Top-Down Induction of Fuzzy Pattern Trees

Robin Senge and Eyke Hüllermeier
Department of Mathematics and Computer Science
University of Marburg, Germany

Draft version of a paper to appear in IEEE Transactions on Fuzzy Systems

Abstract

Fuzzy pattern tree induction was recently introduced as a novel machine learning method for classification. Roughly speaking, a pattern tree is a hierarchical, tree-like structure, whose inner nodes are marked with generalized (fuzzy) logical operators and whose leaf nodes are associated with fuzzy predicates on input attributes. A pattern tree classifier is composed of an ensemble of such pattern trees, one for each class label. This type of classifier is interesting for several reasons. For example, since a single pattern tree can be considered as a kind of logical description of a class, it is quite appealing from an interpretation point of view. Moreover, in terms of classification accuracy, the method has shown promising performance in first experimental studies.

Yet, as will be argued in this paper, the algorithm that has originally been proposed for learning fuzzy pattern trees from data offers scope for improvement. Here, we propose a new method that modifies the original proposal in several ways. Notably, our learning algorithm reverses the direction of pattern tree construction from bottom-up to top-down. Additionally, a different termination criterion is proposed that is more adapted to the learning problem at hand. Experimentally, it will be shown that our approach is indeed able to outperform the original learning method in terms of predictive accuracy.

1 Introduction

Pattern tree induction was recently introduced as a novel machine learning method for classification by Huang, Gedeon and Nikravesh [11].[1] Roughly speaking, a fuzzy pattern tree is a hierarchical, tree-like structure, whose inner nodes are marked with generalized (fuzzy) logical and arithmetic operators, and whose leaf nodes are associated with fuzzy predicates on input attributes. A pattern tree propagates information from the bottom to the top: A node takes the values of its descendants as input, combines them using the respective operator, and submits the output to its predecessor. Thus, a pattern tree implements a recursive mapping producing outputs in the unit interval. A pattern tree classifier consists of an ensemble of pattern trees, one for each class. A query instance to be classified is submitted to each tree, and a prediction is made in favor of the class whose tree produces the highest output.

Pattern trees are interesting for several reasons, especially from an interpretation point of view. Generally, each tree can be considered as a kind of logical description of a class.[2] Besides, more concrete interpretations are possible in the context of specific types of learning problems.

[1]Independently, the same type of model was proposed in [23] under the name "fuzzy operator tree".
[2]Actually, the description is not purely logical, since arithmetic (averaging) operators are also allowed.

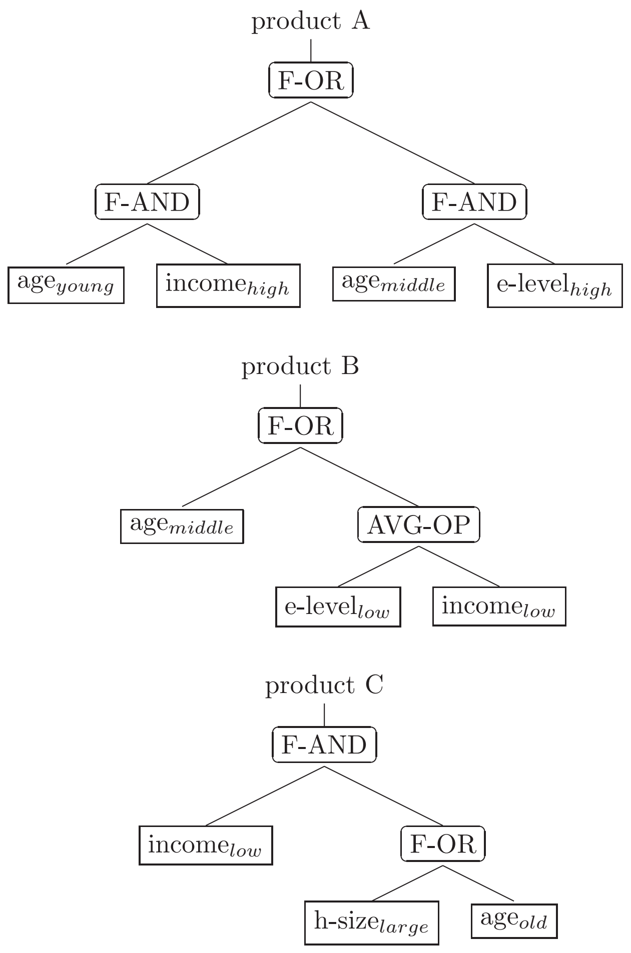

Figure 1: Pattern Tree - Marketing Example

In *preference learning* [12], for instance, where each class corresponds to a choice alternative, a tree can be seen as a utility function, measuring the degree of utility of that alternative as a function of the input. An illustration is shown in Figure 1. Assuming that this pattern tree corresponds to a certain choice alternative, say, a product A, the model suggests that this alternative is liked (receives a high degree of utility) by a person who is either young and has a high income or is middle-aged and has a high level of education. A tree of this kind could have been induced from a data set containing descriptions of persons in terms of attributes like age, income, etc., as well as information about whether product A was chosen or not. Having similar pattern trees for products B and C, the tree ensemble can be used for the purpose of choice prediction: Given a person with specific attributes, the estimated utilities of products A, B and C are derived from the respective trees, and the product with the highest degree of utility is predicted.

Even though first experimental studies, presented in [11], are promising and suggest that pattern tree classifiers perform reasonably well in terms of predictive accuracy, the original learning algorithm offers scope for improvement. In this paper, we therefore propose a new method that modifies the original proposal in several ways. Notably, our learning algorithm reverses the direction of pattern tree construction from bottom-up to top-down. Additionally, a different termination criterion is proposed that is more adapted to the learning problem at hand, taking the inherent complexity of the learning task into consideration. Experimentally, it will be shown that our approach is indeed able to outperform the original learning method in terms of predictive accuracy.

The remainder of the paper is structured as follows. Section 2 gives a brief introduction to pattern tree classification and recalls the essential background, namely the structure of the classification model and the original learning algorithm for tree induction as proposed in [11]. Besides, we shall highlight and discuss some properties of pattern trees and some basic insights that motivated the development of our new learning algorithm. This algorithm is then described in detail in Section 3. Section 4 is devoted to experimental studies evaluating and comparing the two algorithms in terms of predictive accuracy. The paper ends with a summary and some concluding remarks in Section 5.

2 Pattern Tree Induction

In this section, we briefly recall the pattern tree model for classification and the original algorithm for learning such models from data; for further technical details, we refer to [11]. Subsequently, we discuss some deficiencies of this method and motivate an alternative approach to model induction.

2.1 Tree Structure and Model Components

We proceed from the common setting of supervised learning and assume an attribute-value representation of instances, which means that an instance is a vector

$$\boldsymbol{x} \in \mathbb{X} = \mathbb{X}_1 \times \mathbb{X}_2 \times \ldots \times \mathbb{X}_m \ ,$$

where $\mathbb{X}_i$ is the domain of the i-th attribute A_i. Each domain $\mathbb{X}_i$ is discretized by means of a fuzzy partition, that is, a set of fuzzy subsets

$$F_{i,j} : \mathbb{X}_i \to [0,1] \qquad (j = 1, \ldots, n_i)$$

such that $\sum_{j=1}^{n_j} F_{i,j}(\boldsymbol{x}) > 0$ for all $\boldsymbol{x} \in \mathbb{X}_i$. The $F_{i,j}$ are often associated with linguistic labels such as "small" or "large", in which case they are also referred to as *fuzzy terms.* Each instance is associated with a class label

$$y \in \mathbb{Y} = \{y_1, y_2, \ldots, y_k\} \ .$$

A training example is a tuple $(\boldsymbol{x}, y) \in \mathbb{X} \times \mathbb{Y}$.

Unlike decision trees [17], which assume an input at the root node and output a class prediction at each leaf, pattern trees process information in the reverse direction. The input of a pattern tree is entered at the leaf nodes. More specifically, a leaf node is labeled by an attribute A_i and a fuzzy subset $F_{i,j}$ of the corresponding domain $\mathbb{X}_i$. Given an instance $\boldsymbol{x} = (x_1, \ldots, x_m) \in \mathbb{X}$ as an input, the node produces $F_{i,j}(x_i)$ as an output, that is, the degree of membership of x_i in $F_{i,j}$. This degree of membership is then propagated to the parent node.

Internal nodes are labeled by generalized logical or arithmetic operators, including

- t-norms and t-conorms [13],

name	definition	code
Minimum	$\min\{a,b\}$	MIN
Algebraic	ab	ALG
Lukasiewicz	$\max\{a+b-1,0\}$	LUK
Einstein	$\frac{ab}{2-(a+b-ab)}$	EIN

Table 1: Fuzzy operators: T-norms

name	definition	code
Maximum	$\max\{a,b\}$	MAX
Algebraic	$a+b-ab$	COALG
Lukasiewicz	$\min\{a+b,1\}$	COLUK
Einstein	$\frac{a+b}{1+ab}$	COEIN

Table 2: Fuzzy operators: T-conorms

- weighted and ordered weighted average [18, 22].

A t-norm is a generalized conjunction, namely a monotone, associative and commutative $[0,1]^2 \to [0,1]$ mapping with neutral element 1 and absorbing element 0. Likewise, a t-conorm is a generalized disjunction, namely a monotone, associative and commutative $[0,1]^2 \to [0,1]$ mapping with neutral element 0 and absorbing element 1. The t-norms and t-conorms allowed as operators in pattern tree induction are shown in Tables 1–2.

An ordered weighted average (OWA) combination of k numbers $v_1, v_2, \ldots, v_k$ is defined by

$$\mathrm{OWA}_w(v_1, v_2, \ldots, v_k) \overset{\mathrm{df}}{=} \sum_{i=1}^{k} w_i \cdot v_{\tau(i)}, \tag{1}$$

where τ is a permutation of $\{1,2,\ldots,k\}$ such that $v_{\tau(1)} \leq_{\tau(2)} \leq \ldots \leq v_{\tau(k)}$ and $w = (w_1, w_2, \ldots, w_k)$ is a weight vector satisfying $w_i \geq 0$ for $i = 1,2,\ldots,k$ and $\sum_{i=1}^{k} w_i = 1$. Thus, just like the normal weighted average (WA), an OWA operator is parameterized by a set of weights. However, a weight does not directly refer to an attribute, like in WA, but instead to a rank: w_i is the weight of the i-th smallest value among $v_1, v_2, \ldots, v_k$.

Note that for $k = 2$, (1) is simply a convex combination of the minimum and the maximum. In fact, the minimum and the maximum operator are obtained, respectively, as the two extreme cases of (1): $w_1 = 1$ yields $\mathrm{OWA}_w(v_1, \ldots, v_k) = v_{\tau(1)} = \min(v_1, \ldots, v_k)$ and $w_k = 1$ gives $\mathrm{OWA}_w(v_1, \ldots, v_k) = v_{\tau(k)} = \max(v_1, \ldots, v_k)$. Therefore, the class of OWA operators nicely "fills the gap" between the largest conjunctive combination, namely the minimum t-norm, and the smallest disjunctive combination, namely the maximum t-conorm.

The results of the evaluations of internal nodes are propagated to the parents of these nodes in a recursive way. The output eventually produced by a pattern tree is given by the output of its root node. Fig. 2 shows some exemplary pattern trees.

A pattern tree *classifier* is a collection of pattern trees

$$\{\, \mathrm{PT}_i \mid i = 1,2,\ldots,k \,\} \ ,$$

where PT_i is the pattern tree associated with class $y_i \in \mathbb{Y}$. Given a new instance $\boldsymbol{x}$ to be classified, a prediction is made in favor of the class whose tree produces the highest score:

$$\widehat{y} \,=\, \arg\max_{y_i \in \mathbb{Y}} \mathrm{PT}_i(\boldsymbol{x}) \tag{2}$$

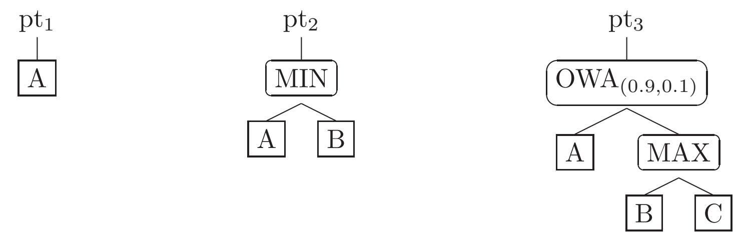

Figure 2: Pattern Tree Examples

A single tree PT_i can also be seen as a "fuzzy" selector of its class, hence the name pattern tree. Like a regular expression, a pattern tree selects instances belonging to its class, albeit in a fuzzy way. Eventually, the class is determined by the pattern tree that is most "confident" of being representative for the instance.

2.2 Learning Pattern Trees from Data

The use of the pattern tree model for classification requires an algorithm for learning such models from data, i.e., from a set of examples

$$\mathcal{T} = \{(\boldsymbol{x}^{(i)}, y^{(i)})\}_{i=1}^{n} \subset \mathbb{X} \times \mathbb{Y} \ .$$

Following the original proposal of [11], pattern trees are built one by one, independently of each other. For each class, the induction method performs as follows:

1. Initialize with primitive pattern trees
2. Filter candidates by evaluation of their similarity to the target class
3. Check stopping criterion
4. Recombine candidates using fuzzy operators
5. Loop at step 2

During initialization, so-called "primitive pattern trees" are created. These consist of only a single leaf node, i.e., a single fuzzy term $F_{i,j}$. The first set of candidate trees is built by creating one such primitive pattern tree for each fuzzy term of each attribute.

In the second step, each candidate tree is evaluated in terms of a performance measure, namely its "similarity" to the target class y_j. Note that, ideally, the output of the pattern tree PT_j is given by the indicator function

$$\mathrm{PT}_j(\boldsymbol{x}) = \begin{cases} 1 & \text{if } \boldsymbol{x} \text{ belongs to } y_j \\ 0 & \text{otherwise} \end{cases}$$

More generally, a tree will make predictions in the unit interval, which can be considered as membership degrees of a fuzzy subset B of the training data: $B(\boldsymbol{x}^{(i)}) = \mathrm{PT}_j(\boldsymbol{x}^{(i)})$ for all training instances $\boldsymbol{x}^{(i)}$. This fuzzy subset can then be compared to the true subset of examples of class y_j, namely the set A_j given by $A_j(\boldsymbol{x}^{(i)}) = 1$ if $y^{(i)} = y_j$ and $A_j(\boldsymbol{x}^{(i)}) = 0$ otherwise. To this end, Huang et al. propose to use similarity measures of fuzzy sets, such as the generalized Jaccard coefficient:

$$\begin{aligned} Sim_{JAC}(A_j, B) &= \frac{|A_j \cap B|}{|A_j \cup B|} \\ &= \frac{\sum_{i=1}^{n} \min(A_j(\boldsymbol{x}^{(i)}), B(\boldsymbol{x}^{(i)}))}{\sum_{i=1}^{n} \max(A_j(\boldsymbol{x}^{(i)}), B(\boldsymbol{x}^{(i)}))} \end{aligned} \tag{3}$$

Of course, other measures can in principle be used as well, for example distance-based measures like the following one, which is inversely related to the root mean squared error (RMSE) between the respective fuzzy sets:

$$Sim_{RMSE}(A_j, B) = 1 - \sqrt{\frac{\sum_{i=1}^{n} (A_j(\boldsymbol{x}^{(i)}) - B(\boldsymbol{x}^{(i)}))^2}{|\mathcal{T}|}} \ . \tag{4}$$

Having evaluated all candidate trees in terms of their similarity to the target A_j, only those trees with a sufficiently high degree of similarity are retained and considered within the next steps of the algorithm.

In step 4, all possible combinations of two trees from the current pool, sticked together by one of the allowed fuzzy operators, are created as new candidates. This is done by taking the operator as a root node and the two trees as subtrees. All these candidates are used for the next iteration, starting at step 2. The algorithm stops if one of two stopping criteria is met. The first criterion is satisfied if none of the newly created candidates improves the maximum similarity found so far. The second criterion sets a maximum tree depth for candidate trees, thereby also limiting the number of iterations of the algorithm.

Along the way, we note that the choice of the weighted average or the ordered weighted average operators also involves the selection of an optimal weight vector; since trees are binary, the vector is of length 2, and hence determined by a single parameter. An optimal value for this parameter can be found analytically [11].

2.3 Discussion

As mentioned previously, the original learning procedure as outlined above offers scope for improvement. In the following, we pick up two specific aspects that might be of interest in this regard, namely the basic structure of the induction algorithm and the criteria used for stopping the iterative procedure. In both cases, we motivate modifications of the original approach, to be realized later on in Section 3.

2.3.1 Bottom-Up Induction

As explained above, pattern tree induction seeks to create a (fuzzy) logical representation for each class in an iterative way. This is done in a bottom-up manner by repeatedly combining two (of the currently best) candidate trees with a new root. The bottom-up strategy can indeed be motivated intuitively. For example, it can be seen as an iterative combination and construction of complex features, corresponding to trees and subtrees, from basic features, given by the original attributes.

Nevertheless, we believe that a top-down (PTTD) instead of a bottom-up (PT) construction of pattern trees is an interesting alternative. Instead of merging two trees into a completely new tree, being much bigger and having a quite different structure, the idea is to modify the

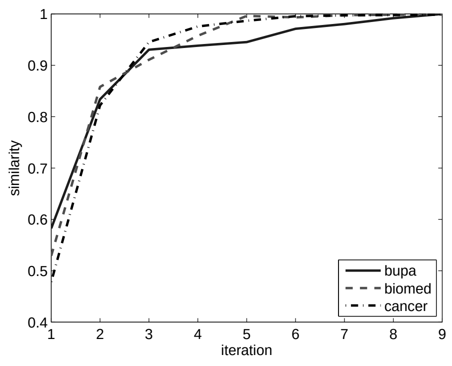

Figure 3: Pairwise similarity between candidate models (first averaged over all pairs of candidates, then over the two classes) for three binary data sets: biomed, bupa, cancer (see also Table 4).

current tree only slightly. This can be done by expanding one of its leaf nodes, namely by replacing a basic feature $F_{i,j}$ with a compound feature (two basic features, one of them being $F_{i,j}$, combined by a single operator). Thus, new operators are introduced at the bottom of the tree and not on the top, which normally means that later operators have a smaller influence on the input-output behavior than those that were chosen earlier. We shall explore this idea in more detail in Section 3 below.

To help understanding the main differences between bottom-up and top-down induction, and to highlight potential advantages of the latter, it is appealing to compare them with operators used in genetic algorithms (even though this analogy is admittedly not perfect): While the merging of two pattern trees into a new tree in bottom-up induction can be seen as a kind of *recombination* operator (combining two solutions into a new one), the expansion of a leaf node in top-down induction is more in line with a *mutation* (modifying a single solution). Now, it is well-known that recombination can be quite beneficial, at least if the two solutions are complementary. However, to actually reach an optimum once being close to it, the mutation operator is indispensable. In particular, by making only small steps in the search space, it allows one to fully explore this space, whereas a recombination alone does not. Indeed, by combining complete candidate trees, bottom-up induction tends to make “large jumps” in the search space (“genotype space”). For example, the combination operator may double the number of nodes of the candidate trees in each iteration. As opposed to this, top-down induction implements a more fine-grained exploration of the search space, trying a large number of small steps in each iteration.

Besides, we also found another problem of the bottom-up approach, namely a lack of diversity. In fact, despite strong differences on the structural (“genotype”) level, it turns out that, after only a few iterations, all candidate trees tend to be very similar to each other in the sense of implementing very similar $\mathbb{X} \rightarrow [0, 1]$ mappings (i.e., the trees are similar on the “phenotype” level). Consequently, a combination of two such trees will remain almost ineffective. Fig. 3 illustrates this problem by plotting the average pairwise similarity between two candidate models, in terms of the measure (4), as a function of the number of iterations.

Against the background of these considerations, one may expect that bottom-up induction is more likely to get trapped in local optima or to find suboptimal solutions, whereas top-down induction is more often able to reach a real optimum. Our experimental results are indeed

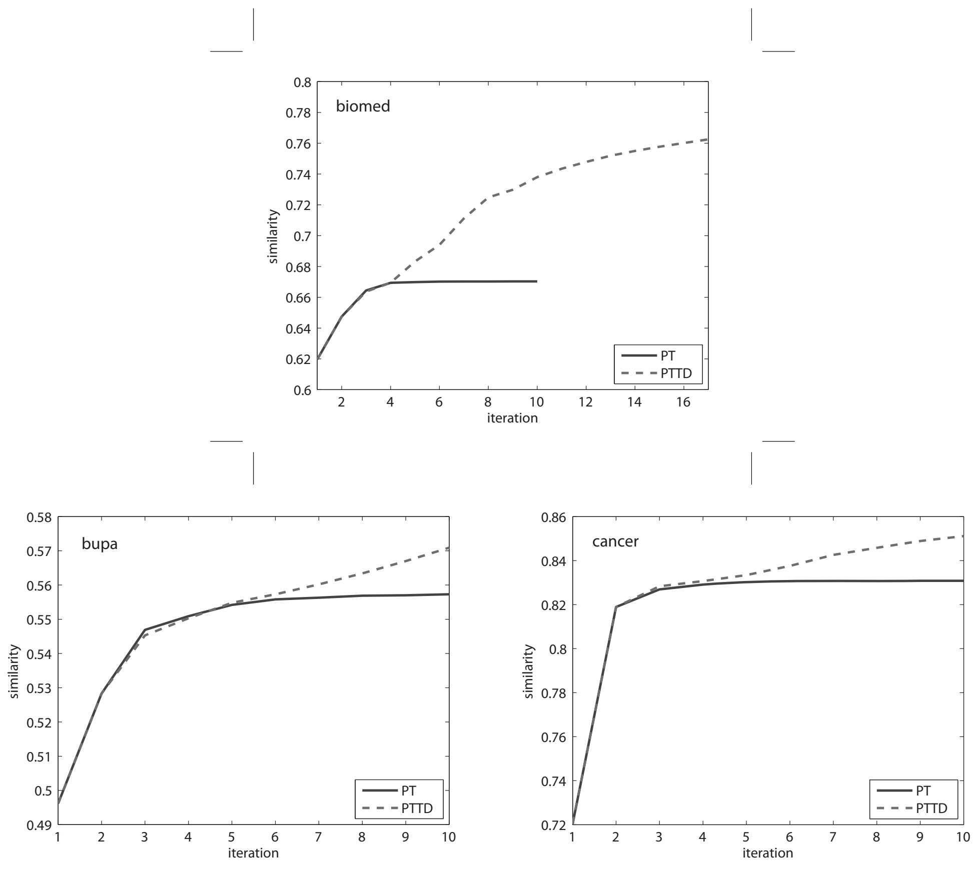

Figure 4: Performance curve (top-down strategy in sold, bottom-up strategy in dashed line, averaged over the classes) on three data sets: biomed, bupa, cancer (see also Table 4).

in agreement with this conjecture. As an illustration, Fig. 4 shows some typical performance curves of the two approaches, plotting the performance of the best model (on the training data) in each iteration against the number of the iteration (note that the curves may have different lengths due to different termination criteria; see Section 2.3.2 below). As can be seen, the two approaches produce quite similar results at the beginning, where the trees are very small; of course, a difference between top-down and bottom-up induction cannot be expected at this stage. However, while bottom-up induction reaches a saturation level quite quickly and apparently converges to a suboptimal solution, top-down induction is still able to achieve improvements of the solution.

2.3.2 Termination Criteria

Recall that two termination criteria are used in the original pattern tree induction algorithm. The first criterion is satisfied if no improvement is achieved, i.e., if the tree with the highest similarity sim_{max}^{t} in iteration t could not be improved by the highest similarity sim_{max}^{t+1} in the following iteration $t+1$. Thus, if the condition $sim_{max}^{t} \geq sim_{max}^{t+1}$ holds, the induction process is stopped and the best tree of iteration t is chosen to be the final tree for the respective class.

This is a standard stopping condition commonly used in iterative optimization procedures.[3] The second termination criterion is satisfied if a candidate tree reaches a certain depth. Note that, since candidate trees grow by one level in each iteration, this criterion is equivalent to limiting the number of iterations, and hence the runtime of the algorithm.

To get an idea of the relevance of the two termination criteria, we conducted an experimental study in which we applied the pattern tree classifier to a number of data sets and checked which of the two termination criteria was responsible for stopping the algorithm. More specifically, as proposed in [11], we used pattern trees with a maximum tree depth of five (PT5). This classifier was applied ten times to each of the forty data sets that will also be used for our experiments later on; see Table 4 in Section 4. As a result, we found that the algorithm terminates due to the maximum-depth criterion in almost 90% of the cases, whereas the no-improvement condition is responsible in only 10%.

Despite being the more important stopping condition, or perhaps rather because of that, the maximum-depth criterion can be criticized. Obviously, it does not take the complexity of the learning task or, more specifically, the decision boundaries separating the classes in the input space, into account. In fact, while some classes can be characterized quite easily with a small pattern tree, more complex trees might be needed for other classes. Thus, it is clear that the maximum-depth criterion comes with the danger of stopping either too early or too late.

More specifically, looking again at the performance curves in Fig. 4, it seems that the original pattern tree learner may actually stop too late: Given that the improvements in later iterations are negligible (in fact hardly visible graphically), the algorithm could stop earlier, thereby producing smaller models of the same quality.[4] For the top-down induction, on the other hand, the condition is prone to premature stopping, thereby preventing this method from finding an optimal solution. To avoid this deficiency, we shall propose an *adaptive* termination criterion, that is, a criterion which seeks to adapt the model complexity to the difficulty of the learning task in an optimal way.

3 Top-Down Induction of Pattern Trees

Based on the above discussion of possibilities to improve the original pattern tree learner, concrete modifications will be proposed in this section. First, our basic algorithm will be introduced in Section 3.1, detailing the ideas of learning pattern trees in a top-down manner and using an adaptive, problem-specific termination criterion. In Sections 3.2 and 3.3, we address two other problems for which no concrete solution was offered by Huang et al., namely the specification of a fuzzy partition for a numeric attribute and the choice of the similarity measure for evaluating candidate trees. Finally, we also propose a strategy for handling the problem of missing values.

3.1 The Basic Algorithm

Our algorithm for learning pattern trees in a top-down manner, PTTD, follows a one-versus-rest scheme [3], just like the method of Huang et al. Thus, the original k-class classification problem involving classes $\mathbb{Y} = \{y_1, \ldots, y_k\}$ is decomposed into k binary problems, one for each

[3] In general, this criterion can be criticized for being myopic. A standard way to improve it is to use a kind of lookahead search.

[4] Following the principle of Occam's razor [5], these smaller trees should be preferred.

Top-down Algorithm

```
1: {Initialization}
2: $\mathbf{P} = \{A_{ij}\}, i = 1, ..., n; j = 1, ..., m$
3: $\mathbf{C}^0 = argmaxB_{P \in \mathbf{P}}[Sim(P, X_0)]$
4: $\epsilon = 0.0025$
5: $t = 0$
6: {Induction}
7: {Loop on iterations}
8: while true do
9:   $t = t + 1$
10:  $\mathbf{C}^t = \mathbf{C}^{t-1}$
11:  {Loop on each candidate}
12:  for all $C_i^{t-1} \in \mathbf{C}^{k-1}$ do
13:    {Loop on each leaf of the chosen candidate}
14:    for all $l \in leafs(C_i^{t-1})$ do
15:      {Loop on each available operator $\psi$}
16:      for all $\psi \in \Psi$ do
17:        {Loop on nearly each primitive pattern tree}
18:        for all $P \in \mathbf{P} \backslash l$ do
19:          $\mathbf{C}^t = \mathbf{C}^t \cup ReplaceLeaf(C_i^{t-1}, l, \psi, P)$
20:        end for
21:      end for
22:    end for
23:  end for
24:  $\mathbf{C}^t = argmaxB_{C_i^t \in \mathbf{C}^t}[Sim(C_i^t, X_0)]$
25:  $sim_{max}^t = max_{C_i^t \in \mathbf{C}^t}(Sim(C_i^t, X_0))$
26:  $sim_{max}^{t-1} = max_{C_i^{t-1} \in \mathbf{C}^{t-1}}(Sim(C_i^{t-1}, X_0))$
27:  if $sim_{max}^t < (1 + \epsilon) sim_{max}^{t-1}$ then
28:    break
29:  end if
30: end while
31: return $argmax_{C_i^t \in \mathbf{C}^t}[Sim(C_i^t, X_0)]$
```

Figure 5: Top-down algorithm

class $y_j \in \mathbb{Y}$. In the j-th problem, y_j is considered as the positive class (for which the sought model prediction is 1) and $\mathbb{Y} \setminus \{y_j\}$ as the negative class (for which the target is 0).

The basic algorithm for solving a single binary problem (class y_j versus all other classes) is presented in pseudo-code in Fig. 5. It implements a beam search and maintains the B best models so far ($B = 5$ is used as a default value).

The algorithm starts by initializing the set of all primitive pattern trees $\mathbf{P}$. A primitive tree is a tree that consists of only one node, labeled by a fuzzy term. Additionally, the first candidate set, $\mathbf{C}^0$, is initialized by the B best primitive pattern trees, i.e., the trees being maximally similar to the target.

After initialization, the algorithm iterates over all candidates trees. Starting from line 11, it

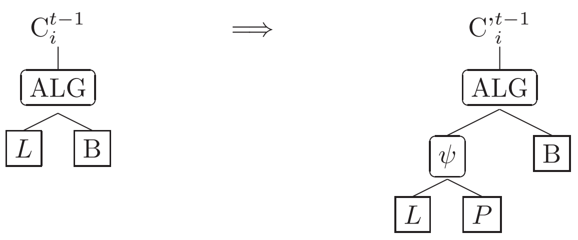

Figure 6: Top-down induction: A leaf node is expanded through replacement with a three-node tree.

seeks to improve the currently selected candidate C_i^{t-1} in terms of similarity to the target. To this end, new candidates are created by tentatively replacing exactly one leaf node L (labeled by a fuzzy term) of C_i^{t-1} by a new subtree. This new subtree is a three-node pattern tree that again contains L as one of its leaf nodes (see Fig. 6 for an illustration). The new candidate tree thus obtained is then evaluated by computing its similarity to the target class. Having tried all possible replacements of all leaf nodes of the trees in $\mathbf{C}^i$, the B best candidates are selected and passed to the next iteration, unless the termination criterion is fulfilled.

In order to lower the overall runtime of the algorithm, all possible three-node trees can be precomputed in advance and the outputs they produce can be stored for later use. Additionally, an implementation should also consider to recalculate the output of an adapted candidate tree in the following way: Starting from the substitute three-node tree, only the nodes on the path from that subtree to the root node of the candidate need to be recalculated.

To make the termination criterion adaptive and problem-specific (cf. Section 2.3.2), our idea is to look for the *relative* improvement of the best model in the t-th iteration as compared to the $(t-1)$-st iteration. More specifically, our algorithm stops if

$$sim_{max}^{t} < (1+\epsilon) sim_{max}^{t-1} \ , \tag{5}$$

i.e., if the relative improvement is smaller than ϵ, where $\epsilon \in (0,1)$ is a user-defined parameter. Based on empirical evidence, we propose $\epsilon = 0.25\%$ as a suitable value for this parameter (see Section 4). Noting that the performance measures sim_{max}^{t} $(t = 1, 2, \ldots)$ form a monotone increasing sequence upper-bounded by 1, the above termination criterion appears to be reasonable. In fact, as shown in Fig. 4, when plotting sim_{max}^{t} against t, one typically obtains a curve with a concave shape, strictly increasing at the beginning but flattening and converging toward a saturation level after a while.

3.2 Fuzzy Partitions

To make pattern tree learning amenable to numeric attributes, these attributes have to be "fuzzified" and discretized beforehand. Fuzzification is needed because fuzzy logical operators at the inner nodes of the tree expect values between 0 and 1 as input, while discretization is needed to limit the number of candidate trees in each iteration of the learning algorithm. Besides, fuzzification may also support the interpretability of the model.

Fuzzy partitions can of course be defined in various ways. In our implementation, we discretize a domain $\mathbb{X}_i$ using three fuzzy sets $F_{i,1}, F_{i,2}, F_{i,3}$ associated, respectively, with the terms "low", "medium" and "high". The first and the third fuzzy set are defined as

$$F_{i,1}(x) = \begin{cases} 1 & x < min \\ 0 & x > max \\ 1 - \frac{x - min}{max - min} & otherwise \end{cases},$$

$$F_{i,3}(x) = \begin{cases} 1 & x > max \\ 0 & x < min \\ \frac{x - min}{max - min} & otherwise \end{cases},$$

with *min* and *max* being the minimum and the maximum value of the attribute in the training data. Noting that all operators appearing at inner nodes of a pattern tree are monotone increasing in their arguments, it is clear that these fuzzy sets can capture two types of influence of an attribute on the class membership, namely a positive and a negative one: If the value of a numeric attribute increases, the membership of the "high"-term of that attribute also increases (positive influence), whereas the membership of the "low"-term decreases (negative influence).

Apart from monotone dependencies, it is of course possible that a non-extreme attribute value is "preferred" by a class. The fuzzy set $F_{i,2}$ is meant to capture dependencies of this type. It is defined as a triangular fuzzy set with center c:

$$F_{i,2}(x) = \begin{cases} 0 & x \leq min \\ \frac{x - min}{c - min} & min < x \leq c \\ 1 - \frac{x - c}{max - c} & c < x < max \\ 0 & x \geq max \end{cases} \tag{6}$$

The parameter c is determined so as to maximize the absolute (Pearson) correlation between the membership degrees of the attribute values in $F_{i,2}$ and the corresponding class information (encoded by 1 for instances belonging to the class and 0 for instances of other classes) on the training data. In case the correlation is negative, $F_{i,2}$ is replaced by its negation $1 - F_{i,2}$.

Finally, nominal attributes are modeled as degenerated fuzzy sets: For each value v of the attribute, a fuzzy set with membership function

$$Term_v(x) = \begin{cases} 1 & x = v \\ 0 & otherwise \end{cases}$$

is introduced.

3.3 Similarity Measures

As mentioned previously, there are several ways to measure the similarity between two fuzzy sets or, more generally, two $[0, 1]$-valued functions. From a theoretical point of view, it is difficult to anticipate which of these measures might be especially suitable. Since Huang et al. do not give any clear recommendation either, we tried to answer this question experimentally.

More specifically, we conducted a number of experiments in which we compared similarity measures of different type in terms of classification performance, using the same forty data sets that we shall also use later on in Section 4. Apart from the measures (3) and (4), a number of other measures have been tested, including, for example,

$$Sim_{ME}(A,B) = 1 - \frac{\sum_{i=1}^{n} |A_j(\boldsymbol{x}^{(i)}) - B(\boldsymbol{x}^{(i)})|}{|\mathcal{T}|}$$

$$Sim_{MSigE}(A,B) = 1 - \frac{\sum_{i=1}^{n} e(A(\boldsymbol{x}^{(i)}) - B(\boldsymbol{x}^{(i)}))}{|\mathcal{T}|} \tag{7}$$

where the sigmoid-shaped error function $e(\cdot)$ is given by the mapping $v \mapsto (1 + e^{-8(v-0.5)})^{-1}$.

Table 3: Win/Loss of different similarity measures

	Win	Loss
Sim_{RMSE}	25	15
Sim_{MSigE}	6	34
Sim_{ME}	5	35
Sim_{JAC}	4	36

Table 3 summarizes the finding in terms of a win-loss-statistic. The results are clearly in favor of Sim_{RMSE} that performed best on 25 of the 40 data sets (while the second-best, Sim_{MSigE}, was only 6 times the winner). Our explanation for the strong performance of Sim_{RMSE} is the convex shape of the quadratic error function. In particular, convexity implies that larger deviations from the target prediction (1 for positive and 0 for negative examples) are punished disproportionately high, whereas small deviations are less critical. In connection with the "arg max" classification rule (2), this appears to be quite reasonable. This classification rule is indeed robust toward small deviations from the target prediction. In fact, it will predict the correct class as long as the errors $|A_j(\boldsymbol{x}^{(i)}) - B(\boldsymbol{x}^{(i)})|$ are all smaller than 1/2.

As a potential disadvantage of the Sim_{RMSE} measure, note that the quadratic error function, like all convex losses, is quite sensitive toward outliers: If an instance cannot be classified correctly, it may yield a disproportionately high error, thereby devaluing a model with otherwise strong performance. It was mainly for this reason that we also tried the measure (7) with a sigmoid-shaped error function. The idea here is to pay special emphasis on errors around 1/2, since this point is critical for the correctness of the classification rule (2), whereas less distinction is needed for errors close to 0 or close to 1. A sigmoid-shaped function, which is a smooth version of the step function jumping from 0 to 1 at 1/2, does obviously comply with these requirements, so that Sim_{MSigE} appears advantageous from this point of view. On the other hand, there are also arguments against the expectation that Sim_{MSigE} may outperform Sim_{RMSE}:

- In contrast to problems like regression, the squared error is upper-bounded (namely by 1), which means that the sensitivity toward outliers is less severe.
- One should note that a devaluation caused by a true outlier will probably apply to all candidate models in more or less the same way. In other words, it is unlikely to change the relative performance, and hence the selection, of a pattern tree in comparison to other candidates.
- A sigmoid-shaped loss function is arguably less suitable for handling pseudo-outliers, i.e., instances for which the current model has a high error even though they are not true outliers: A small improvement on such instances, moving their score a bit to the correct direction, will hardly be rewarded, thereby preventing the right model adaptation.

As can be seen, there are arguments both in favor and against a loss function like (7). Anyway, our experiments have clearly shown that, regarding performance, the arguments against seem to dominate. Based on this empirical evidence, our recommendation is to use the Sim_{RMSE} measure (4). Specifically, we shall do so in the experimental studies presented in the Section 4 below.

3.4 Handling Missing Values

Since most real-world data sets are incomplete, an important ability of machine learning methods is a proper handling of missing attribute values. In the context of pattern tree induction, this means that a pattern tree learner should be able to make a prediction for an instance $\boldsymbol{x}$ for which some of the attribute values x_i are unknown.

To this end, we propose a method that is motivated by the observation that a pattern tree implements a monotone function of the fuzzy input features, i.e., the fuzzified input values $f_i = F_{i,j}(\boldsymbol{x})$: Since all operators used in a pattern tree are monotone increasing in their arguments, the function implemented by the tree itself is monotone increasing, too. Thus, even though the true output cannot be determined for an incomplete instance $\boldsymbol{x}$, it is possible to derive a lower bound y^l and an upper bound y^u. The former is obtained by setting all unknown fuzzy features to 0, while the latter is produced by the tree when all these features are set to 1. Given the bounds y^l and y^u computed in this way, we suggest to predict the output $(y^l + y^u)/2$, which minimizes expected loss under the assumption of a uniform distribution on the interval $[y^l, y^u]$.

4 Experiments

This section presents a number of experimental studies that we conducted to get an idea of the performance of our pattern tree learner. First of all, we were of course interested in a comparison with the original approach due to Huang et al. Besides, however, we also compared our method to other state-of-the-art classifiers from the field of machine learning.

4.1 Data Sets

The data sets used in all experiments are shown in Table 4. The table provides the number of instances (#instance), the number of classes (#classes), the number of numeric attributes (#num) and the number of nominal attributes (#nom) per data set. The 40 data sets have been collected from the UCI [4] and the STATLIB [14] repositories. Since the original method of Huang et al. does not handle missing values, we only selected data sets without missing values.[5]

4.2 Methods

We included two variants of the original pattern tree algorithm in the experiments, PT5 (maximum tree depth of 5) and PT10 (maximum tree depth of 10); these variants were also used in [11]. Likewise, we included two variants of the top-down pattern trees, namely PTTDE.5 and PTTDE.25. These two variants only differ with respect to the parameter ϵ of our new

[5]Actually, some of the data sets do contain a few missing values; the corresponding records were simply deleted.

Table 4: Properties of the data sets used in the experiments.

Data set	#instances	#classes	#num	#nom
analcatdata-braziltourism	411	6	4	4
analcatdata-cyyoung8092	97	2	7	3
analcatdata-germangss	400	4	1	4
analcatdata-homerun	163	2	13	14
analcatdata-lawsuit	264	2	3	1
australian	690	2	6	9
authorship	841	4	69	1
autos	205	6	15	10
balance-scale	625	3	4	0
biomed	209	2	7	1
blood	748	2	4	1
bupa	345	2	6	0
cancer	683	2	9	1
cars	406	3	6	1
cloud	108	4	6	1
cmc	1473	3	2	8
confidence	72	6	3	0
credit	690	2	6	10
fl2000	65	3	14	2
flag	194	8	17	11
flare2	1065	7	0	10
german	1000	2	7	14
glass	214	6	9	0
haberman	306	2	3	1
heart	270	2	7	7
ionosphere	351	2	34	1
iris	150	3	4	1
irish	500	2	2	3
lupus	87	2	3	0
lymphography	148	4	3	15
metStatRST	336	12	3	0
pima	768	2	8	0
primary-tumor	339	22	0	17
prnn-crabs	200	2	6	1
prnn-synth	250	2	2	0
schizo	340	2	12	2
sonar	208	2	60	0
vehilce	846	4	18	1
wine	178	3	13	1
zoo	101	7	1	15

termination criterion (5). PTTDE.5 is parameterized with $\epsilon = 0.5\%$, whereas PTTDE.25 uses $\epsilon = 0.25\%$. As an evaluation criterion, we used the similarity measure (4) for both top-down and bottom-up learning. Moreover, for all variants, a beam size of 5 and the same fuzzy partitions for input attributes were used, namely those described in Section 3.2.

Recall that our new pattern tree learner differs from the original one with regard to two properties, namely the direction of tree construction (top-down versus bottom-up) and the termination criterion. In order to differentiate the effects produced by these two modifications, we included yet another variant of the top-down pattern tree learner, namely PTTDL5. This variant does not use the adaptive termination criterion (5) but the two standard termination criteria of the original algorithm, namely the no-improvement and the maximum-depth (with limit 5) conditions.

Finally, we included a number of well-known classification methods from the field of machine learning as baselines: The C4.5 decision tree learner (C4.5) [17], nearest neighbor classification (NN) [1], support vector machines with linear kernel[6] (SVM) and the RIPPER (Repeated Incremental Pruning to Produce Error Reduction) rule learner [6]. All these algorithms are implemented in the WEKA Machine Learning Framework [21], and we used these implementations with their default parameterization.[7] Additionally, we included the fuzzy rule learner SLAVE [9, 8]. An implementation of this algorithm is offered by the KEEL suite [2], another machine learning framework. We ported this implementation to WEKA.[8] Our own implementations of the pattern tree learners have also been integrated in WEKA.[9]

4.3 Accuracy

In the first experimental study, we measured the classification rate (percentage of correct predictions) of the different methods, using five repetitions of a 10-fold cross validation. The results (mean accuracy with standard deviation in brackets) are summarized in Table 5 and Table 6.

In order to verify that the top-down induction strategy, even without the adaptive termination criterion, is able to improve upon the original bottom-up strategy, we first compared PTTDL5 and PT5. To this end, we applied a Wilcoxon Signed-Rank Test [20]. This is a non-parametric test that ranks the differences in performances of two classifiers for each data set, ignoring the signs, and compares the ranks for the positive and the negative differences. For the results in Table 5 and Table 6, the test rejects the null hypothesis of equal performance at the 5% significance level (the p-value is 0.00017).

Thus, one can conclude that a top-down induction strategy is significantly better than a bottom-up strategy, a finding that is furthermore confirmed by a simple sign test (PTTDL5 wins 30 times and PT5 only 10 times).

Next, we evaluated the effect of our adaptive termination criterion. To this end, we compared PTTDL5 with PTTDE.25. Again, the Wilcoxon test finds a significant difference in performance (at the significance level of 5%), this time in favor of PTTDE.25 or, in other words, in favor of our adaptive stopping condition (the p-value is 0.033). In terms of wins and losses, PTTDE.25 wins 23 times, loses 15 times and ties 2 times.

Finally, we compared our pattern tree learner PTTDE.25 with the benchmark classifiers SVM, NN, RIPPER, C4.5 and SLAVE. To compare multiple classifiers with a control classifier, García et. al. [7] suggests the use of a two step procedure. First, the non-parametric Quade test [16] is conducted in order to test the null-hypothesis of equal performance of all classifiers. In case this hypothesis is rejected, i.e., if there are statistically significant differences between the methods, the Holm post hoc test [10] is used to analyze the differences between PTTDE.25 and the benchmark classifiers in a pairwise manner. Both tests are based on the ranks of the methods: For each data set, the methods are sorted according to their performance, i.e., each method is assigned a rank (in case of ties, average ranks are assigned). After that, the ranks are weighted

[6]Implemented by John Platt's sequential minimal optimization (SMO) procedure [15].

[7]The following parameters are used in WEKA: C4.5: -C 0.25 -M 2; RIPPER: -F 3 -N 2.0 -O 2 -S 1; NN: -K 3 -W 0 -A LinearNNSearch -A "EuclideanDistance -R first-last"; SMO: -C 1.0 -L 0.0010 -P 1.0E-12 -N 0 -V -1 -W 1 -K "PolyKernel -C 250007 -E 1.0"

[8]We used the following parameter setting: 5 fuzzy sets, 500 iterations without change, mutation probability 0.01, use weights, population size 100.

[9]The implementation is available at `http://www.uni-marburg.de/fb12/kebi/research/`

Table 5: Average classification rates and standard deviation on test sets. Part I

Dataset	SVM	NN	RIPPER	C4.5	SLAVE
analcatdata-braziltourism	**78.25(1.82)**	75.23(4.14)	76.4(2.25)	76.2(3.28)	77.04(6.63)
analcatdata-cyyoung8092	77.47(9.39)	74.96(9.46)	79.4(11.79)	**80.8(9.16)**	75.78(13.63)
analcatdata-germangss	19.4(5.62)	7.4(3.64)	38.65(6.6)	37.1(5.8)	25(5.71)
analcatdata-homerun	88.15(7.61)	51.82(11.19)	95.53(6.09)	**95.57(6.01)**	74.98(10.77)
analcatdata-lawsuit	93.87(1.99)	97.49(2.7)	97.73(2.75)	**98.26(2.55)**	93.48(5.48)
australian	84.84(3.65)	85.48(4.01)	85.74(4.01)	**85.86(4.05)**	85.51(3.9)
authorship	**99.67(0.63)**	99.52(0.79)	94.08(2.41)	94.24(2.49)	76.74(5.58)
autos	72.25(10.6)	68.48(10.61)	72.98(9.8)	**80.3(9.93)**	28.22(11.88)
balance-scale	**87.78(2.16)**	86.91(2.71)	80.6(3.54)	77.89(3.52)	68.4(6.54)
biomed	86.61(6.83)	**90.06(6.25)**	89.86(5.71)	88.01(8.69)	78.17(9.87)
blood	76.15(0.47)	74.92(3.57)	**77.86(3.33)**	77.64(3.44)	76.01(4.62)
bupa	58.04(1.15)	62.67(8)	67.09(7.47)	66.38(7.7)	56.98(6.13)
cancer	**97.07(2)**	96.87(2.13)	96.16(2.45)	95.64(2.53)	89.55(4.42)
cars	73.3(5.84)	72.76(4.93)	78.81(4.98)	**84.68(5.36)**	63.36(6.68)
cloud	29.67(9.79)	21.56(9.82)	50.65(16.68)	**71.69(17.73)**	17.85(11.72)
cmc	48.56(3.93)	46.8(3.05)	52.54(3.48)	51.57(3.53)	42.7(3.63)
confidence	64.57(11.66)	83.54(10.3)	75.39(14.55)	77.86(12.95)	63.86(21.44)
credit	85.01(3.7)	85.94(3.82)	85.3(3.34)	85.68(3.69)	85.3(4.37)
fl2000	73.71(8.96)	81.33(12.58)	80.05(13.41)	81.29(15.16)	71.71(16.41)
flag	61.37(10.36)	54.87(10.67)	61.65(8.32)	64.73(9.47)	25.01(10.93)
flare2	**83.01(0.36)**	81.9(1.14)	82.74(0.76)	82.93(0.35)	83.01(3.33)
german	**75.08(3.53)**	72.06(3.28)	72.2(4.23)	71.38(2.7)	70.18(4.54)
glass	57.81(7.92)	**70.12(6.84)**	66.59(9.58)	69.05(8.01)	62.61(11.42)
haberman	73.46(1.01)	69.74(4.48)	72.04(5.61)	71.12(5.3)	73.93(7.55)
heart	**83.85(5.96)**	78.59(6.08)	79.85(6.25)	78.44(6.4)	71.78(8.09)
ionosphere	88.16(4.99)	86.33(4.85)	88.9(4.85)	89.63(4.8)	80.92(6.04)
iris	**96.27(4.84)**	95.2(5.5)	93.47(8.69)	94.93(5.43)	92.13(8.16)
irish	**98.8(1.6)**	98.72(1.59)	**98.8(1.6)**	**98.8(1.6)**	98.12(1.78)
lupus	75.61(14.1)	74.81(16.66)	75.44(12.42)	**77.53(12.07)**	75.75(16.57)
lymphography	**85.91(7.45)**	81.14(8.28)	75.36(12.18)	75.7(11.62)	69.27(12.47)
metStatRST	27.62(4.65)	39.42(6.69)	36.48(6.86)	39.95(7.84)	8.99(6.02)
pima	**76.85(4.25)**	74.24(4.65)	75.62(4.71)	74.85(5.35)	66.3(5.01)
primary-tumor	**46.37(5.68)**	39.65(5.4)	39.41(5.42)	40.59(5.64)	24.83(6.85)
prnn-crabs	**96(4)**	95.5(4.5)	90.6(6.61)	93.7(4.88)	61.7(12.4)
prnn-synth	**85.6(6.74)**	86.32(7.25)	83.68(7.37)	84.48(7.14)	79.52(8.07)
schizo	59.24(5.85)	64.29(9.3)	**81.24(8.3)**	80.76(6.66)	50.47(7.82)
sonar	76.27(8.95)	**83.36(8.46)**	73.12(10.12)	73.86(8.46)	63.48(13.44)
vehilce	**74.37(3.89)**	71.03(3.84)	68.53(4.35)	71.92(4.79)	29.47(4.59)
wine	**99(2.41)**	95.97(4.03)	93.58(7.13)	93.59(5.42)	81.23(8.94)
zoo	**93.65(6.53)**	92.49(7.56)	88.93(8.73)	92.49(7.01)	86.98(11.01)

respectively by the difference between the minimum and the maximum performance values for each dataset. The average weighted ranks are shown in Table 7.

In our case, the Quade test rejects the null-hypothesis at a significance level of 5%. The p-value computed is quite persuasive at 2.7E-16. The results of the pairwise comparisons between PTTDE.25 and the baseline classifiers using the Holm test are summarized in Table 8. As can be seen, our PTTDE.25 learner shows the best performance on average, though not all differences are statistically significant. Significant differences are only found for SLAVE.

Summarizing this experimental study, we can conclude that our pattern tree learner PTTDE.25 is at least competitive to state-of-the-art classifiers from the field of machine learning, while being significantly stronger than the original algorithm for pattern tree induction. In particu-

Table 6: Average classification rates and standard deviation on test sets. Part II

Dataset	PTTDE.5	PTTDE.25	PTTDL5	PT5	PT10
analcatdata-braziltourism	76.01(2.1)	75.82(2.48)	76.25(2.3)	76.6(1.91)	76.5(2.22)
analcatdata-cyyoung8092	77.6(9.6)	75.96(11.16)	79.56(10.26)	79.33(10.94)	77.96(10.47)
analcatdata-germangss	39.4(6.11)	39.5(6.42)	**40.3(6.34)**	37.25(6.13)	36.5(6.67)
analcatdata-homerun	85.51(8.46)	90.95(7.83)	88.83(9.02)	83.9(8.11)	83.4(8.17)
analcatdata-lawsuit	94.92(2.72)	94.77(2.4)	94.92(2.37)	95.52(2.82)	95.83(2.88)
australian	85.51(3.59)	85.57(3.6)	85.51(3.59)	85.62(3.58)	85.39(3.68)
authorship	98.36(1.11)	99.07(0.9)	96.62(1.83)	94.63(3.03)	95.65(2.61)
autos	72.82(9.66)	74.65(9.51)	69.36(9.26)	65.88(9.43)	71.32(8.4)
balance-scale	87.14(3.92)	87.29(3.32)	86.05(3.46)	85.43(3.09)	85.43(3.09)
biomed	87.05(7.89)	88.5(6.82)	86.37(7.54)	85.62(7.3)	85.62(7.36)
blood	76.93(1.8)	77.03(1.84)	77.33(1.85)	76.98(1.42)	77.01(1.47)
bupa	70.74(7.49)	70.69(7.04)	**71.37(7.74)**	68.71(8.25)	69.36(7.4)
cancer	96.22(2.45)	96.55(2.19)	96.78(2.25)	95.9(2.33)	96.22(2.49)
cars	72.21(4.43)	72.6(4.35)	72.41(4.5)	71.08(4.09)	70.98(4.37)
cloud	42.31(14.55)	41.25(14.41)	42.33(15.42)	42(15.38)	40.84(14.47)
cmc	54.92(3.37)	**55.49(3.29)**	55.17(3.07)	53.86(4)	53.93(3.39)
confidence	86.5(9.43)	87.11(9.26)	**87.64(9.37)**	83.96(9.84)	83.14(10.96)
credit	85.51(3.75)	85.42(3.68)	**85.51(3.75)**	85.62(3.78)	85.39(3.78)
fl2000	**87.67(12.19)**	83.24(11.73)	87.62(11.06)	74.52(12.83)	79.95(12.4)
flag	67.41(8.63)	**68.29(7.47)**	66.36(7.58)	67.36(8.2)	65.3(9.2)
flare2	82.72(0.78)	82.74(0.75)	82.67(0.74)	82.82(0.63)	82.84(0.63)
german	73.38(2.81)	74.28(3.35)	73.64(3.25)	73.52(3.28)	73.38(3.01)
glass	64.4(8.8)	67.51(8.34)	64.78(8.64)	60.83(7.3)	62.98(8.37)
haberman	74.37(6.29)	74.64(5.85)	**74.9(5.89)**	74.44(5.68)	74.44(5.88)
heart	82.07(6.08)	82.59(5.51)	82.07(5.85)	82.59(5.75)	81.78(6.01)
ionosphere	91.28(5.4)	**91.97(4.87)**	91.68(5.04)	88.49(5.39)	88.84(5.42)
iris	95.6(5.6)	95.6(5.6)	95.6(5.6)	95.33(5.7)	95.07(6.23)
irish	97.68(2.09)	98.68(1.86)	97.28(2.15)	97.64(2.14)	97.64(2.14)
lupus	75.08(12.5)	77(12.85)	75.67(13.08)	77.44(12.28)	77.44(12.28)
lymphography	82.41(7.25)	83.8(7.61)	82.29(7.73)	79.46(9)	79.87(8.73)
metStatRST	37.33(5.94)	38.11(6.89)	38.82(6.41)	38.77(5.56)	**40.56(5.05)**
pima	76.3(5.22)	76.49(4.92)	76.54(4.93)	75.75(5.11)	75.68(4.99)
primary-tumor	42.01(5.33)	43.18(6.13)	43.48(6.48)	41.83(5.7)	42.3(6.23)
prnn-crabs	75.2(11.22)	77.5(11.01)	76.7(11.21)	73.5(10.11)	73.5(10.11)
prnn-synth	83.84(6.88)	83.76(6.9)	83.76(7.08)	83.6(6.85)	83.6(6.85)
schizo	66.94(7.82)	67.59(7.34)	66.65(7.12)	64.29(7.3)	66.29(7.28)
sonar	80.69(8.73)	80.38(8.92)	74.86(8.48)	70.81(9.85)	69.56(10.41)
vehilce	69.34(3.84)	71.47(3.39)	66.76(4.17)	63.05(4.73)	64.37(4.4)
wine	98.2(3.06)	98.31(2.81)	97.19(3.75)	93.35(5.93)	94.25(5.72)
zoo	90.25(7.02)	91.07(6.65)	90.45(7.15)	92.67(7.89)	93.47(6.73)

Table 7: Average Rankings of the algorithms (Quade)

Algorithm	Rank
PTTDE.25	2.4902
C4.5	2.6932
SVM	3.0792
RIPPER	3.3939
NN	3.6567
SLAVE	5.6865

Table 8: Holm test results for a significance level of 5% based on Quade ranks comparing to PTTDE.25.

	Algorithm	p-Value	Reject
5	SLAVE	2.54E-5	yes
4	NN	0.1243	no
3	RIPPER	0.2338	no
2	SVM	0.4377	no
1	C4.5	0.7890	no

lar, both modifications, the top-down induction strategy and the use of an adaptive stopping criterion, seem to pay off and improve predictive performance.

4.4 Model Size and Runtime

In addition to measuring the accuracy of each method, we also looked at the runtime of the algorithms and the size of the resulting models. Table 9 shows the average results over all datasets for the different pattern tree induction methods. The runtime has been measured in seconds needed to train a model. The measurements have been conducted on the same machine type (Dual Opteron 270 2.0GHz) without any additional work load. The model size is given by the average number of nodes contained in one fuzzy pattern tree.

Table 9: Average and standard deviation of runtime (seconds) and model size (#nodes) for all fuzzy pattern tree induction variants.

	Runtime	Model Size
PT5	138.1(478.4)	17.9(5.8)
PT10	1495.6(6466.9)	105.0(106.1)
PTTDL5	63(174.6)	11.2(2.5)
PTTDE.5	511.9(2058.2)	11.4(8.7)
PTTDE.25	2564.8(8562)	21.8(17.9)

Summarizing the results, it can be seen that the top-down strategy tends to be more costly than the bottom-up approach, at least for trees of larger depth. In fact, if the tree size is limited by 5, it is even the other way around: PTTDL5 is the most efficient among all variants, and even faster than PT5. However, for larger candidate trees, the top-down approach needs to check a large number of alternative modifications, and this slows down the search process.

In terms of the model size, the top-down approach is clearly superior. Recalling the different nature of the underlying search strategies, this was to be expected. In fact, by combining complete candidate trees, the bottom-up approach tends to create unnecessarily large models. Thus, the top-down approach produces models that are not only more accurate but also more compact.

4.5 Overfitting

As one of the advantages of their approach, Huang et al. emphasize the fact that the learning method does apparently not tend to overfit the training data. In order to demonstrate this property, they compare the performance of the learner on the training data and the test data, and found that the former is not much higher than the latter.

Since we modified the original algorithm in several ways, notably by using a different termination criterion, an obvious question is whether or not this robustness toward overfitting is preserved by our learner. For this reason, we repeated the same kind of experiment. Before presenting the results, however, we like to point out that the experiment itself should be considered with caution. Even though it is true that a small training error coming along with a high test error normally indicates an overfitting effect, the difference between these two errors alone does actually not, as it loses information about the absolute error rates. Consequently, an overfitting effect cannot be distinguished from an underfitting effect. For example, although a learner with 10% training and 20% test error seems to overfit a bit, it is still preferrable to a learner with 30% training and test error, simply because the latter seems to strongly underfit the training data. (In particular, note that the majority classifier, which always predicts the most frequent class, is likely to have a very similar training and test error.) Seen from this point of view, one may indeed argue that the truly relevant measure is the performance on the test data, which has already been presented above.

Despite these reservations, a comparison of training and test error is of course not useless and may provide some interesting information. Table 10 shows the differences between the average accuracy of a method on the training data and on the test data (determined by means of a 10-fold cross validation, repeated five times, which means that the latter corresponds to the results shown in Table 5 and Table 6).

As can be seen, SLAVE and PTTDE.5 both perform quite similar and achieve the best results on average, followed by PTTDE.25. In fact, the results are even better than those for the linear SVM, which is almost a bit surprising, given that linear models, due to their simplicity, are more likely to underfit than to overfit the data. Overall, the results are quite plausible in the sense of being in agreement with the flexibility of the methods. In particular, the nearest neighbor classifier, which is able to produce quite complex decision boundaries and is knowingly prone to overfitting, performs quite poorly. The comparison of the pattern tree learners is plausible, too: The larger the trees (due to a relaxed stopping condition), the worse the performance becomes. Worth mentioning in this regard is the poor performance of PT10. Actually, this result is in contrast to what was reported in [11], albeit on the basis of a much smaller number of data sets. Yet, it is perhaps not very surprising, given that pattern trees of depth 10 are indeed rather complex models.

5 Concluding Remarks

In this paper, we have proposed a new learning algorithm for fuzzy pattern tree induction, a machine learning method that was recently introduced in [11]. Apart from some minor modifications and extensions, including the fuzzification of numeric attributes, the evaluation of candidate models, and the handling of missing values, two major changes of the original learning algorithm have been proposed: First, pattern trees are induced in a top-down instead of a bottom-up manner, and second a dynamic termination criterion is used instead of a less flexible condition that simply limits the depth of a tree.

Experimentally, we have shown that our new algorithm, PTTD, significantly outperforms the hitherto existing method in terms of predictive accuracy and, moreover, is at least competitive to state-of-the-art classifiers from the field of machine learning. Apart from the mere results, we have also offered several explanations of these improvements, for example in terms of the search strategy realized by the two approaches: Even though both methods search the space of candidate models in a greedy manner (using beam search), the top-down strategy is more fine-

Table 10: Difference in accuracy between training and test data

Dataset	SVM	NN	RIPPER	C4.5	SLAVE	PTTDE.25	PTTDE.5	PTTDL5	PT5	PT10
analcatdata-braziltourism	1.31	7.25	1.70	1.17	0.60	2.77	3.06	2.33	2.48	6.96
analcatdata-cyyoung8092	16.35	8.55	14.41	11.98	1.13	21.98	11.06	18.38	12.42	6.58
analcatdata-germangss	13.85	36.10	3.90	5.90	0.00	1.00	3.85	0.30	6.25	20.75
analcatdata-homerun	7.52	28.43	3.24	1.96	0.74	3.91	0.91	11.17	2.52	15.36
analcatdata-lawsuit	1.21	0.61	1.89	0.99	0.90	0.69	2.05	0.92	2.20	3.03
australian	1.10	5.54	3.83	3.13	0.00	0.06	0.14	0.00	0.17	5.62
authorship	0.33	0.36	5.68	3.50	2.34	0.45	3.23	3.02	2.52	3.52
autos	14.58	16.40	14.83	9.94	4.46	12.66	3.77	23.32	19.49	23.32
balance-scale	0.10	4.29	8.52	8.83	0.69	0.07	0.06	1.31	1.77	4.65
biomed	3.34	2.76	9.19	4.33	1.98	2.89	0.51	5.02	3.37	7.20
blood	0.05	7.84	1.69	2.70	0.50	0.11	0.48	0.19	0.43	3.88
bupa	0.22	20.23	17.55	6.67	2.09	2.35	2.89	3.12	4.91	15.28
cancer	0.14	1.08	2.37	2.75	0.64	0.23	0.26	1.02	0.73	1.73
cars	0.59	14.44	14.78	1.28	1.05	0.30	0.05	0.99	1.34	24.59
cloud	7.36	41.40	43.79	19.98	8.57	15.23	16.02	21.56	14.48	57.31
cmc	1.61	22.65	3.20	5.93	0.00	0.16	0.68	0.98	1.53	17.22
confidence	22.93	5.35	17.66	9.64	10.81	3.17	2.39	4.02	6.31	12.69
credit	0.93	5.51	3.25	0.17	0.48	0.09	0.14	0.00	0.03	5.62
fl2000	26.29	6.36	15.34	9.48	4.32	16.76	0.03	12.38	20.86	15.43
flag	32.96	18.84	20.82	16.20	2.03	13.67	9.39	23.33	10.99	17.69
flare2	0.00	1.58	1.61	0.08	0.00	0.73	0.28	0.81	0.19	0.17
german	3.32	13.94	4.30	7.72	1.10	0.58	0.32	2.96	1.08	12.12
glass	2.93	10.72	26.40	15.06	3.01	1.65	5.69	5.78	10.20	33.28
haberman	0.07	10.98	5.41	6.00	2.01	1.83	2.43	1.24	2.68	2.69
heart	2.44	8.44	10.52	7.11	1.78	3.33	3.11	6.81	2.96	11.19
ionosphere	3.30	4.84	9.11	1.54	0.24	3.76	1.25	6.33	1.82	10.88
iris	0.40	1.47	4.53	3.07	1.11	0.40	0.40	1.07	0.67	2.93
irish	0.00	0.48	0.00	0.00	0.19	0.12	0.28	1.52	0.24	1.16
lupus	0.25	6.80	3.87	1.78	2.51	2.29	3.08	1.34	0.72	1.87
lymphography	8.68	11.42	20.58	12.14	9.14	10.79	2.05	15.01	5.00	13.38
metStatRST	0.54	24.57	9.36	14.22	1.02	4.16	4.93	0.77	3.19	30.87
pima	0.63	11.57	4.97	3.93	1.24	0.08	0.91	0.03	1.20	8.44
primary-tumor	18.23	15.22	12.51	3.95	0.96	6.37	6.08	9.91	11.56	22.30
prnn-crabs	1.00	3.50	7.90	1.30	2.97	2.50	0.70	4.80	1.00	24.50
prnn-synth	0.00	7.68	3.92	3.12	1.20	1.04	0.96	1.04	1.20	4.80
schizo	0.47	18.94	15.53	12.76	6.04	2.71	0.71	7.18	4.24	30.18
sonar	11.23	8.46	25.92	18.45	2.15	15.30	0.56	23.70	12.84	28.52
vehilce	1.87	14.19	15.75	6.45	0.68	0.52	3.97	8.54	3.62	32.55
wine	0.44	1.22	5.86	4.72	3.65	1.69	2.70	2.81	3.84	4.62
zoo	6.35	2.56	6.12	1.57	5.64	4.97	7.77	5.59	7.33	5.54
Average	5.37	10.81	10.05	6.29	2.25	4.08	2.73	6.01	4.76	13.76

grained and explores a larger number of search states. Regarding the termination condition, it goes without saying that an adaptive criterion is reasonable, as it can reduce the danger of overfitting (due to late stopping) or underfitting (due to premature stopping) the training data.

Regarding predictive performance, we also like to mention the results of a previous study in which we tried to learn pattern trees using evolutionary algorithms, or, more specifically, a co-evolution strategy [19]. In this approach, one population of pattern trees is maintained for each class, and the evolution of these populations is coordinated by means of a global fitness function. Despite a significantly higher search effort, the improvements in comparison to the original pattern tree learner were only small, and in fact even smaller than the improvements achieved by the top-down induction method presented in this paper. We interpret this result as evidence in favor of the search strategy underlying PTTD: Despite its greedy nature, it performs as well or even better than an evolutionary search method, while being computationally much more efficient.

As mentioned previously, the model class of fuzzy pattern trees is interesting for several reasons, not only with regard to predictive performance in classification. For example, it is user-friendly in the sense of being almost parameter-free.[10] Moreover, it is quite appealing from an interpretation point of view, for example in the field of preference learning, where a pattern tree represents a (latent) utility function of a choice alternative. Besides, the model class of pattern trees has specific features that make it attractive for other types of learning problems. As an example, we mention the learning of *monotone* classifiers, that is, classifier that guarantee monotonicity in certain attributes. Exploring the use of pattern trees for these and related purposes is a topic of ongoing and future work. Besides, another important issue to be addressed in future work concerns the reduction of the runtime complexity of pattern tree induction. Finally, it will also be interesting to consider other search strategies, for example a combination of top-down and bottom-up induction.

A Java implementation of our PTTD algorithm, running under the open-source machine learning framework WEKA, can be downloaded in the Internet.[11]

References

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

[2] J. Alcala-Fdez, L. Sánchez, S. García, M.J. del Jesus, S. Ventura, J.M. Garrell, J. Otero, C. Romero, J. Bacardit, V.M. Rivas, J.C. Fernández, and F. Herrera. Keel: A software tool to assess evolutionary algorithms for data mining problems. *Soft Computing*, 13(3):307–318, 2009.

[3] E.L. Allwein, R.E. Schapire, and Y. Singer. Reducing multiclass to binary: a unifying approach for margin classifiers. *The Journal of Machine Learning Research*, 1:113–141, 2001.

[4] A. Asuncion and D. Newman. Uci machine learning repository, 2009. Accessed 13 Nov 2009.

[5] Anselm Blumer, Andrzej Ehrenfeucht, David Haussler, and Manfred K. Warmuth. Occam's razor. *Information Processing Letters*, 24:377–380, 1987.

[6] W.W. Cohen. Fast effective rule induction. In *Proc. ICML-95, 12th International Conference on Machine Learning*, pages 115–123, Tahoe City, CA, 1995.

[7] Salvador Garca, Alberto Fernndez, Julin Luengo, and Francisco Herrera. Advanced non-parametric tests for multiple comparisons in the design of experiments in computational intelligence and data mining: Experimental analysis of power. *Information Sciences*, 180:2044 – 2064, 2010.

[10]PTTD has two parameters, the beam size B and the improvement threshold ϵ. Our experience has shown, however, that the default values $B = 5$ and $\epsilon = 0.25\%$ perform very well in general, so that there is hardly any need to use other values.

[11]`http://www.uni-marburg.de/fb12/kebi/research/`

[8] Antonio González and Raúl Pérez. Slave: A genetic learning system based on an iterative approach. *IEEE Transactions on Fuzzy Systems*, 7:176–191, 1999.

[9] Antonio González and Raúl Pérez. Selection of relevant features in a fuzzy genetic learning algorithm. *IEEE Transactions on Systems and Man and and Cybernetics and Part B: Cybernetics*, 31:417–425, 2001.

[10] Sture Holm. A simple sequentially rejective multiple test procedure. *Scandinavian Journal of Statistics*, 6:65–70, 1979.

[11] Zhiheng Huang, Tams D. Gedeon, and Masoud Nikravesh. Pattern trees induction: A new machine learning method. *IEEE Transactions on Fuzzy Systems*, 16(4):958–970, 2008.

[12] Eyke Hüllermeier, Johannes Fürnskranz, Weiwei Cheng, and Klaus Brinker. Label ranking by learning pairwise preferences. *Artificial Intelligence*, 172:1897–1917, 2008.

[13] Erich Peter Klement, Radko Mesiar, and Endre Pap. *Triangular Norms*. Kluwer Academic Publishers, 2002.

[14] M. Meyer and P. Vlachos. Statlib data, software and news from the statistics community, 2009. Accessed 13 Nov 2009.

[15] John C. Platt. Fast training of support vector machines using sequential minimal optimization. In *Advances in kernel methods: support vector learning*, pages 185–208. MIT Press, Cambridge, MA, USA, 1999.

[16] Dana Quade. Using weighted rankings in the analysis of complete blocks with additive block effects. *Journal of the American Statistical Association*, 74(367):680–683, 1979.

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

[18] B. Schweizer and A. Sklar. *Probabilistic Metric Spaces*. New York, 1983.

[19] R. Senge and E. Hüllermeier. Learning pattern tree classifiers using a co-evolutionary algorithm. In F. Hoffmann and E. Hüllermeier, editors, *Proceedings 19. Workshop Computational Intelligence*, pages 22–33, Dortmund, Germany, 2009. KIT Scientific Publishing.

[20] F. Wilcoxon. Individual comparisons by ranking methods. *Biometrics*, 1:80–83, 1945.

[21] Ian H. Witten and Eibe Frank. *Data Mining: Practical machine learning tools and techniques*. Morgan Kaufmann, 2 edition, 2005.

[22] R.R. Yager. On ordered weighted averaging aggregation operators in multi criteria decision making. *IEEE Transactions on Systems, Man and Cybernetics*, 18(1):183–190, 1988.

[23] Yu Yi, Thomas Fober, and Eyke Hüllermeier. Fuzzy operator trees for modeling rating functions. *International Journal of Computational Intelligence and Applications*, 8(4):413–428, 2008.