDAD is composed of a subjacent non-adaptive formalism and an adaptive mechanism. The subjacent mechanism is usually characterized by a well known formalism with a static structure defined by a set of rules. The adaptive mechanism consists of an additional set of rules, called meta-rules, that are attached to the subjacent rules and that act on the structure of the subjacent mechanism by querying, inserting and removing subjacent rules along its operation. For instance, if the subjacent mechanism is a finite state automaton (FSA), its adaptive version, the adaptive finite state automaton [15], is an automaton that allows their transitions to be removed or new transitions to be created while accepting an input string. Some other research, related to RDAD, include that of Shutt and Rubinstein [21], Klein and Kutrib [8] and Jackson [7].

In order to apply RDAD techniques in the context of decision tree induction, a new formalization for decision trees was created and denominated non-deterministic decision trees, or NDD-trees. The extended device, an adaptive non-deterministic decision tree became a framework for the development of decision tree induction algorithms, based on adaptive technology. This new approach offers a new formal tool for the development of distributed and incremental learning strategies, based on decision trees. This paper presents such framework, together with a case study, the adaptive prefix tree, whose learning performance is comparable to some state-of-the-art inducers. Non-deterministic decision trees are described in the next section. Afterwards, their adaptive versions are introduced. Section 5 reports some experimental results. Finally, conclusions and future works are presented.

2 Non-Deterministic Decision Tree

In a decision tree, decisions, or classes (for classification problems), are represented as the leaves of the tree, whereas internal nodes carry discrete tests. The results of these tests, when applied to the problem at hand, determine the path to be followed when the tree is traversed from its root. When a path from the root to any leaf does not exist, due to missing or inconsistent information, NDD-trees proceed, non-deterministically, using all possible outgoing paths from a node whose test result is missing, or some other suitable path when the result of the test is not represented in any of the outgoing paths.

When working non-deterministically, decision trees may give more than a single answer to a question. In such cases, the final answer may be given as a majority vote amongst the occurrences of each possible answer. A NDD-tree is formally defined as a 6-tuple $T = (I, \Sigma, \Gamma, c, A, R)$ where:

I, Σ, Γ are non-empty sets representing, respectively: instances, attributes and values. Both Γ and Σ are finite sets, whereas I may be countably infinite. The Γ set always contains the *unknown value* symbol "**?**", that represents missing, unknown or non-existent information.

$c \in \Sigma$ defines the *class attribute*, whose values may appear exclusively at the decision tree leaves.

$A : I \times \Sigma \rightarrow \Gamma$ describes the instances, by associating values to each instance attribute. An instance whose class attribute is mapped into the unknown value, $A(i, c) =$ **?**, is called a *training instance*; otherwise, it is called a *test instance*.

R is the set of *rules*, inductively defined as:

Leaf 2-tuple (id, v), where $v \in f(c)$ is a class label and id is a rule identifier, or

Internal (n+2)-tuple $(id, a, (v_1, R_1), ..., (v_n, R_n))$, where id is an identifier, $a \in \Sigma - \{c\}$ is an attribute and $v_i \in f(a), 1 \leq i \leq n$ are attribute values. All elements $R_i, 1 \leq i \leq n$, are also rules.

The rule that contains all other rules, corresponding to the top of this hierarchy, is called the $root$. The space of configurations, C, of a NDD-tree, is a set of ordered pairs (τ, ι), where $\tau \in R$ is a rule and $\iota \in I$ is an instance. Initial configurations, $c_0 \in C$, are characterized by the root coupled with some instance $\iota \in I$. Final configurations are all configurations that have leaves as the first element of the pair. The stream of input stimuli of a NDD-tree consists of instances, whereas the stream of output stimuli is composed of class values. The operation of a NDD-tree, T, proceeds as long as there are instances in the input stream, as follows:

1. Given a non-final configuration (τ', ι), with $\tau' = (id, a, (v_1, R_1), ..., (v_n, R_n))$, we say that T *reaches in one step* the configuration (τ'', ι), denoted $(\tau', \iota) \vdash (\tau'', \iota)$, if, and only if, one of these happens:

   Deterministic Step: $A(\iota, a) = v_i$ and $\tau'' = R_i$, for some $1 \leq i \leq n$.

   Non-deterministic Step: $\neg \exists i | A(\iota, a) = v_i$, for $1 \leq i \leq n$ and $\tau'' = R_j$, for some $1 \leq j \leq n$. If there is no path for a given attribute value then follow all available paths. In other words, handle this attribute as if it were a missing value.

2. When a final configuration, $((id, v), \iota)$, $v \in f(c)$, is reached, v (or the majority voted class, in case of non-determinism) is appended to the output stream. If there are new instances to be read from the input stream, the machine is set to the initial configuration (reading the next example), and the algorithm is restarted.

The overall approach to handle missing value is very similar to the one proposed by Quinlan [17, 18], the difference being only in the way that the concepts are formalized. This new formalization seeks to define decision trees as rule-driven devices, an essential step in the application of adaptive technology [11].

3 Adaptive NDD-tree

The application of adaptive technology, using NDD-trees as the subjacent mechanism, results in the adaptive device called adaptive non-deterministic decision tree, or ANDD-tree, for short. The adaptive layer improves the subjacent mechanism by allowing it to change its structure R (hierarchical set of rules) just before or after a configuration change happens in the subjacent layer. All changes are accomplished through the execution of a set of *elementary adaptive actions*, used to query, insert or remove rules. Each set of elementary adaptive actions is called an *adaptive function*, which can be attached to any rule, by appending *adaptive function calls* to it. Adaptive function calls may appear in many rules, but each rule is allowed to carry at most two adaptive function calls, one to be executed before and the second, after the rule is applied.

An ANDD-tree is a duple $AT = (SM, AM)$, where SM, the subjacent mechanism, is a NDD-tree $T =$

$(I, \Sigma, \Gamma, c, A, R)$ with a slight change in its R structure, which now accommodates optional adaptive function calls attached to rules; and AM, the adaptive mechanism, is a set of adaptive functions. The operation of an ANDD-tree starts following that of the subjacent mechanism, described in section 2, until an adaptive function is called. In this circumstance, structure R is modified according to the elementary actions specified, and the execution of the subjacent mechanism proceeds. The overall mechanism stops its execution whenever the subjacent rules becomes inoperable, for instance, when an adaptive function removes all rules, or when the input stream is empty (no more instances).

4 Case Study: Adaptive Prefix Trees

An adaptive prefix tree is a special kind of ANDD-tree that grows incrementally, driven by training examples, optionally interleaved with testing examples. Unlike most usual decision tree induction algorithms, adaptive prefix tree does not try to build a minimum-size decision tree. Instead, it keeps all training instances in a prefix-tree-like structure, and therefore, may also be considered as an instance-based learning approach. The non-deterministic nature of the underlying device gives the whole mechanism the ability to generalize beyond training cases.

The adaptive functions in an adaptive prefix tree impose a complete order on the attribute set Σ. The structure R consists, initially, of a single leaf containing the missing value. As training instances are processed, R grows like a prefix-tree for strings of the same size, with each string position corresponding to some attribute. The maximum depth of an adaptive prefix tree is $|\Sigma|$. Figure 1 shows the evolution of an adaptive prefix tree for a hypothetical machine learning problem and some arbitrarily chosen training instances. The problem has four binary (yes-or-no) attributes, $a_1, ..., a_4$, the last one being a class attribute. In this graphic representation, left and right child edges correspond, respectively, to the attribute values yes and no. Adaptive function calls are indicated in brackets. Strings of Y's and N's represent the attribute values of a specific training instance.

Thus, an adaptive prefix tree is an ANDD-tree $TA = ((I, \Sigma, \Gamma, c, A, R), \Theta)$ with some restrictions, namely: (1) the set of attributes, Σ, follows some arbitrary order, thus it may be expressed by a sequence, $a_1, a_2, ..., a_j$, where $j = |\Sigma|$, (2) the class attribute is a_j, (3) $R =$ **[**A_1**]** $(R_1, ?)$ and finally, (4) Θ contains j adaptive functions, $A_1, A_2, ..., A_j$, one for each attribute. All adaptive functions A_i, $1 \leq i < j$, are parameterless and contain just one elementary insertion action that appropriately adds a new node to the tree.

Datasets	Attributes		Sample Size	Miss
	Contin.	Discrete		
audiology	0	69	226	0.06%
autos	15	10	205	1%
breast	0	9	286	0.3%
diabetes	8	0	768	none
german	7	13	1000	none
heart	13	0	270	none
hepatitis	6	13	150	5%
horse-colic	7	20	368	18.7%
ionosphere	34	0	341	none
iris	4	0	150	none
mushroom	0	22	8124	1%
sick	7	22	3772	5.4%
vehicle	18	0	946	none
vote	0	16	435	3%
vowel	10	3	990	none

Table 1: Datasets from UCI used in the adaptive prefi x tree's experiments.

5 Experiments and Results

In order to compare the adaptive prefix tree performance on a broader set of benchmarks, some issues related to continuous value, missing and inconsistent data should be handled. A pre-processing phase discretizes continuous data using the Fayyad and Irani's discretization method [6], and substitutes missing values by mode and mean estimations. The inherent non-determinism of the adaptive prefix tree's underlying mechanism, the NDD-tree, handles the problem of inconsistent samples in a straightforward manner by simply choosing all available paths, and using majority vote to decide.

The adaptive prefix tree has been implemented and experimented using Weka[1] (Waikato Environment for Knowledge Analysis) [22]. Table 2 shows the average accuracy (correct classification rate) of the adaptive prefix tree compared to Naive Bayes [5], 3-NN (k-Nearest Neighborhood with k = 3), C4.5 [20] and ID3 [9], using the Weka default values [22] for parameters specific to each classifier. The average accuracy has been taken from a 10-Fold Stratified Cross-Validation experiment. Plus and minus signals indicate, respectively, better and worse performance of the adaptive prefix tree compared to each of the other techniques, based on a T-Test statistic in the 95% confidence level. The cases where adaptive prefix trees presented a statistically superior performance are shown in bold font. Datasets were taken from

[1]Weka is a free open source software available at http://www.cs.waikato.ac.nz/˜ml/weka/

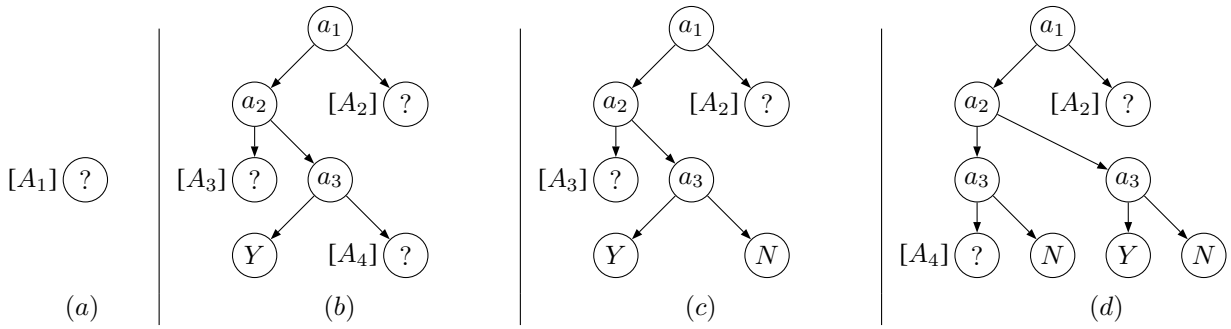

Figure 1: Evolution of an Adaptive Prefi x Tree
(a) Initial Configuration (b) After YNYY (c) After YNNN (d) After YYNN

the UCI repository and detailed description for each one of them can be found in [2]. Table 1 summarizes some important information about theses datasets.

The experiments indicate that this simple RDAD, with a global discretization method and very simple adaptive functions (just one elementary adaptive action), had a competitive performance in several datasets. For 13 out of 15 datasets, adaptive prefix tree presented the same or higher accuracy than Naive-Bayes. Comparison with instance-based approach (KNN) is balanced: no significant difference on 11 datasets , but with a remarkable difference in favor of adaptive prefix tree on the audiology and auto datasets. The C4.5 classifier performed better on 4 datasets, all of them seem to benefit from the local discretization method inherent to this algorithm (in the case of adaptive prefix tree, the discretization is performed globally, before the learning phase). Adaptive prefix tree also performed well on the four datasets with only nominal attributes: audiology, breast, mushroom, vote. This helps to dismiss any claim that its accuracy is solely dependent of the supervised discretization method applied.

ID3 has been considered here just because it is the only tested method that does not have a built-in mechanism to deal with continuous features, just like the adaptive prefix tree. This makes the comparison a bit more fair, and as can be seen in table 2, adaptive prefix tree shows significantly better results on almost half of the datasets and lower accuracy on only 3 of them. In order to apply ID3, the datasets were first discretized, using the same method employed with adaptive prefix tree.

6 Conclusion

A new algorithm for learning classifiers, represented as non-deterministic decision trees, has been introduced, and experimental results indicating that this algorithm performs well when compared to other machine learning algorithms were presented. Nonetheless, the main contribution of this work is not the algorithm, but a new formal framework, based on adaptive devices, that represents the learning algorithm as a self-modifying process. The formalism clearly distinguishes the subjacent, essentially static level (e.g. a non-deterministic decision tree), from the adaptive level, where structural changes on the underlying level are predicted to happen as external information is gathered.

Another important feature of an adaptive device is that the learning process, which occurs only inside adaptive functions, may be distributed among different parts of the underlying structure. One would have, for instance, an interactive decision tree which allows a specialist to indicate groups of nodes in which learning should not happen. A prototype of such environment is under development. Suggestions for future works include the development of new ANDD-trees and the search for new ways to integrate prefix-tree instance-based learning and decision tree learning.

Acknowledgments

This work has been funded by the Dom Bosco Catholic University, UCDB, the Foundation of Teaching, Science and Technology Development of Mato Grosso do Sul State, FUNDECT, and the Brazilian's Studies and Projects Funding Body, FINEP.

References

[1] Basseto, B. A. and Neto, J. J. A stochastic musical composer based on adaptative algorithms. In *Anais do XIX Congresso Nacional da Sociedade Brasileira de Computação. SBC-99.*, volume 3, pages 105–130, PUC-RIO, Rio de Janeiro, Brazil, July 1999.

Table 2: Accuracies using Adaptive Prefi x Tree (APT), NaiveBayes, KNN (K=3), C4.5 and ID3

Dataset	APT	NaiveBayes	KNN	C4.5	ID3
audiology	69.35±7.17	**65.58±5.38 +**	53.12±7.85 -	74.03±5.37	77.91±5.3 -
autos	75.43±6.02	**56.57±4.91 +**	**55 ±4.22 +**	**70.86±7.6 +**	80.02±8.8 -
breast	71.71±6.9	74.06±9.83	73.42±6.77	75.18±6.37	**57.67±7.71 +**
diabetes	73.05±4.9	75.78±3.61	74.74±5.73	74.09±4.55	**60.93±4.85 +**
german	70.3±5.52	74.9±3.9 -	74.2±3.43 -	69.7±4.47	**60.4±5.52 +**
heart	78.89±7.21	85.19±7.2 -	78.89±8.74	77.78±8.73	**62.96±9.56 +**
hepatitis	77.5±8.94	83.79±7.08	82.54±7.44	79.42±7.91	**73±8 +**
horse-colic	68.75±3.84	66.59±5.34	66.04±7.75	66.31±1.29	**10.33±3.35 +**
ionosphere	88.02±5.2	**82.36±6.04 +**	85.18±5.19	90.9±5.31	**83.78±6.67 +**
iris	92.67±4.92	96 ±4.66	95.33±5.49	95.33±4.5 -	92±6.13
mushroom	100 ±0	**95.38±0.37 +**	99.96±0.06	100 ±0	100±0
sick	97.67±0.65	**92.74±1.57 +**	**96.21±0.57 +**	98.65±0.54 -	97.77±0.61
vehicle	65.72±3.99	**44.45±4.38 +**	69.15±5.2	73.39±4.43 -	62.17±4.49
vote	93.92±1.72	**90.2±1.02 +**	93.24±1.42	95.88±1.25 -	93.56±3.21
vowel	74.78±2.44	**62.11±3.43 +**	76.65±2.27	76.44±0.89	78.99±4.66 -

[2] Blake, C. and Merz, C. UCI - repository of machine learning databases. *University of California, Irvine, Dept. of Information and Computer Sciences*, 1998.

[3] Breiman, L., Friedman, J., Olshen, R., and Stone, C. *Classification and Regression Trees*. Wadsworth and Brooks, Monterey, CA, 1984.

[4] Costa, E. R., Hirakawa, A. R., and Neto, J. J. An adaptive alternative for syntactic pattern recognition. In *Proceeding of 3rd International Symposium on Robotics and Automation, ISRA*, pages 409–413, Toluca, Mexico, September 2002.

[5] Domingos, P. and Pazzani, M. J. Beyond independence: Conditions for the optimality of the simple bayesian classifier. In *International Conference on Machine Learning*, pages 105–112, 1996.

[6] Fayyad, U. M. and Irani, K. B. Multi-interval discretization of continuous-valued attributes for classification learning. In *Proceedings of the Thirteenth International Joint Conference on Artificial Intelligence*, pages 1022–1027, San Mateo, CA, 1993. Morgan Kaufmann.

[7] Jackson, Q. T. Adaptive predicates in natural language parsing. *Perfection*, (4), 2000.

[8] Klein, A. and Kutrib, M. Self-assembling finite automata. Technical report, Institut für Informatik, Giessen, Germany, January 2002.

[9] Mitchell, T. M. *Machine Learning*. McGraw-Hill, 1997.

[10] Neto, J. J. Uma solução adaptativa para reconhecedores sintáticos. Technical report, PCS-POLI-USP, São Paulo, Brasil, 1988.

[11] Neto, J. J. Adaptative rule-driven devices - general formulation anda case study. In *CIAA'2001 Sixth International Conference on Implementation and Application of Automata*, pages 234–250, Pretoria, South Africa, July 2001.

[12] Neto, J. J. and Iwai, M. K. Adaptive automata for syntax learning. In *Anais da XXIV Conferencia Latinoamericana de Informática - CLEI 98*, pages 135–149, Quito, Equador, 1998.

[13] Neto, J. J. and Moraes, M. d. Using adaptive formalisms to describe context-dependencies in natural language. *Lecture Notes in Artificial Intelligence. N.J. Mamede, J. Baptista, I. Trancoso, M. das Graças, V. Nunes (Eds.): Computational Processing of the Portuguese Language 6th International Workshop, PROPOR 2003*, 2721:94–97, June 2003.

[14] Paul E. Utgoff, N. C. B. and Clouse, J. A. Decision tree induction based on efficient tree restructuring. *Machine Learning*, 29(1):5–44, October 1997.

[15] Pistori, H., Martins, P. S., and Jr., A. A. C. Adaptive finite state automata and genetic algo-

rithms: Merging individual adaptation and population evolution. In *Proceed. Int. Conf. on Adaptive and Natural Computing Algorithms - ICANNGA 2005*, Coimbra, Portugal, March, 21-23 2005.

[16] Pistori, H. and Neto, J. J. An experiment on handshape sign recognition using adaptive technology: Preliminary results. *Lecture Notes in Artificial Intelligence. XVII Brazilian Symposium on Artificial Intelligence - SBIA'04*, 3171, September 29 2004.

[17] Quinlan, J. R. Decision trees as probabilistic classifiers. In *Proceedings of the Fourth International Workshop on Machine Learning*, pages 31–37, Irvine, CA, 1987. Morgan Kaufmann Publishers.

[18] Quinlan, J. R. Unknown attribute values in induction. In Segre, A., editor, *Proceedings of the Sixth International Machine Learning Workshop*, New York, USA, 1989. Morgan Kaufmann Publishers.

[19] Quinlan, J. R. *C4.5 Programs for Machine Learning*. Morgan Kaufmann, 1993.

[20] Quinlan, J. R. Learning decision tree classifiers. *ACM Computing Surveys*, 28(1), 1996.

[21] Rubinstein, R. S. and Shutt, J. N. Self-modifying finite automata: An introduction. *Information Processing Letters*, 56(4):185–190, 1995.

[22] Witten, I. H. and Frank, E. *Data Mining: Practical Machine Learning Tools and Techniques with Java Implementations*. Morgan Kaufmann, 2000.