# Decision Tree Induction using Adaptive FSA

**Hemerson Pistori**
Universidade Católica Dom Bosco, Depto. Engenharia de Computação,
Campo Grande, Brasil, 79117-900
Universidade de São Paulo, Escola Politécnica,
São Paulo, Brasil, 05508-900
pistori@ec.ucdb.br

and

**João José Neto**
Universidade de São Paulo, Escola Politécnica,
São Paulo, Brasil, 05508-900
joao.jose@poli.usp.br

**Abstract**

This paper introduces a new algorithm for the induction of decision trees, based on adaptive techniques. One of the main feature of this algorithm is the application of automata theory to formalize the problem of decision tree induction and the use of a hybrid approach, which integrates both syntactical and statistical strategies. Some experimental results are also presented indicating that the adaptive approach is useful in the construction of efficient learning algorithms.

**keywords** Machine Learning, Decision Tree Induction, Adaptive Automata.

## 1 Introduction

The induction of decision trees from attribute vectors is an important and fairly explored machine learning paradigm [23]. The majority of the algorithms aiming to solve this problem, like ID3 and C4.5 [34, 35] works in two different phases: a training phase, where the decision tree is built from the available instances, and the testing, or performing phase, where new instances may be classified using the just constructed model. In general, the decision tree is built in a top-down style, using a greedy strategy to choose, based on the instances corresponding to the sub-tree in construction, the root of this sub-tree. Most researches in this area concentrate on the search of new methods to compare attributes and to determine the point where the top-down construction must stop (the pruning problem). At least two algorithms, ID5 and ITI [31], based on ID3 and C4.5, respectively, provide incremental capabilities.

In general, most operational machine learning systems currently used, including the ones based on neural networks and genetic programming, are not incremental. However, many applications do require, or would benefit from, incremental learning, as for instance, hand-written and speech recognition, where the final user could incrementaly improve the system performance, by inserting new training instances. Clearly, any machine learning algorithm could be made "incremental" by

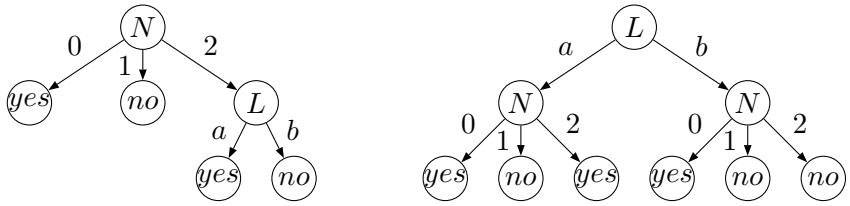


Figure 1: Two decision trees representing the same function $f$

rebuilding all the learned model as a new training instance is available, however this solution is highly inefficient and impractical.

This paper presents a new formalization and strategy to solve the decision tree induction problem, using concepts from automata theory and adaptive technology [28]. This approach promotes the use of syntactical techniques, in addition to the statistical ones, aiming at the construction of less opaque, or easier understandable, models. The adaptive technique, detailed in section 3, facilitates the specification of devices whose internal structures tends to change over time, as new external stimuli are provided. Besides, the technique boosts that re-utilization of well-known, static formalisms. Adaptree is a decision tree induction algorithm specified as an adaptive finite state automata, extended with some non-syntactical characteristics to handle continuous features and statistical generalization. Adaptree can be incrementaly trained to solve classification problems.

The reminder of this work is organized as follow: in the next section decision trees and induction algorithms are reviewed. Then, the adaptive theory is presented, as well as one of its particular cases: the adaptive finite state automaton. The algorithm constructed using this new technique and some experimental results appear in section 4 and 5, respectively. At last, conclusions and suggestions for future work are related.

## 2 Induction of Decision Trees

Decision trees are a way to hierachicaly represent discrete functions over multiple variables. This hierarchical characteristic makes decision trees very suitable for human inspection and utilization. The internal nodes of a decision tree represent tests to be applied over some variable $X$. Departing from each internal node are edges that represent possible values for $X$. The function range is represented in the tree's leaves. In order to calculate the function value using a decision tree, one must find a path from the root to a leaf, using the variable values to choose from different descending edges. The function value will be on this leaf. For example, any of the decision trees shown in figure 1 may be used to represent the function, $f : N \times L \rightarrow C$, depicted in table 1, where $N = \{0, 1, 2\}$, $L = \{a, b\}$ and $C = \{yes, no\}$.

| N | L | f(N,L) |
|---|---|---|
| 0 | a | yes |
| 0 | b | yes |
| 1 | a | no |
| 1 | b | no |
| 2 | a | yes |
| 2 | b | no |

Table 1: Function $f$

In the case of continuous variable, the tests in the edges are usually represented as simple intervals, like $X \leq 2.23$ and $X \geq 5.6$, with a discretization step being conducted during, or just before, the learning phase. The handling of continuous values in the functions range (tree's leaves) requires much more elaborated strategies, like the regression [33], model [18] and hybrid [40] trees. In the regression trees, the leaves simply keep the average function value, given the restriction represented in the internal nodes. The model trees generalizes the regression trees by admiting the existence of any linear classifier in each leaf. Hybrid trees are even more general, as they associate each leaf to an artificial neural network. This work concentrates on the still more popular discrete

decision trees.

### 2.1 Algorithms for the Induction of Decision Trees

An important problem in machine learning is the construction of a decision tree to represent an unknown function $f$, given a usually not proper subset $T$ of $f$. The $f$ function is denominated, in the machine learning jargon, the *target concept*, while the decision tree resulting from the application of an induction algorithm is named an, possible not accurate, *target concept representation*. Each element of the $T$ set, or *training set*, is a *training instance* or an *attribute vector*. In the non-incremental approaches, the training set and the target concept representation are, respectively, the input and output of the induction algorithm.

Most decision tree induction (DTI) algorithms follow the same overall strategy of building the tree from the root, partitioning the training set according to the tests chosen for that node, and recursively building a subtree for each test value. The partition used to induce any new subtree gets smaller at each recursion, leading to worse induction estimates for decisions near the leaves. This kind of problem is frequent in algorithms based on local optimization, which is not the case of our proposal.

The first distinguishing feature of DTI algorithms is the way that attributes (variables) are compared when choosing a subtree root. Most comparison approaches, like information gain [34] and $\chi$-square statistic [24], aim at increasing the chances of constructing the smaller (in number of nodes) decision tree that correctly represents the available instances. The preference for smaller trees are based on the *Occam Razor* [16] principle, which suggests that between two equally satisfactory explanations, the most simple should be chosen. A systematic analysis of this principle, that seems to permeate the process of scientific discovery, can be found in [16].

Another well explored feature of a DTI algorithm is the criteria used to decide when the decision tree should stop growing and branching: the *pruning* strategy [17]. Decision trees that perfectly adhere to the training set can suffer from the *overfitting* problem: when the learned model do not generalize well to the testing set. The pruning can happen during the growing phase, using some heuristic metric to prevent the tree from growing beyond a predetermined level, or by letting the tree grow to its maximum height and then "post-pruning" it, usually basing the decision on the classification performance measured over an independent instance set [23]. An interesting study showing that pruning is just another kind of generalization bias and that may, in fact, be prejudicial to some group of problems, can be found in [36, 37]. The algorithm presented in this paper is an example of how an unpruned decision tree may ensue good performances on well known benchmark datasets.

## 3 Rule-Driven Adaptive Devices

A recurrent question in computer science is how to balance expressiveness and clearness when proposing a new representation scheme for algorithms and problems. A Turing machine, for instance, is a very expressive formalism, however, it can hardly be used in the solution of real problems as its application is, at least, uncomfortable. Finite state automata, on the other hand, are easy to use, but lacks the expressivenes of the Turing machines.

The emerging technique called rule-driven adaptive devices [27, 28] approaches the problem of expressiveness versus clearness by empowering a simple, but not much expressive subjacent device, as a finite state automata, with an adaptive layer. This adaptive layer, which preserves much of the subjacent mechanism syntax and semantics, consists of deletion and insertion actions that operates on the subjacent mechanism rules (e.g. transitions for automata and productions for grammars)

and may be started during the subjacent mechanism operation. In this way, the initial static structure of the subjacent device becomes dynamic, but with the very desirable property that this dynamism is, in great extent, expressed in terms of the subjacent formalism. Adaptive technique is already being applied in the solution of problems in areas as diverse as grammatical inference [29], automatic music composition [3], natural language processing [8], robotics [22, 6] and computer vision [32].

### 3.1 Adaptive Finite State Automata

An adaptive finite state automaton ($\mathscr{A}$-FSA ) is an adaptive device that extends the expressive power of the well-known finite state automaton. Informally, an $\mathscr{A}$-FSA is just an FSA that can change its transition and state set during input reading. It is important to note that an $\mathscr{A}$-FSA is not an adaptive automata [26], as the subjacent device of the later consists of the slightly more complex, structured pushdown automata [30]. This simpler device, however, preserves the Turing-powerful expressiveness of adaptive automata: the proof of expressiveness for adaptive automata may be trivially adapted to finite state automata [21], as none of the extensions introduced by the structured pushdown automata formalism, in relation to the finite state automata, is used in the proof. Besides using a simpler subjacent device, the present work also proposes some simplifications to the adaptive layer, which may now be fully formalized in set theoretic and algebraic terms. Formally, $\mathscr{A}$-FSA are 8-uples $\mathrm{M} = \langle Q, \Sigma, q_0, F, \delta, Q_\infty, \Gamma, \Pi \rangle$ where the first five elements refer to the non-adaptive formal mechanism (an FSA):

$Q \subseteq Q_\infty$ is a finite subset of a possible infinite state set.

$\Sigma$ is the input alphabet, finite and non-empty.

$q_0 \in Q$ is the initial state of the automaton.

$F \subseteq Q$ is the final state set.

$\delta \subseteq (Q_\infty \times (\Sigma \cup \{\epsilon\}) \times Q_\infty)$ is the transition relation.

The last three elements of M, representing the adaptive layer, are defined as follow:

$Q_\infty$ is a possibly infinite state set.

$\Gamma \subseteq (\{+, -\} \times (Q_\infty \times (\Sigma \cup \{\epsilon\}) \times Q_\infty))$ is the set of elementary actions, where the plus signal stands for insertion and the minus for deletion.

$\Pi : (Q_\infty \times (\Sigma \cup \{\epsilon\}) \times Q_\infty) \rightarrow \Gamma^*$ is the function that maps each transition in $\delta$ to a sequence of elementary adaptive actions. Normal transitions, those that should not start adaptations, are mapped to the empty sequence.

For each transition $x \in \delta$ where $\Pi(x) = \epsilon$, the $\mathscr{A}$-FSA functionally reduce to simple FSA (non-deterministic). Otherwise, the $\mathscr{A}$-FSA must first execute the sequence of elementary adaptive actions, and then the transition [1]. After the execution of the adaptive actions, the $\mathscr{A}$-FSA starts to operate on its new structure. The notion of computation for an $\mathscr{A}$-FSA takes into account the possible changes occuring during execution: a configuration is any element $(q, w, K, \Delta)$ from

[1] The adaptive device formulation allows for the use of "after actions", however, its is proven that this feature do not increases the expressiveness of the device [21].

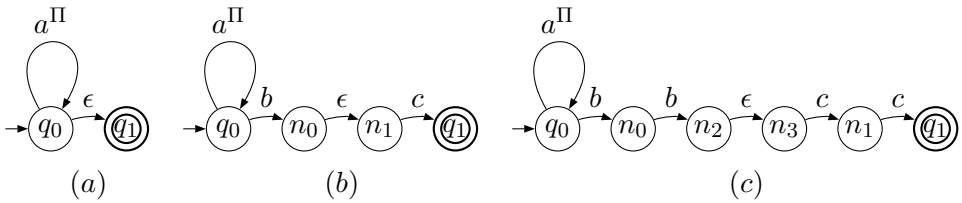


Figure 2: Adaptive Automaton that Recognizes $a^n b^n c^n$
(a) Initial Configuration (b) After first adaptation (c) After second adaptation

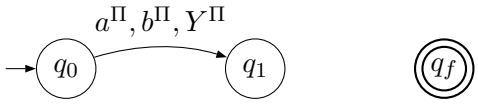


Figure 3: $\mathscr{A}$-FSA $M$ with an state $q_f$ intentionally unreachable)

$Q_\infty \times \Sigma^* \times 2^{Q_\infty} \times 2^{(Q_\infty \times (\Sigma \cup \{\epsilon\}) \times Q_\infty)}$, where $q \in Q_\infty$ is the current state, $w \in \Sigma^*$ is the part of the input string not read yet, $K \in 2^{Q_\infty}$ is the set of states referenced in the current transition set and $\Delta \in 2^{(Q \times (\Sigma \cup \{\epsilon\}) \times Q)}$ is the current transition set. The ordered pair $(C_1, C_2)$ of configurations, with $C_1 = (q, w, K, \Delta)$ and $C_2 = (q', w', K', \Delta')$, is an element of the binary relation $\vdash_M$ (*step relation*) if, and only if, $w = aw'$ for some $a \in \Sigma \cup \{\epsilon\}$, $\delta(q, a) = q'$ and the adaptive actions, $\Pi((q, a, q'))$, modify the state set $K$ and the transition relation $\Delta$, to $K'$ and $\Delta'$, respectively. A string $w \in \Sigma^*$ is *accepted* by $M$ if, and only if, exists an state $q \in F$ such that $(q_0, w, Q, \delta) \vdash_M^* (q, \epsilon, K'', \Delta'')$, with $K''$ and $\Delta''$ being any set of states and transitions.

### 3.2 An $\mathscr{A}$-FSA that recognizes a context-dependent language

Figure 2.(a) represents an $\mathscr{A}$-FSA $\mathrm{M} = \langle Q, \Sigma, q_0, F, \delta, Q_\infty, \Gamma, \Pi \rangle$ that recognizes the classical context-dependent language $a^n b^n c^n$. The superscripted $\Pi$, in figure 2, denotes an association of a transition to a sequence of adaptive actions, in this case:

$$\Pi((q_0, a, q_0)) \quad = \quad \{(-, p, \epsilon, q), (+, p, b, p'), (+, p', \epsilon, p''), (+, p'', c, q)\}, \forall p, q \in Q, p', p'' \notin Q \quad (1)$$

The subjacent mechanism keeps reading the symbols $a$ and calling the sequence of adaptive actions for each symbol $a$ read. The adaptive actions associated to the transition $(q_0, a, q_0)$ may be interpreted as: seek the $\epsilon$ transition (here working as a mark) and replaces it by a pair of transitions that reads the substring $bc$. The $\epsilon$ transition mark is kept between the transitions that read $b$ and $c$, so that when the i-th $a$ is read, the automaton will have a sequence of transitions that are able to consume the sequence $b^i c^i$. Figures 2.(b) and 2.(c) shows the automaton after reading two consecutives $a$.

### 3.3 An $\mathscr{A}$-FSA that memorizes strings

This examples illustrates a simple $\mathscr{A}$-FSA that shapes the core of the decision tree inductor that will be presented further. Basically, it reads positive examples of strings from an "unknown" language, adapting itself to memorize this strings in a prefix-tree like structure. The automaton alphabet is $\Sigma = \{a, b, Y\}$, where $Y$ (from Yes) will be sufixed to the strings that should be learned (memorized). Strings not sufixed with an $Y$ are to be classified (accepted or rejected). A string $w = w'\alpha$, with

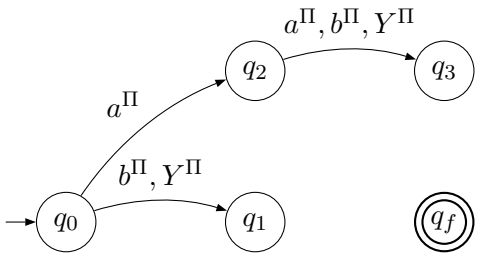


Figure 4: $M$ after reading $a$

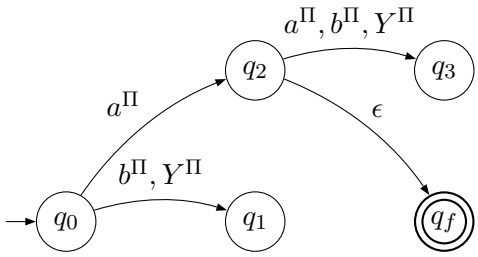


Figure 5: $M$ after reading $aS$

$\alpha \in \Sigma$ and $\alpha \neq Y$, will be accepted by $M$ iff $M$ had already read the "training" string $wS$. For instance, given the sequence $ab$ , $abS$ , $ab$ , $aa$, the automaton rejects the first occurrence of $ab$, learns $ab$ (after reading $abS$), accepts $ab$ and finally, rejects $aa$ (as it had not learned $aa$ yet). It should be clear, from this example, that what is being explored here is the ability of the formalism to handle context dependency.

Figure 3 shows the subjacent structure of the automaton $M = \langle Q, \Sigma, q_0, F, \delta, Q_\infty, \Gamma, \Pi \rangle$, where $Q = \{q_0, q_1, q_f\}$, $\Sigma = \{a, b, Y\}$, $q_0 = q_0$, $F = \{q_f\}$, $\delta$ contains the transitions presented in figure 3, $Q_\infty = \{q_f, q_0, q_1, q_2, ...\}$, $\Gamma = (\{+, -\} \times (Q_\infty \times (\Sigma \cup \{\epsilon\}) \times Q_\infty))$ and the $\Pi$ function is defined below, with $q'$ e $q''$ in the equation 2 being used to indicate two states in $Q_\infty - Q$ (states that are not in the current state set). Empty transitions, or marks, used to "boot" the automaton to the initial state (but keeping all adaptations) after each string read, are omitted from this model in order to keep the example more readable.

$$\Pi(q_i, \alpha, q_j) = \{(-, q_i, \alpha, q_j), (+, q_i, \alpha, q'), (+, q', \beta, q'')\}, \forall \alpha \in \{a, b\}, q_i \neq q_j, \beta \in \Sigma \quad (2)$$

$$\Pi(q_i, S, q_j) = \{(+, q_i, \epsilon, q_f)\}, \forall q_i \neq q_j \quad (3)$$

Figures 4 and 5 show the automaton $M$ after reading $a$ and $S$, respectively. Notice that the automaton would reject string $a$ in its initial configuration (figure 3), but would accept it after reading $aS$ (rejecting any other string, as $a$ is the only string learned).

# 4 Adaptree - Adaptive Decision Trees

In this section a novel way to formalize the decision tree induction problem, based on automata and languages theories, is presented. The general idea is to handle each training and testing instance

as a string and consider the decision tree itself as a special kind of adaptive finite state automaton, with the initial state of this automaton corresponding to the root of a classical decision tree.

The target concept is a language $L$ where all the strings have the same size: the number of attributes, including the class. Given $n$ sets $A_1, A_2, ..., A_n$, the training set is $T = \{w_1, w_2, ..., w_m\}$, where $|w_i| = n$ and $w_i = \alpha_1\alpha_2...\alpha_n$ with $\alpha_j \in A_j$, for $1 \leq j \leq n$ and $1 \leq i \leq m$. The last symbol of each string always represents the class value, hence, $|A_n|$ is the number of classes and $n-1$ the number of attributes to be considered during learning. The $A_1, A_2, ..., A_n$ sets represent attributes domain. The size of a testing string is $n-1$ (no class value).

The adaptree is basically the $\mathscr{A}$-FSA that memorizes strings, presented in the previous section, with an additional probabilistic layer that allows adaptree's to generalize beyond the training set. In order to discriminate multiple class values, the automaton will have one different final state for each possible class value. During learning, the adaptree creates a path from the initial state to the final state corresponding to the last symbol of the training string (the class), using the mechanism showed in section 3.3. When reading a testing string, adaptree may end up in a final state, classifying the string (the final state is the class), or may stop at a non-final state. In this case, the statistical inference mechanism is issued. This mechanism is presented in the next section.

### 4.1 Generalizing from the training set

When the $\mathscr{A}$-FSA stops before reaching a final state, the automaton, which have a tree-like structure, proceeds its execution following all possible paths from that state on. The information represented by each final state reached in this 'non-deterministic' execution are counted, an the most frequent class value is returned as the testing string classification. Given a testing instance $w = \alpha_1, \alpha_2, ..., \alpha_n$ and assuming that the automaton stopped when reading $\alpha_i, i < n-1$; the returned class $c$ corresponds to the most probable class given $\alpha_1, \alpha_2, ..., \alpha_i$, with estimates for $p(\alpha_n = a_n|\alpha_1, \alpha_2, ..., \alpha_i), \forall a_n \in A_n$, being taken from the training strings presented to the automaton until that moment.

It is important to note that the quality of the above estimates is highly dependent on the $i$ value, or on how far from the initial state (the root) the automaton stopped. The higher $i$ the lesser the number of training instances the estimates will count on. However, on the other hand, less information about the testing instance will be used. When $i = 0$, for instance, all training instances will be counted, as all the tree will be traversed, however, no attribute value will be considered. The order of the attributes is also important, as the attribute in the root will be used more frequently then the one in the next level, and so on. Optionally, the attributes of the datasets can be reordered using some attribute quality measure, as information-gain [35], before the learning starts.

Table 2 compares the classification performance of adaptree using the default order and information-gain reordering. The numbers represent the average percentage of correct results using a random split test with 66% for training and the remainder for testing. Each experiment where repeated 10 times and standard deviations are shown in brackets. The plus and minus signs, respectively, indicate significantly (t-test at 95% confidence level) better and worse performance of the reordered over the not-reordered versions of adaptree. All datasets where taken from and are described at the widely used UCI machine learning repository [4]. As expected, the benefits of reordering are highly dependent on the dataset, but in most cases (19 out of 33), it does not affect the performance of adaptree ( prejudicing it in two cases). As the reordering do affect the incrementality of the overall strategy - the automaton should be reconstructed after a reordination - the use of the default order should always be considered when using adaptree.

### 4.2 Adaptree and Incremental Learning

The core of the adaptree learning strategy, including the generalization mechanism based on conditional probabilities estimates, which are calculated dinamicaly, is incremental. Hence, the training and testing instances could be presented one by one, interchangeably. In fact, the strategy could also be viewed as a different kind of instance-based learning [1], with some resemblance to the approaches based on k-d trees [39]. However, if the attributes are to be reordered, the automaton should suffer some major modifications from time to time.

Datasets containing continuous value attributes should also present a problem to the incrementality of adaptree, at they must be previously discretized. In the current implementation, the Fayyad and Irani's discretization method [14] is automatically performed on each continuous feature present in the dataset, before the learning takes place. This method employs a supervised (use class information) approach based on recursive entropy minimization and a minimum description language (MDL) stopping criteria, and so, depends on the existence of some training examples.

## 5 Experimental Results

The adaptree was implemented using the Waikato Environment for Knowledge Analysis (Weka) [2], a software package that aids the development of machine learning and data mining systems. Besides implementing several machine learning algorithms, in Java, Weka encompass a set of basic routines that can be reused in the construction of new algorithms. Among them are routines to handle attribute vectors, probability distributions, dataset filters, decision trees and artificial neural networks, to name a few.

Another powerful Weka's resource is the experiment environment, where different machine learning algorithms can be compared using a variety of metrics and statistical tests. Using this resource, adaptree was compared to Id3 [35], Naive Bayes [10], C4.5 [35], 5NN [25] (K-Nearest Neighbour with K = 5) and Artificial Neural Networks trained by backpropagation [25]. Default Weka's parameters where used for all algorithms and the statistics where obtained by averaging 10 executions of the 10-Fold Stratified Cross-validation test, given a total of 100 execution for each algorithm over each dataset.

All datasets are in public domain and were extracted from the UCI machine learning repository [4]. The first, one of the most cited dataset in machine learning bibliography [15, 11, 7, 19, 12], has 150 instances of 3 different kinds of Iris, classified from 4 numeric attributes describing sepals and petals widths and lengths. The Ionosphere dataset contains 351 instances and 34 attributes representing data gather from a radar that detects the presence of free electrons in the ionosphere [38]. Two datasets from the medical area, *hypothyroid* (3772 instances, 30 attributes) and *hepatitis* [9, 5] (155 instances, 20 attributes), and another one, *glass*, representing a problem of glass type classification that can be used in criminal investigation [13], completes the experiment. This five datasets where chosen at random from the ones used in the reordering experiment.

Table 3 shows percentage-correct rates and average training time (in seconds) obtained through Weka, for each algorithm and dataset compared. The results indicate no significant performance difference (greater than 2%) from adaptree and the other algorithms in the iris and hypothyroid datasets. Backpropagation is significantly better in the other datasets, however, adaptree performs well in relation to the other algorithms. It is worth noting that backpropagation's training time is tremendously higher.

[2] Free software available at http://www.cs.waikato.ac.nz/ ml/weka/

An important result, shown in table 3, is adaptree's superiority over Id3, the only one among the five tested algorithms that, as adaptree, do not handle continuous values intrinsically: the same discretization filter were used for adaptree and Id3. As to the execution time, tables 3 and 4 indicate the adaptree's performance is comparable C4.5's, both in training an testing. The time values shown in the tables were obtained from the average time of the 100 runs, on the same hardware and software platform: Penthium III CPU, 700Mhz and 257Mb RAM. Table 4 indicates that the greater decision tree size generated by adaptree, in relation to Id3 and C4.5, do not compromise execution time.

It is important to note that all the algorithms could be improved using different parameters fine-tuning to each dataset. In order to prevent from this kind of bias, adaptree was not fine-tuned, and the same version and parameters were used in all runings.

## 6 Conclusions and Future Work

In this paper we presented a new algorithm for decision tree induction whose performance rivals some well-known machine learning algorithms. One important feature of this algorithm is a new formalization approach, based on automata theory, enriched by adaptive techniques. This approach links the usual decision tree learning techniques and automata theory, paving the way for the construction of other solutions from the interaction of this two areas. Some interesting research may result from applying the grammatical inference techniques [2, 20] to the decision tree induction framework, and vice-versa.

Although adaptree's core is incremental some further research must be conducted in order to study the impact and search alternatives for handling continuous values. An experimental work involving the utilization of adaptree in a prototype system implementing eyes-gaze interaction is described in [32]. This prototype, which intermix machine learning, adaptive automata and computer vision techniques, is a tic-tac-toe game that can be trained to be played via eyes gazing. Images are captured using a webcam placed over the computer monitor and directed at the user face. Even after training, the user may correct wrong throws, due to miss-classifications of the eyes image, by pointing and looking at the right place and instructing the system to collect more training instances. This system is a good example of how incremental learning may be important.

## References

[1] D. W. Aha, D. Kibler, and M. Albert. Instance-based learning algorithms. *Machine Learning*, 6:37–66, 1991.

[2] R. Alquezar and A. Sanfeliu. Incremental grammatical inference from positive and negative data using unbiased finite state automata, 1994.

[3] B. A. Basseto and J. J. Neto. 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.

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

[5] G. Cestnik, I. Konenenko, and I. Bratko. *Progress in Machine Learning*, chapter Assistant-86: A Knowledge-Elicitation Tool for Sophisticated Users, pages 31–45. Sigma Press, 1987.

[6] E. R. Costa, A. R. Hirakawa, and J. J. Neto. 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.

[7] B.V. Dasarathy. Nosing around the neighborhood: A new system structure and classification rule for recognition in partially exposed environments. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 2(1):67–71, 1980.

[8] Carlos Eduardo Dantas de Menezes. Um método para a construção de analisadores morfológicos, aplicado à lingua portuguesa, baseado em autômatos adaptativos. Master's thesis, Escola Politécnica, Universidade de Sao Paulo, São Paulo, Brasil, Julho 2000.

[9] P. Diaconis and B. Efron. Computer-intensive methods in statistics. *Scientific American*, 248, 1983.

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

[11] R. O. Duda and P. E. Hart. *Pattern Classification and Scene Analysis*. John Wiley and Sons, 1973.

[12] Cheeseman et alli. Autoclass: A bayesian classification system. *Proceedings of the International Machine Learning Conference*, pages 54–64, 1988.

[13] Ian W. Evett and E. J. Spiehler. Rule induction in forensic science. In *KBS in Goverment*, pages 107–118. Online Publications, 1987.

[14] U. M. Fayyad and K. B. Irani. 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.

[15] R. A. Fisher. The use of multiple measurements in taxonomic problems. *Annual Eugenics*, 7:179–188, 1936.

[16] M. R. Forster. *The New Science of Simplicity*, chapter 6, pages 83–119. Cambridge University Press, 2001.

[17] E. Frank. *Pruning Decision Trees and Lists*. PhD thesis, University of Waikato, Department of Computer Science, Hamilton, New Zealand., 2000.

[18] Eibe Frank, Yong Wang, Stuart Inglis, Geoffrey Holmes, and Ian H. Witten. Using model trees for classification. *Machine Learning*, 32(1):63–76, 1998.

[19] G.W. Gates. The reduced nearest neighbor rule. *IEEE Transactions on Information Theory*, pages 431–433, May 1972.

[20] Colin De La Higuera. Current trends in grammatical inference. *Lecture Notes in Computer Science*, 1876:28–32, 2001.

[21] M. K. Iwai. *Um Formalismo Gramatical Adaptativo para Linguagens Dependentes de Contexto*. PhD thesis, Escola Politécnica, Universidade de São Paulo, São Paulo, Brasil, 2000.

[22] J. R. Almeida Junior, J. J. Neto, and A. R. Hirakawa. Adaptive automata for independent autonomous nagivation in unknown enviroment. In *Proceedings of IASTED International Conference on Applied Simulation and Modelling - ASM 2000*, Banff, Alberta, 2000.

[23] R. Kothari and M. Dong. *Pattern Recognition: From Classical to Modern Approaches*, chapter 6, Decision Trees For Classification: A Review and Some New Results, pages 169–184. World Scientific, 2001.

[24] J. Mingers. Expert systems - rule induction with statistical data. *Journal of the Operational Research Society*, 38:39–47, 1987.

[25] Tom M. Mitchell. *Machine Learning*. McGraw-Hill, 1997.

[26] J. J. Neto. Contribuição à metodologia de construção de compiladores. *Post-Doctoral Tese de Livre Docência, Escola Politécnica, Universidade de São Paulo*, 1993.

[27] J. J. Neto. Solving complex problems efficiently with adaptative automata. In *Conference on the Implementation and Application of Automata - CIAA 2000*, Ontario, Canada, July 2000.

[28] J. J. Neto. 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.

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

[30] Joo Jos Neto, Csar Bravo Pariente, and Fabrizio Leonardi. Compiler construction - a pedagogical approach. In *Proceedings of the V International Congress on Informatic Engineering - ICIE 99*, Buenos Aires, Argentina, August 1999.

[31] Neil C. Berkman Paul E. Utgoff and Jeffery A. Clouse. Decision tree induction based on efficient tree restructuring. *Machine Learning*, 29(1):5–44, October 1997.

[32] H. Pistori, J. J. Neto, and E. R. Costa. Utilização de tecnologia adaptativa na detecção da direção do olhar (completo). *SPC Magazine*, 2(2), Maio 2003.

[33] J. R. Quinlan. Learning with Continuous Classes. In *5th Australian Joint Conference on Artificial Intelligence*, pages 343–348, 1992.

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

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

[36] C. Schaffer. Overfitting avoidance as bias. In *IJCAI-91 Workshop on Evaluating and Changing Representation in Machine Learning*, Sydney, Australia, 1991.

[37] Cullen Schaffer. A conservation law for generalization performance. In W. Cohen and H. Hirsh, editors, *Proceedings of the Eleventh International Conference on Machine Learning*. Morgan Kaufmann, 1994.

[38] V. G. Sigillito and S. P. Wing et alli. Classification of radar returns from the ionosphere using neural networks. *Johns Hopkins APL Technical Digest*, 10:262–266, 1989.

[39] Marko Robnik Sikonja. Speeding up relief algorithms with k-d trees. *Proceedings of the Electrotechnical and Computer Science*, 1998.

[40] Z. H. Zhou and Z. Q. Chen. Hybrid decision tree. *Knowledge-Based Systems*, 15(8):515–528, 2002.

| Dataset | Default Attribute Order | Re-ordering using Information Gain |
|---|---|---|
| anneal.ORIG | 79.11( 2.43) | 84.1 ( 2.42) + |
| anneal | 99.44( 0.41) | 95.74( 3.02) - |
| audiology | 53.12( 4.91) | 66.88( 7.28) + |
| autos | 63.29( 6.8 ) | 75.14( 6.07) + |
| balance-scale | 75.61( 4.33) | 75.61( 4.33) |
| breast-cancer | 64.54( 5.37) | 65.88( 3.52) |
| wisconsin-breast-cancer | 94.16( 1.43) | 94.2 ( 1.38) |
| horse-colic.ORIG | 66.64( 3.37) | 68.24( 4.2 ) + |
| horse-colic | 75.36( 2.61) | 79.52( 3.72) + |
| credit-rating | 69.4 ( 3.05) | 82 ( 2.48) + |
| german-credit | 68.32( 2.45) | 68.21( 1.89) |
| pima-diabetes | 73.87( 2.56) | 74.06( 2.59) |
| Glass | 55.07(11.71) | 58.49( 9.29) |
| cleveland-14-heart-diseas | 78.16( 3.86) | 77.77( 3.67) |
| hungarian-14-heart-diseas | 77.3 ( 2.79) | 78.5 ( 2.59) |
| heart-statlog | 77.72( 4.14) | 78.26( 3.28) |
| hepatitis | 77.92( 4.96) | 76.98( 4.43) |
| hypothyroid | 97.88( 0.3 ) | 98.16( 0.42) |
| ionosphere | 89.58( 2.14) | 84.87( 2.8 ) - |
| iris | 91.96( 5.43) | 93.92( 3.26) |
| kr-vs-kp | 79.79( 1.39) | 96.32( 0.65) + |
| labor | 75.79(12.21) | 86.84( 7.94) + |
| lymphography | 71.2 ( 4.54) | 75.4 ( 5.08) |
| mushroom | 99.68( 0.15) | 100 ( 0 ) + |
| primary-tumor | 27.04( 2.79) | 32.87( 6.01) + |
| segment | 71.57( 2.37) | 89.97( 1.92) + |
| sick | 96.44( 0.49) | 97.57( 0.27) + |
| sonar | 68.45( 5.68) | 65.07( 6.33) |
| soybean | 43.41( 2.68) | 82.37( 2.43) + |
| vehicle | 63.61( 2.73) | 63.58( 3.23) |
| vote | 90.14( 2.59) | 94.32( 1.53) + |
| vowel | 76.17( 3.35) | 74.87( 2.42) |
| zoo | 40.88( 5.27) | 40.88( 5.27) |

Table 2: Adaptree with and without attribute reordering

| Dataset | Adaptree | Id3 | NBayes | 5NN | C4.5 | Backpro |
|---|---|---|---|---|---|---|
| iris | 93.92% (0.03s) | 89.22% (0.88s) | 93.33% (0s) | 92.75% (0s) | 94.31% (0.01s) | 93.08% (1.83s) |
| ionosphere | 86.72% (0.66s) | 84.29% (2.09s) | 90.76% (0.03s) | 89.58% (0.01s) | 87.39% (0.48s) | 93.84% (74.66s) |
| hypothyroid | 92.22% (5.18s) | 90.12% (29.05s) | 92.78% (0.17s) | 93.28% (0.07s) | 93.41% (4.37s) | 94.57% (830.67s) |
| hepatitis | 76.6 % (0.1s) | 70.75% (1.23s) | 84.15% (0s) | 81.13% (0s) | 80.75% (0.06s) | 82.39% (10.5s) |
| Glass | 52.47% (0.07s) | 47.95% (5.86s) | 52.19% (0.01s) | 53.42% (0s) | 52.47% (0.11s) | 68.95% (10.71s) |

Table 3: Correct answers percentage and average classification time (in brackets)

| Dataset | Adaptree | Id3 | NBayes | 5NN | C4.5 | Backpro |
|---|---|---|---|---|---|---|
| iris | 0s | 0s | 0.01s | 0.05s | 0.01s | 0.01s |
| ionosphere | 0.01s | 0.01s | 0.04s | 1.3s | 0.01s | 0.13s |
| hypothyroid | 1.3s | 0.11s | 0.45s | 184.48s | 0.07s | 1.17s |
| hepatitis | 0s | 0s | 0.01s | 0.18s | 0s | 0.02s |
| Glass | 0.01s | 0s | 0.02s | 0.37s | 0.01s | 0.01s |

Table 4: Average time to classify an instance