Incremental Fuzzy Decision Trees

Marina Guetova and Steffen Hölldobler and Hans-Peter Störr *

Artificial Intelligence Institute
Department of Computer Science
Technische Universität Dresden
01062 Dresden, Germany
{sh,hans-peter}@inf.tu-dresden.de

Abstract. We present a new classification algorithm that combines three properties: It generates decision trees, which proved a valuable and intelligible tool for classification and generalization of data; it utilizes fuzzy logic, that provides for a fine grained description of classified items adequate for human reasoning; and it is incremental, allowing rapid alternation of classification and learning of new data. The algorithm generalizes known non–incremental algorithms for top down induction of fuzzy decision trees, as well as known incremental algorithms for induction of decision trees in classical logic. The algorithm is shown to be terminating and to yield results equivalent to the non–incremental version.

Keywords: incremental learning, classification, decision trees, fuzzy logic

1 Introduction

Decision trees have proven to be a valuable tool for description, classification and generalization of data. This is related to the compact and intelligible representation of the learned classification function, and the availability of a large number of efficient algorithms for their automated construction [8]. They provide a hierarchical way to represent rules underlying data. Today, a wealth of algorithms for the automatic construction of decision trees can be traced back to the ancestors ID3 [9] or CART [3].

In many practical applications, the data used are inherently of imprecise and subjective nature. A popular approach to capture this vagueness of information is the use of fuzzy logic, going back to Zadeh [12]. The basic idea of fuzzy logic is to replace the "crisp" truth values **1** and **0** by a degree of truth in the interval $[0, 1]$. In many respects, one can view classical logic as a special case of fuzzy logic, providing a more fine grained representation for imprecise human judgments. Consider, for instance, the question whether a jacket is fashionable. Here, a simple yes/no judgment looses information, or might even be infeasible.

To combine the advantages of decision trees and fuzzy logic, the concept of a decision tree has been generalized to fuzzy logic, and there is a number of algorithms creating such trees, e.g. [7, 13]. While there are alternatives, we feel the fuzzy decision trees have some merits not easily achievable by other methods. One approach to

* This author acknowledges support by hybris GmbH, Munich, within the project "Intelligent Systems in e-Commerce" (ISeC).

eliminate unfortunate need to have only yes/no judgments is the use of continuous attribute values, as done in many algorithms, e.g. Quinlans C4.5 [10]. However, there is a difference what continuous attributes and fuzzy values can naturally express. While decision trees with continuous attributes can capture more fine grained decisions (e.g. "if the attribute temperature is between 25.3 and 27.7 the state is classified as safe"), exactly this expressiveness can turn into a problem, since the algorithm has to determine the split points itself. Fuzzy decision trees do not admit such fine grained distinctions of the fuzzy truth values (which would not be appropriate in our application, anyway), and thus do not have to make the sometimes computational expensive decision between the many possible split points of one continuous attribute. On the other hand, fuzzy logic is able to generalize multi-valued attributes naturally: the value for one attribute, described by a linguistic variable in fuzzy logic, can be chosen freely out of a multi-dimensional continuum of truth-degrees for all linguistic terms (corresponding to the attributes values) of that linguistic variable. Furthermore, unlike the approaches based on classical logic, a fuzzy decision tree is able to capture degrees of truth in the calculated classification for new examples; "crisp" truth values yielded only after the optional defuzzification. [1]

A complimentary set of approaches is the use of decision trees for probability estimation, as e.g. in CART [3], which can also be viewed as a generalization of the classical logic. We do not have an easy answer to the relation of those to approaches using fuzzy logic; yet we feel in our case fuzzy logic captures better the nature of human intuition: in our example, a jacket cannot be described as fashionable with 30% probability.

To the best of our knowledge, the present algorithms for induction of fuzzy decision trees are non–incremental. However, in many applications the collection of new data and the application of the learned function is intertwined. For instance in our project "Intelligent Systems in e-Commerce" (ISeC) [2] we try to approximate the preferences of the customers using an e-shop online. Each new web page visited by a user is customized to the learned preferences of the user, and each visit of a web page yields new data on the user behavior, which are used to update the preference function memorized for the user. In such an application non–incremental classification algorithms meet their limits, because they have no way to adapt their results to the new data, but rather have to be restarted from scratch. In such an application, incremental algorithms almost always perform better, because they simply adjust their results.

In other words, we are looking for classification algorithms that are (i) based on decision trees, (ii) utilize fuzzy logic, and (iii) are incremental. Surprisingly, there are non–incremental algorithms for fuzzy decision trees [7, 13], and incremental algorithms for decision trees [11], but we have found no algorithm that satisfies all three properties. The work presented in this paper is an attempt to fill this gap.

[1] An interesting approach that has been suggested, is to to break examples into fractional examples (as used in C4.5 to deal with unknown attributes) capturing the different attribute values. Under some circumstances this can lead to similar results as the fuzzy decision trees, but the number of fractional examples needed is exponential in the number of attributes an example is split on.

This paper discusses the general ideas underlying our algorithm for incremental induction of fuzzy decision trees (IIFDT). In [5] the algorithms are presented in more detail. Moreover it contains the proofs which are omitted for space reasons.

After introducing our notation in Section 2, we will shortly present the concept of decision trees based on fuzzy logic (Section 3). Starting from the general idea of top–down induction of such trees (Section 4, based on Janikows FID [7]), we move in Section 5 and 6 to an incremental algorithm along a path similarly to Utgoff et. al.'s algorithm for incremental tree induction in the classical case [11]. Section 7 states our results on termination of the incremental algorithm and on the identity of the decision trees generated by the incremental and non–incremental algorithm. We conclude with a discussion in Section 8.

2 Notations

Let us first introduce some basic notions. A **fuzzy set** μ over a set $\mathcal{X}$ is a mapping from $\mathcal{X}$ into the interval $[0,1]$ of fuzzy degrees of truth. In this paper, fuzzy sets are denoted by small Greek letters. The **size** of a fuzzy set μ over $\mathcal{X}$ is defined as $|\mu| = \sum_{x \in \mathcal{X}} \mu(x)$. Let $\mathcal{U}$ be a universe of examples. A **condition** is a fuzzy set over $\mathcal{U}$. This generalizes the concept of attribute–value pairs often used for describing the examples: to each attribute–value pair (a, v) corresponds a fuzzy set $\mu_{a,v}$ that specifies the degree $\mu_{a,v}(e)$ an example e is described by the attribute–value pair (a, v). In the terminology of fuzzy logic, an attribute corresponds to a linguistic variable, and a value to a linguistic term. But we do not use these notions here explicitly. Because we use only binary tests in this paper, we write, e.g., $\mu_{technical}$ and $\overline{\mu_{technical}}$ instead of $\mu_{technical,true}$ and $\mu_{technical,false}$.

A learning/classification task consists of a finite *set of training examples* $\mathcal{L} \subseteq \mathcal{U}$, a finite set $\mathcal{T}$ of conditions that can be used to describe examples, a finite set of classes $\mathcal{C}$, and a (partially known) mapping $\chi : \mathcal{U} \to \mathcal{C}$ called *classification function*, that defines for each example $e \in \mathcal{L}$ a class $\chi(e)$. The goal of a learning algorithm is to generate an internal representation describing the set of training examples, e.g. a decision tree. This representation can be used to predict the class of new examples, i.e. examples which are not in $\mathcal{L}$ and for which c is unknown. The goal is usually to reach a good generalization, i.e. that the classes of unseen examples will be predicted with a good accuracy.

Example 1. Suppose a user is given some information about products. This information is divided into several parts like a picture e_1, a short texts e_2 describing features of the product, a text e_3 representing details etc. The user gives positive/negative feedback on her interest in that particular piece of information. For example, if an image is presented and the user enlarges it, then the feedback concerning the image is positive. A second click on the image can shrink the image again, indicating negative interest. Likewise, a text can be decollapsed from its headline / collapsed into its headline by a single mouseclick, notifying the system about the interest of the user in that text. We aim to get an internal representation of the users' preferences in order to estimate her interest before presenting a new piece of information, such that we can prefer information likely to be relevant for her in the presentation.

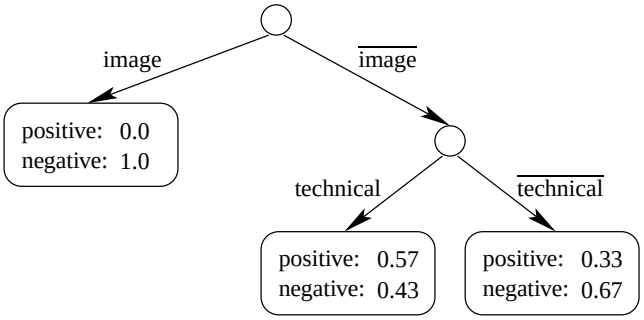


Fig. 1. A decision tree specifying the interests of the user in Example 1 considering e_1, e_2, e_3. How such a tree is calculated will become apparent later.

If she is not interested in images, and weakly prefers technical information, we could have the following learning problem:

$\mathcal{U} = \{e_1, e_2, e_3, \dots\}$
$\mathcal{C} = \{$*positive*, *negative*$\}$

e	$\mu_{image}(e)$	$\mu_{technical}(e)$	$\mu_{design}(e)$	$\chi(e)$
e_1	1	0.5	0.4	*negative*
e_2	0	0.6	0.6	*negative*
e_3	0	0.8	0.5	*positive*
$\vdots$	$\vdots$	$\vdots$	$\vdots$	$\vdots$

Finally, let us introduce some operations of fuzzy logic. Here, the operations corresponding to the classical conjunction, disjunction, and negation are the t-norm $\top$, the t-conorm $\perp$ and the complement $\overline{\ }$. In fuzzy logic, these functions can be chosen freely as long as they satisfy some technical conditions. As a requirement for our algorithm we restrict ourselves to the widely used so called algebraic norms

$$\top(x, y) = x * y \qquad\qquad \perp(x, y) = x + y - x * y$$

and the complement $\overline{x} = 1 - x$, where $*, +, -$ are the usual multiplication, addition and subtraction operations on real numbers.

The fuzzy operators are generalized to the operators $\cap, \cup$ and $\overline{\ }$ on fuzzy sets by applying the operators $\top$, $\perp$ and $\overline{\ }$ point–wise. Because they are commutative and associative, they can be extended to an arbitrary number of arguments. For more information about fuzzy logic the reader is referred to the widely available literature, e.g. [4].

3 Fuzzy Decision Trees

A *fuzzy decision tree* (see e.g. Fig. 1) is a tree structure where every edge is annotated with a condition, and every leaf is annotated with a fuzzy set over $\mathcal{C}$. For conciseness,

we consider only binary trees in this paper, where one of the conditions at the outgoing edges of a node is chosen from $\mathcal{T}$ (we speak of the **test condition** of the node), and the condition at the other outgoing edge is the negation of the test condition. The restriction to binary trees is lifted in [5]. A further restriction is that each condition is used at most once in each path from root to leaf.

Consider a path from the root to a leaf of the tree. Intuitively, such a path can be interpreted as follows: whenever an example fulfills the conditions the edges of a path are annotated with, we expect the example to belong to the classes according to the fuzzy set the leaf is annotated with. E.g., in Fig. 1 along the edges labelled $\overline{\text{image}}$ and technical gives us a clue, that the interest of the user in non–technical images is classified to positive with a degree of truth 0.57 and negative with a degree of truth 0.43. One should observe that unlike in decision trees based on classical logic, the conditions on the outgoing edges of a node are not necessarily mutually exclusive: e.g. an example can be both technical and non–technical to a non–zero degree of truth. So we have to combine the clues given by all paths into one final result. There is a variety of methods to accomplish this [7]. In this paper, we compute the weighted average of the fuzzy sets the leafs are annotated with, where each fuzzy set is weighted by the degree the example belongs to the conjunction of the conditions on the edges of the path. After presenting this formally, we will discuss an example. One should observe that CLASSIFY yields a fuzzy degree of truth as value; if a crisp output value is desired one can apply any of the well–known defuzzification methods [4], e.g. take the class with the highest truth value.

Example (1 continued). We classify an example

e	$\mu_{image}(e)$	$\mu_{technical}(e)$	μ_{design}
e_c	0	0.3	0.9

according to the fuzzy decision tree in Fig. 1:

Path	*leaf annotation*	*weight*	$\top$(*weight*, *annotation*)
image	$\{(pos, 0.0), (neg, 1.0)\}$	0	$\{(pos, 0.0), (neg, 0.0)\}$
$\overline{\textit{image}}$, *technical*	$\{(pos, 0.57), (neg, 0.43)\}$	$\top(1, 0.3)$	$\{(pos, 0.17), (neg, 0.13)\}$
$\overline{\textit{image}}$, $\overline{\textit{technical}}$	$\{(pos, 0.33), (neg, 0.67)\}$	$\top(1, 0.7)$	$\{(pos, 0.23), (neg, 0.47)\}$
CLASSIFY(e_c)			$\{(pos, 0.4), (neg, 0.6)\}$

The example is estimated negative to a truth degree 0.6*. Thus, the corresponding text would be shown in collapsed form.*

4 Induction of Fuzzy Decision Trees (FID)

Suppose we have a set of training examples $\mathcal{L}$, and we want to construct a fuzzy decision tree classifying those examples. Similarly to ID3, Janikow describes a process of top-down induction of the tree [7]. The main idea is to partition a fuzzy set of training examples according to a heuristically chosen condition recursively, until the remaining

CLASSIFY(e):

$$\text{CLASSIFY}(e) = \sum_{\text{paths } \langle\theta_1,\ldots,\theta_i,\gamma\rangle \text{ in tree}} \top(\theta_1(e),\ldots,\theta_i(e)) \cdot \gamma \ ,$$

where a path (from root to leaf) is denoted by a vector $\langle\theta_1,\ldots,\theta_i,\gamma\rangle$ of the conditions $\theta_1,\ldots,\theta_i$ the edges of the path are annotated with and the fuzzy set γ the leaf of this path is annotated with. The scalar multiplication $\cdot$ and sum are element-wise.

Fig. 2. Algorithmus CLASSIFY

parts satisfy a pruning criterion, e.g. are either largely of one class, or of negible size. The partitioning is then represented as a tree.

One should observe, that partitioning a set in the context of fuzzy logic does not necessarily mean that each example occurs in only one of the partitions to a nonzero degree. E.g. the fuzzy set $\lambda = \{(e_1,0),(e_2,1),(e_3,1)\}$ partitioned by the condition $\mu_{technical}$ and $\overline{\mu_{technical}}$ yields the partitions

$$\lambda \cap \mu_{technical} = \{(e_1,0),(e_2,0.6),(e_3,0.8)\}$$
$$\lambda \cap \overline{\mu_{technical}} = \{(e_1,0),(e_2,0.4),(e_3,0.2)\} \ .$$

The basic idea behind the partitioning process is to reach partitions in which the vast majority of the examples belongs to only one class, i.e. the partition is "pure". Thus, if an example of unknown class belongs to such a partition, one can safely assume it is of the class the majority is in. A widely used idea to reach this goal quickly is to select a condition θ in each step such that the resulting partitions are as pure as possible. One measure for this "purity" of a partition λ is the information theoretic **information content** of the distribution of the classes in a fuzzy set λ of examples: [2]

$$IC(\lambda) = -\sum_{c\in\mathcal{C}} \left[\frac{|\lambda \cap \mu_c|}{|\lambda|} \log_2 \left(\frac{|\lambda \cap \mu_c|}{|\lambda|} \right) \right] \ ,$$

where $\mu_c = \{(e,1) \mid e \in \mathcal{U} \ \wedge \ \chi(e) = c\} \cup \{(e,0) \mid e \in \mathcal{U} \ \wedge \ \chi(e) \neq c\}$.

Because the algorithm divides the set of training examples in each step in two partitions of in general different size, we can estimate the heuristic quality of the division by a condition θ as the weighted mean of the information contents of both partitions weighted by the partition size:

$$ICW(\lambda,\theta) = \frac{|\lambda \cap \theta|}{|\lambda|} IC(\lambda \cap \theta) + \frac{|\lambda \cap \overline{\theta}|}{|\lambda|} IC(\lambda \cap \overline{\theta}) \ .$$

[2] Strictly speaking, this is not the information content, because the information content is calculated from probabilities, not from fuzzy truth degrees. However, the discussed heuristic is in close analogy to the information gain heuristic in the classical case, so this name is used here as well.

FID $(\lambda, \chi, \mathcal{T})$:

parameters: a fuzzy set of examples λ , a classification function χ, a set of conditions $\mathcal{T}$.
implicit parameters: test selection heuristic $\mathcal{H}$ and pruning criterion $\mathcal{P}$
returns a fuzzy decision tree

When the pruning criterion $\mathcal{P}$ is met, return a leaf annotated with a fuzzy set γ representing the distribution of the classes in λ:

$$\gamma = \left\{ \sum_{e \in \mathcal{L}} \top\left(\mu_c(e), \lambda(e)\right) \;\middle|\; c \in \mathcal{C} \right\}$$

else Choose condition θ from $\mathcal{T}$ according to a heuristic $\mathcal{H}$, and return tree:

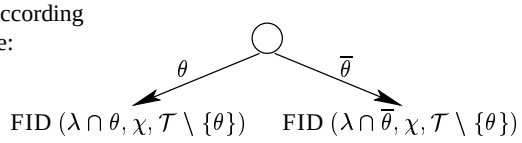


Fig. 3. Algorithm FID

The basic idea of the so called **information gain heuristic** is choose the condition θ for partitioning λ that has the lowest weighted information content $ICW(\lambda, \theta)$.

Only one more ingredient is missing to formally define the algorithm FID for generation of fuzzy decision trees: we need to specify when to stop the partitioning. This is the task of a **pruning criterion**. In this paper we use a criterion that is fulfilled if the size $|\lambda|$ of the partition is below 1.5 or if more than 90% of the examples of the partition belong to one class. Both the pruning criterion and the heuristic need to be fixed during the lifetime of a tree, so we just mention these as implicit parameters in the algorithms.

In this algorithm, as well as in most of the algorithms discussed later, the restriction that in each path from root to leaf each condition occurs only once is ensured by choosing the tests from the parameter $\mathcal{T}$, that contains all conditions that may still be used in the subtree we are about to construct or modify. In the recursive calls this parameter is adjusted accordingly. The reader is invited to verify that FID called with the set of training examples $\lambda = \{(e_1, 1), (e_2, 1), (e_3, 1)\}$, the information gain heuristic, the mentioned pruning criterion, and χ and $\mathcal{T}$ chosen according to Example 1 yields the decision tree shown in Fig. 1.

The algorithm FID is by design a generalization of ID3: if all degrees of truth are 0 or 1, FID yields the same tree as ID3, and CLASSIFY returns the same classification result for new examples as the classification procedure of ID3. Note that the selection of a test condition is a computationally quite expensive process for large sets of training examples, because one has to calculate $|\lambda|$, $|\lambda \cap \mu_c|$, $|\lambda \cap \theta|$ and $|\lambda \cap \mu_c \cap \theta|$ for all $c \in \mathcal{C}$ and all conditions $\theta \in \mathcal{T}$. This will be one of the crucial points to consider in the incremental version.

5 Incremental Induction

Imagine we have a tree t constructed from a training set λ. Now we get new information and build a new tree t' that is based on a new training set λ' containing λ and a couple of new training examples. Comparing the two trees, two things may occur. First, the annotated class distributions may change, and second, the structure of the tree may change. We follow [11] that proposes to separate these changes explicitly.

In a first step, we just adapt the annotations on the tree according to the new examples without changing the structure of the tree, except turning leafs into subtrees when the pruning criterion is no longer met. Thus, the new examples are taken into considerations without computationally expensive recalculations of the tree. However, because the conditions at the inner tree nodes are not changed in this scheme, these will not necessarily be the conditions the heuristic would choose at that point if the tree was constructed with FID . So we can expect, that the tree has more nodes than necessary, and that the ability of the tree to capture the essence of the classification function will suffer. Thus, the ability to generalize the received information to classify new, unseen, examples will decrease. So we have to perform a second step once in a while, that corrects these shortcomings and ensures that the test condition at each inner node is the one the used heuristic recommends.

Our algorithm consists of four recursive procedures used to perform various tasks in updating the fuzzy decision tree. For the first step, ADDEXAMPLE performs the discussed adding of new examples without changing the structure of the tree. For the second step, OPTIMIZE checks whether structure changes in the tree are necessary, and performs the necessary changes in the tree. OPTIMIZE itself uses a procedure TRANSPOSE for swapping nodes in a subtree such that the conditions on the top node of the subtree are as desired. These procedures are discussed in the next section after some preliminary considerations. Additionally, the procedure CLASSIFY (already described in Fig. 2) can be used to estimate the class of unseen examples based on the tree.

In practice, calls to ADDEXAMPLE, OPTIMIZE and CLASSIFY can be freely mixed. For testing purposes, as discussed later, we repeatedly apply the following learning regime:

1. Learn u new examples with ADDEXAMPLE.
2. Call OPTIMIZE if the last optimization was at least v learned examples ago.
3. Classify w examples with CLASSIFY.

with the parameters, say, $u = 5, v = 50, w = 5$.

6 Restructuring the Tree

After adding new examples to the tree the split conditions are likely to be no longer **optimal** in the sense that they are the condition chosen by the given heuristic. Thus, we have (i) to find out what the best split condition is at each node, and (ii) to change the tree accordingly. As discussed before, computing the heuristic is a computationally expensive process. In the context of classical decision trees, [11] suggests to memorize the values needed to calculate the heuristics at each node of the tree, such that they need

not to be recalculated every time the condition is checked. These values are updated each time a new example is added, or the tree structure is changed. We adopt this idea for fuzzy decision trees as well. For the calculation of the information gain heuristic or a similar heuristic this concerns the values $|\lambda_t|$, $|\lambda_t \cap \mu_c|$, $|\lambda_t \cap \theta|$ and $|\lambda_t \cap \mu_c \cap \theta|$ for all $c \in \mathcal{C}$ and all conditions $\theta \in \mathcal{T}$, where λ_t is the partition of the fuzzy set λ of training examples added so far, that corresponds to each node t. I.e.:

$$\lambda_t = \lambda \cap \theta_1 \cap \cdots \cap \theta_i \ , \tag{1}$$

where $\theta_1, \ldots, \theta_i$ are the conditions the edges along the path from root to t are labelled with. To put it differently - on each node we do some bookkeeping on the set of training examples added to the subtree below that node, such that the heuristics can be calculated more efficiently.

As an addition in every node there is a *dirty* flag, that is set whenever this bookkeeping information that node is changed. As a consequence, the algorithm can skip checking the test condition of a node for optimality whenever the dirty flag is not set. Furthermore, the set λ_t needs to be memorized at each leaf node at the tree, because the leaf might be replaced by a subtree later on.

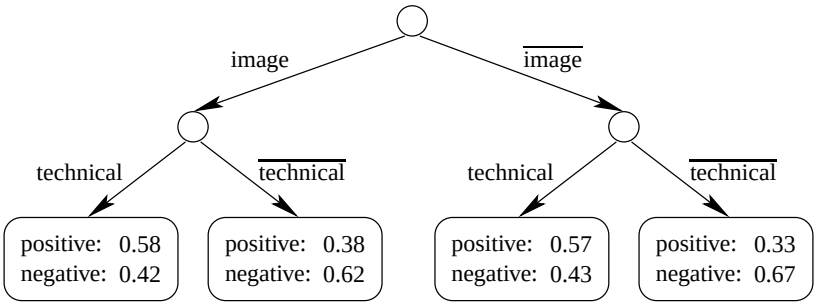
So we introduce the new concept of an **incremental fuzzy decision tree**: a fuzzy decision tree augmented with the data discussed in the previous paragraphs. For brevity, we will call these decision trees if the meaning is clear from context. A node is called **correct**, if the memorized information, that is updated by the algorithm, is equal to the values according to the definition above.

The job of the first algorithm in the suite, ADDEXAMPLE, is mostly to recursively update the memorized values at the fuzzy decision tree. If the pruning criterion is no longer met at a leaf node, the leaf node is split, so that the tree can grow to accomodate new information. In Fig. 4 the algorithm ADDEXAMPLE is given in detail.

Example (1 continued). Starting with an empty tree t, we get after adding examples e_1, e_2 and e_3 an incremental fuzzy tree like Fig. 1. (We leave out the memorized data for clarity.) Let us add a further example to the tree.

e	$\mu_{image}(e)$	$\mu_{technical}(e)$	μ_{design}	$\chi(e)$
e_4	1	0.7	0.4	*positive*

The call ADDEXAMPLE$(t, e_4, 1, \chi, \mathcal{T})$ *modifies t to the following tree:*



ADDEXAMPLE$(t, e, m, \chi, \mathcal{T})$:

parameters: A fuzzy decision tree with root node t, an example e and its membership value m to the tree, a classification function χ, aand set of conditions $\mathcal{T}$.
implicit parameters: test selection heuristic $\mathcal{H}$ and pruning criterion $\mathcal{P}$.
effect: Modifies t such that it takes e into account.

- If $m = 0$ return.
- If t is an inner node with edges marked θ and $\overline{\theta}$ leading to subtrees t_l and t_r then
  - update Node t with e and m.
  - ADDEXAMPLE$(t_r, e, \top(m, \overline{\theta(e)}), \chi, \mathcal{T} \setminus \{\theta\})$
  - ADDEXAMPLE$(t_l, e, \top(m, \theta(e)), \chi, \mathcal{T} \setminus \{\theta\})$
- else
  - update memorized information at Node t with e and m
  - If pruning criterion is *not* met (anymore) and $\mathcal{T} \neq \emptyset$:
    - Choose condition θ from $\mathcal{T}$ according to heuristig $\mathcal{H}$
    - Replace t by an empty tree of depth 1 with edges labelled θ and $\overline{\theta}$ and add the examples memorized on t one by one with ADDEXAMPLE.

Fig. 4. Algorithm ADDEXAMPLE

The left node was transformed to a subtree, because it did no longer fulfill the pruning criterion.

The task of TRANSPOSE is to ensure that the condition at the root node of a subtree t is the condition θ. This is done in two steps.

First, we ensure that the nodes at the 2nd level of subtree t have the test condition θ. If they are leafs, this is done by constructing an empty tree of depth 1 with the test condition θ (i.e. a tree, where all memorized values are set to 0), and adding the examples memorized at the leaf to this new subtree with ADDEXAMPLE. For the nodes at the 2nd level that are not leaves, we recursively call TRANSPOSE to ensure θ is at the top of this subtree.

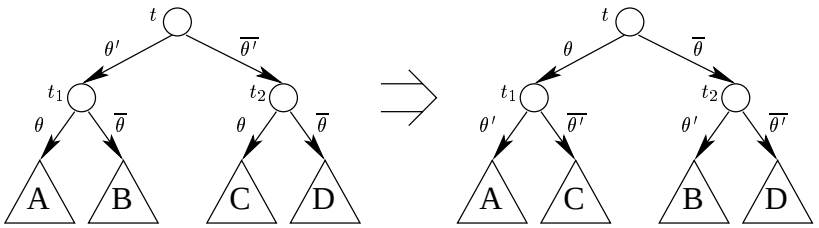
As a second step, the nodes are transposed as shown in the picture in Fig. 5, such that θ is now at the top of the tree. The crucial point is here, that we have to ensure that the information memorized on every node of the tree is correct after this whole process. In the first step, this property is ensured either by ADDEXAMPLE, or by the recursion. (More formally, in the proof of the correctness of TRANSPOSE this serves as the induction assumption.) As the reader can verify, the information memorized on the nodes of the subtrees A, B, C, D does not change because of the transposition since the set conditions on the path to the tree root does not change; neither does the memorized information at t. Unfortunately, the nodes t_1 and t_2 need further consideration, because both get new children.

Our scheme of memorizing the values needed to calculate the heuristic has the advantage that we usually do not need to go through all learned examples when restructuring a tree. We want to keep that advantage in this case as well. Here, the choice of the

TRANSPOSE$(t, \theta, \mathcal{T})$:

parameters: A fuzzy decision tree t and a condition θ.
implicit parameters: test selection heuristic $\mathcal{H}$ and pruning criterion $\mathcal{P}$
returns: A fuzzy decision tree, where the edges of the top node are labelled with θ and $\overline{\theta}$.

- If t is an inner node with edges marked θ and $\overline{\theta}$ return t.
- For each child node t' of the root node of t do:
  - If t' is a leaf, replace it by an empty tree of depth 1 with edges labelled θ and $\overline{\theta}$ and add the examples memorized on t' one by one with ADDEXAMPLE,
  - else $t' :=$ TRANSPOSE$(t', \theta, \mathcal{T} \setminus \{\theta\})$.
- Swap subtrees and conditions in t as follows:



- Recalculate memorized values at t_1 resp. t_2 as sum of memorized values at A, C resp. B, D.
- If any of A, B, C and D is an empty leaf, remove it and replace its parent node by its sibling.

Fig. 5. Algorithm TRANSPOSE.

algebraic t-norm pays off, because the memorized values at t_1 resp. t_2 are just the sum of the memorized values at the roots of subtrees A and C resp. B and D. For example:

$$\begin{aligned}
|\lambda_A| + |\lambda_C| &= |\lambda_{t_1} \cap \theta'| + |\lambda_{t_1} \cap \overline{\theta'}| \\
&= \sum_{e \in \mathcal{U}} \top(\lambda_{t_1}(e), \theta'(e)) + \sum_{e \in \mathcal{U}} \top(\lambda_{t_1}(e), 1 - \theta'(e)) \\
&= \sum_{e \in \mathcal{U}} \big[\lambda_{t_1}(e) * \big(\theta'(e) + 1 - \theta'(e)\big)\big] \\
&= |\lambda_{t_1}|
\end{aligned}$$

Summing up, a formal description of TRANSPOSE is given in Fig. 5.

Last, let us consider the last procedure, OPTIMIZE. Basically, it recursively traverses the tree in preorder, checks whether nodes are optimal (i.e. the test condition is the one chosen by the heuristic) and calls, if needed, TRANSPOSE to change the test conditions. Furthermore, it prunes subtrees if they fulfill the pruning condition. Note that the calculation of the heuristic in each node is computationally much less expensive than in FID because it can rely on the data memorized at the nodes of the incremental fuzzy decision tree. Fig. 6 gives the algorithm in more detail.

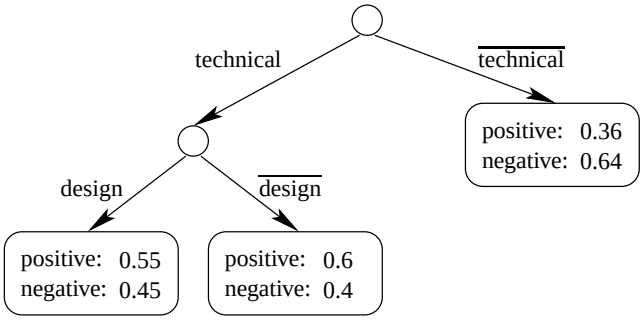
OPTIMIZE$(t, \mathcal{T})$:

parameters: A incremental fuzzy decision tree t, and a set of conditions $\mathcal{T}$.
implicit parameters: test selection heuristic $\mathcal{H}$ and pruning criterion $\mathcal{P}$.
returns: A fuzzy decision tree, where the conditions on every edge are exactly the ones chosen by the heuristic, and the pruning criteria are met at the leafs, and only at the leafs.

- If t is a leaf or $\mathcal{T} = \emptyset$ return t.
- If t meets pruning criteria $\mathcal{P}$, return leaf annotated with data calculated from t.
- If node is marked *dirty*
  - Let θ be the condition chosen from $\mathcal{T}$ for this node by the heuristic $\mathcal{H}$,
  - $t :=$ TRANSPOSE$(t, \theta, \mathcal{T} \setminus \{\theta\})$,
  - clear the *dirty* mark of the root node of t.
- Return t with all child nodes t' of the root node of t replaced by OPTIMIZE$(t', \mathcal{T} \setminus \{\theta\})$.

Fig. 6. Algorithm OPTIMIZE.

Example (1 continued). The tree generated by ADDEXAMPLE for $e_1, \ldots, e_4$ is not optimal: the test heuristic recommends the test technical at the top. OPTIMIZE$(t, \mathcal{T})$ therefore modifies the tree. Due to the better choice of the test conditions, one node in the tree is pruned. Thus, the following tree represents the users interests more compactly.



7 Properties of the algorithms

We are mainly concerned with two properties, namely termination and correctness of our algorithms. We conclude with the proof, that FID and the incremental algorithm suite yield equivalent decision trees under suitable conditions.

Theorem 1. *The algorithms* ADDEXAMPLE, TRANSPOSE *and* OPTIMIZE *terminate.*

Sketch of proof. In the case of ADDEXAMPLE the proof is done by induction on the size of the finite set of available tests $\mathcal{T}$ in the classification task: this size decreases

on every recursive call. The termination of TRANSPOSE can be proven by induction on the tree height: each recursive call is done on a tree of decreased height. The proof of termination for OPTIMIZE is analogous to the proof of ADDEXAMPLE, but bases on the termination properties of the other algorithms. □

The next step is to verify that the algorithms produce incremental fuzzy decision trees that correctly represent the set of training examples.

Proposition 1. *The algorithms* ADDEXAMPLE, TRANSPOSE *and* OPTIMIZE *are correct.*

The proof of this proposition involves a detailed discussion of the steps of all algorithms and the calls between the algorithms. Due to lack of space we cannot present details here.

Our intention for the use of the algorithm suite is the following. Suppose a tree t is constructed by adding examples with ADDEXAMPLE to an empty tree and optimizing it with OPTIMIZE. Then t should be equivalent to the tree t' generated by FID from the same set of examples in the sense that t has the same structure and carries all the labels of t'. This ensures that CLASSIFY yields the same results for both trees. Of course, t carries some more data on each node for bookkeeping reasons outlined in Section 6, but this does not need to concern us here, because these additional data are not used by CLASSIFY.

Because in most practical application of incremental algorithms one wants to interlace the classification of some examples and the addition of new training examples, we allow for optimization of the tree using OPTIMIZE in between calls to ADDEXAMPLE. Formally, this tree construction procedure is defined as follows:

Definition 1. *A tree t is **constructed and optimized** from a sequence of examples $\langle e_1, e_2, \ldots e_m \rangle$ if it can be constructed by the following procedure:*

- *Initialize t with an empty leaf.*
- *For each i in $\{1, \ldots, m\}$ do:*
  - ADDEXAMPLE$(t, e_i, 1, \chi, \mathcal{T})$
  - *Nondeterministically perform* $t :=$ OPTIMIZE$(t, \mathcal{T})$.
- $t :=$ OPTIMIZE$(t, \mathcal{T})$.

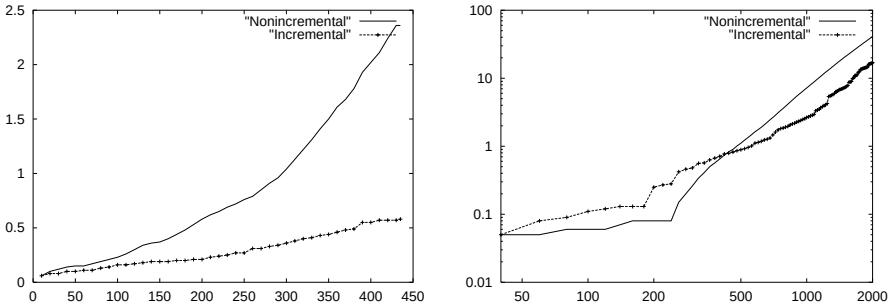
The learning scheme introduced in Section 5 is an instance of this definition.

Now we can proceed to our final theorem proving the equivalence of FID and our suite of algorithms with regard to the results.

Theorem 2. *Let χ be a classification function, $\mathcal{T}$ a finite set of conditions and t a tree constructed and optimized from a sequence of examples $\langle e_1, e_2, \ldots e_m \rangle$. Let $\lambda = \{(e_i, 1) \mid i = 1, \ldots, m\}$. Then, t is equivalent to the tree constructed by* FID $(\lambda, \chi, \mathcal{T})$.

Sketch of proof. It is easy to see that OPTIMIZE returns a tree in which every node is optimal, i.e. has the test condition chosen by the given heuristic. One can show by induction over the tree structure that for each node t we have that the subtree below t is equivalent to the result of FID $(\lambda_t, \chi, \mathcal{T}_t)$, where λ_t is defined in (1), and $\mathcal{T}_t$ is $\mathcal{T}$ reduced by the conditions that label the edges on the path from the tree root to t.

Thus, the whole tree is equivalent to FID $(\lambda, \chi, \mathcal{T})$. □



(a) (b)

Fig. 7. Time behavior for our incremental algorithm in comparison to FID on the house votes database [1] (a), and simulated user behavior data (b). The diagram shows the computation time in seconds needed to learn the examples over the number of examples learned resp. classified.

8 Discussion

In this paper we have presented a suite of algorithms for incremental induction of fuzzy decision trees. The full description and omitted proofs can be found in [5]. In fact, [5] generalizes the results presented in this paper in two aspects: Whereas we have used only binary tests in this article, non–binary tests are allowed, and methods for handling missing attributes are discussed. Furthermore, the handling of the bookkeeping data memorized at the nodes of the tree is presented in detail.

The algorithms presented here have been developed in our project to introduce personalization into a generic Web shop system. We started out from the approach of [6] to compose a web page by automatically arranged information items, like pictures, texts, or videos. These items can be collapsed / decollapsed by user actions, providing both means to adapt the web page to user preferences, as well as means to implicitly collect data about these preferences: we take the action of collapsing resp. decollapsing as an indication of negative resp. positive interest. When the user accesses a new web page, the interest of the user in the items of that page is predicted by a learning algorithm on the basis of the previous interest indications of the user and attributes the shop operator gave each item. The states of the items are then chosen accordingly.

Using a learning algorithm in classical logic, this was implemented into the hybris jakarta e-Commerce solution [2]. However, it quickly became apparent that classical logic is too inflexible for that purpose, because simple yes/no indications whether an item has a feature are often counter–intuitive. So a solution based on the fuzzy logic learning algorithm FID was implemented. To improve performance we have been developing the presented incremental algorithm suite.

In comparison to FID the incremental algorithm has the advantage that the already learned examples do not necessarily have to be reconsidered when updating the tree. On the other hand there is a considerable amount of additional bookkeeping as discussed in Section 6. It is not too difficult to construct an example showing that in the worst case

the incremental algorithm cannot perform better than FID . However, we argue that such worst–case examples hardly ever occur in practical applications. In practice, the incremental algorithm can save a significant amount of CPU time, especially when the learned tree more or less stabilizes its structure after learning some examples. In Fig. 7 we show a comparison of the computation times of the incremental algorithm suite in comparison to the non–incremental FID . Both algorithms were employed in a learning regime as presented at the end of of Section 5. This regime is meant to resemble our discussed application. Every time a new web page is shown to an user, the information about the interest of the user in the information items of the previous is added to the information memorized about the user. Then, the items of the new web page are classified by the algorithm, and arranged accordingly. When using the non–incremental algorithm, we have to recalculate the tree in this step, while it suffices in the incremental case to add the data to the tree using the relatively inexpensive ADDEXAMPLE to update the tree, and reorganize the tree with the more expensive OPTIMIZE once in a while. Fig. 7 shows clear advantages of the incremental algorithms in our learning regime given a sufficient number of examples. In the future we hope to check the performance of our algorithms on data gathered from usage of a real web shop.

References

[1] C.L. Blake and C.J. Merz. UCI repository of machine learning databases, 1998.

[2] Sven-Erik Bornscheuer, Yvonne McIntyre, Steffen Hölldobler, and Hans-Peter Störr. User adaptation in a Web shop system. In M. H. Hamza, editor, *Proceedings of the IASTED International Conference Internet and Multimedia Systems and Applications*, pages 208–213, Anaheim, Calgary, Zurich, 2001. ACTA Press.

[3] L. Breiman, J.H. Friedman, R.A. Olshen, and C.J. Stone. *Classification and Regression Trees*. Wadsworth & Brooks Advanced Books and Software, Pacific Grove, CA, 1984.

[4] D. Dubois and H. Prade. *Fuzzy Sets and Systems: Theory and Applications*. Academic Press, New York, 1980.

[5] Marina Guetova. Inkrementelle Fuzzy–Entscheidungsbäume. Diplomarbeit, Knowledge Representation and Reasoning Group, Department of Computer Science, Technische Universität Dresden, Dresden, Germany, January 2001, (in German).

[6] Tanja Hölldobler. *Temporäre Benutzermodellierung für multimediale Produktpräsentationen im World Wide Web*. Peter Lang Europäischer Verlag der Wissenschaften, Frankfurt, 2001, (in German).

[7] Cezary Z. Janikow. Fuzzy decision trees: Issues and methods. *IEEE Transactions on Systems, Man, and Cybernetics*, 28(1):1–14, 1998.

[8] S. K. Murthy. Automatic construction of decision trees from data: a multi-disciplinary survey. *Data Mining and Knowledge Discovery*, 2:345–389, 1998.

[9] J. R. Quinlan. Induction on decision trees. *Machine Learning*, 1:81–106, 1986.

[10] J. R. Quinlan. *C4.5: Programs for Machine Learning*. Morgan Kaufmann, San Mateo, CA, 1993.

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

[12] L. A. Zadeh. Fuzzy sets. *Information and Control*, 8:407–428, 1965.

[13] Jens Zeidler. *Unscharfe Entscheidungsbäume*. PhD thesis, Technische Universität Chemnitz, Fakultät Informatik, 1999.