Pattern Trees: An Effective Machine Learning Approach

Zhiheng Huang, Masoud Nikravesh, Tamás D. Gedeon and Ben Azvine

Abstract Fuzzy classification is one of the most important applications of fuzzy logic. Its goal is to find a set of fuzzy rules which describe classification problems. Most of the existing fuzzy rule induction methods (e.g., the fuzzy decision trees induction method) focus on searching rules consisting of t-norms (i.e., AND) only, but not t-conorms (OR) explicitly. This may lead to the omission of generating important rules which involve t-conorms explicitly. This paper proposes a type of tree termed *pattern trees* which make use of different aggregations including both t-norms and t-conorms. Like decision trees, pattern trees are an effective machine learning tool for classification applications. This paper discusses the difference between decision trees and pattern trees, and also shows that the subsethood based method (SBM) and the weighted subsethood based method (WSBM) are two specific cases of pattern trees, with each having a fixed pattern tree structure.

A novel pattern tree induction method is proposed. The comparison to other classification methods including SBM, WSBM and fuzzy decision tree induction over datasets obtained from UCI dataset repository shows that pattern trees can obtain higher accuracy rates in classifications. In addition, pattern trees are capable of generating classifiers with good generality, while decision trees can easily fall into the trap of over-fitting. According to two different configurations, simple pattern trees and pattern trees have been distinguished. The former not only produce high prediction accuracy, but also preserve compact tree structures, while the latter can produce even better accuracy, but as a compromise produce more complex tree structures. Subject to the particular demands (comprehensibility or performance), simple pattern trees and pattern trees provide an effective methodology for real world applications.

Weighted pattern trees have been proposed in which certain weights are assigned to different trees, to reflect the nature that different trees may have different confidences. The experiments on British Telecom (BT) customer satisfaction dataset show that weighted pattern trees can slightly outperform pattern trees, and both are slightly better than fuzzy decision trees in terms of prediction accuracy. In addition, the experiments show that both weighted and unweighted pattern trees are robust to over-fitting. Finally, a limitation of pattern trees as revealed via BT dataset analysis is discussed and the research direction is outlined.

M. Nikravesh et al. (eds.), *Forging the New Frontiers: Fuzzy Pioneers II.*

Key words: Pattern trees · fuzzy model classification · fuzzy decision trees · machine learning · data mining

1 Introduction

The major advantage of using fuzzy rules for classification applications is to maintain transparency as well as a high accuracy rate. The difficulty in obtaining an optimal classification model is to balance the complexity of a rule base and the classification accuracy. Generally, fuzzy rule models can produce arbitrary accuracy if a unlimited number of rules are allowed [15]. However, this inevitably violates the original motivation of using fuzzy rules – transparency. To address this, many fuzzy rule induction methods have been developed. In Wang and Mendel [14] an algorithm for generating fuzzy rules by learning from examples has been presented. Yuan and Shaw [18] have proposed the induction of fuzzy decision trees for generating fuzzy classification rules (the extension of the classic decision tree induction method by Quinlan [10]). Chen, Lee and Lee [1] have presented a subsethood based method (SBM) for generating fuzzy rules. Rasmani and Shen [12] have proposed a weighted fuzzy subsethood based rule induction method (WSBM). To avoid the exponential growth of the size of the rule base when the number of input variables increases, Raju, Zhou and Kisner [11] have proposed hierarchical fuzzy systems. Recently, Kóczy, Vámos and Biró [6], and Wong, Gedeon and Kóczy [16] have presented fuzzy signatures which model the complex structure of the data points in a hierarchical manner.

Most of the existing fuzzy rule induction methods including fuzzy decision trees [18] focus on searching for rules which only use t-norm operators [13] such as the MIN and algebraic MIN. Disregarding of the t-conorms such as MAX and algebraic MAX is due to the fact that any rule using t-conorms can be represented by several rules which use t-norms only. This is certainly true and it is helpful to simplify the rule induction process by considering t-norms only. However, it may fail to generate important rules in which fuzzy terms are explicitly connected with t-conorms. This will be clearly shown in an artificial dataset in Sect. 4.6. Research has been conducted to resolve this problem. For example, Kóczy, Vámos and Biró [6] have proposed fuzzy signatures to model the complex structures of data points using different aggregation operators including MIN, MAX, and average etc. Mendis, Gedeon and Kóczy [7] have investigated different aggregations in fuzzy signatures. Nikravesh [9] has presented evolutionary computation (EC) based multiple aggregator fuzzy decision trees.

Huang and Gedeon [3] have first introduced the concept of pattern trees and proposed a novel pattern tree induction method by means of similarity measures and different aggregations. Like decision trees, pattern trees are an effective machine learning approach for classification applications. The experiments carried out on UCI datasets show that the pattern trees can obtain higher accuracy rates than the SBM, WSBM and the fuzzy decision trees in classifications. In addition, pattern trees perform more consistently than fuzzy decision trees. The former are capable

of generating classifiers with good generality, while the latter can easily fall into the trap of over-fitting.

Simple pattern trees and pattern trees [4] are distinguished with two different configurations. The former not only produce high prediction accuracy, but also preserve compact tree structures, while the latter can produce even better accuracy, but as a compromise, produce more complex tree structures. Subject to the particular demands (comprehensibility or performance), simple pattern trees or pattern trees provide an effective methodology for real world applications.

Weighted pattern trees [5] have been proposed in which certain weights are assigned to different trees. As a result, it enhances the semantic meaning of pattern trees. The experiments on BT customer satisfaction dataset show that weighted pattern trees can slightly outperform pattern trees. In fact, both weighted and unweighted pattern trees with only two or three tree levels are good enough for most experiments carried out in this paper. This provides a very transparent way to model real world applications.

The rest of the paper is arranged as follows: Sect. 2 provides the definitions for similarity, aggregations and pattern trees. Section 3 proposes a novel pattern tree induction method, which is configured differently to build simple pattern trees and pattern trees. Section 4 shows that the SBM and WSBM are two specific cases of pattern trees. It also outlines the difference between decision trees and pattern trees, and the advantage of using trees. Section 5 presents the experimental results using Saturday Morning Problem dataset and datasets from UCI machine learning repository [8]. Section 6 suggests the concept of weighted pattern trees and shows how to use them for classification. Section 7 presents the experimental results applying both weighted and unweighted pattern trees induction to BT customer satisfaction dataset. Finally, Sect. 8 concludes the paper and points out some further research work.

2 Similarity, Aggregations and Pattern Trees

2.1 Similarity

Let A and B be two fuzzy sets [19] defined on the universe of discourse U. The commonly used fuzzy similarity measures can be defined in Table 1, where $\cap$ and $\cup$ denote a certain t-norm operator and a t-conorm respectively. Usually, the MIN ($\wedge$) and MAX ($\vee$) operators are used. According to the definition, $0 \leq S(A, B) \leq 1$. Without losing generality, the most commonly used similarity measure namely Jaccard is used to construct the pattern trees in Sect. 3. In practice, it is computed as

$$S(A, B) = \frac{\sum_{j=1}^{m} [\mu_A(x_j) \wedge \mu_B(x_j)]}{\sum_{j=1}^{m} [\mu_A(x_j) \vee \mu_B(x_j)]}, \tag{1}$$

Table 1 Similarity measures

Name	Definition
Simple matching	$A \cap B$
Jaccard	$\frac{A \cap B}{A \cup B}$
Dice	$2\frac{A \cap B}{A+B}$

where x_j, $j = 1, \ldots, m$, are the crisp values discretized in the variable domain, and $\mu_A(x_j)$ and $\mu_B(x_j)$ are the fuzzy membership values of x_j for A and B.

An alternative similarity definition is proposed in this paper for pattern tree construction. Consider that the root mean square error (RMSE) of fuzzy sets A and B can be computed as

$$RMSE(A, B) = \sqrt{\frac{\sum_{j=1}^{m}(\mu_A(x_j) - \mu_B(x_j))^2}{m}}, \tag{2}$$

the RMSE based fuzzy set similarity can thus be defined as

$$S(A, B) = 1 - RMSE(A, B). \tag{3}$$

The large value $S(A, B)$ takes, the more similar A and B are. Of course, this alternative definition retains $0 \leq S(A, B) \leq 1$ given $\mu_A(x_j), \mu_B(x_j) \in [0, 1]$.

2.2 Fuzzy Aggregations

Fuzzy aggregations are logic operators applied to fuzzy membership values or fuzzy sets. They have three sub-categories, namely t-norm, t-conorm, and averaging operators such as weighted averaging (WA) and ordered weighted averaging (OWA) [17].

Triangular norms were introduced by Schweizer and Sklar [13] to model distances in probabilistic metric spaces. In fuzzy sets theory, triangular norms (t-norm) and triangular conorms (t-conorm) are extensively used to model logical operators *and* and *or*. The basic t-norm and t-conorm pairs which operate on two fuzzy membership values a and b, $a, b \in [0, 1]$ are shown in Table 2.

Table 2 Basic t-norms and t-conorms pairs

Name	t-norm	t-conorm
MIN/MAX	$min\{a, b\} = a \wedge b$	$max\{a, b\} = a \vee b$
Algebraic AND/OR	ab	$a + b - ab$
Šukasiewicz	$max\{a + b - 1, 0\}$	$min\{a + b, 1\}$
EINSTEIN	$\frac{ab}{2-(a+b-ab)}$	$\frac{a+b}{1+ab}$

Although the aggregations shown above only apply to a pair of fuzzy values, they can apply to multiple fuzzy values as they retain associativity.

Definition 1. A WA operator of dimension n is a mapping $E : \mathbb{R}^n \to \mathbb{R}$, that has an associated n-elements vector $w = (w_1, w_2, \ldots, w_n)^T$, $w_i \in [0, 1]$, $1 \leq i \leq n$, and $\sum_{i=1}^{n} w_i = 1$ so that

$$E(a_1, \ldots, a_n) = \sum_{j=1}^{n} w_j a_j. \tag{4}$$

Definition 2. An OWA operator [17] of dimension n is a mapping $F : \mathbb{R}^n \to \mathbb{R}$, that has an associated n-elements vector $w = (w_1, w_2, \ldots, w_n)^T$, $w_i \in [0, 1]$, $1 \leq i \leq n$, and $\sum_{i=1}^{n} w_i = 1$ so that

$$F(a_1, \ldots, a_n) = \sum_{j=1}^{n} w_j b_j, \tag{5}$$

where b_j is the jth largest element of the collection $\{a_1, \ldots, a_n\}$.

A fundamental difference of OWA from WA aggregation is that the former does not have a particular weight w_i associated for an element, rather a weight is associated with a particular ordered position of the element.

Example 1. Assume $w = (0.2, 0.3, 0.1, 0.4)^T$, then the WA operator on the vector of (0.6, 1, 0.3, 0.5) is

$$\begin{aligned} E(0.6, 1, 0.3, 0.5) &= 0.2 \times 0.6 + 0.3 \times 1 + 0.1 \times 0.3 \\ &\quad + 0.4 \times 0.5 = 0.65, \end{aligned}$$

and the OWA operator on the vector is

$$\begin{aligned} F(0.6, 1, 0.3, 0.5) &= 0.2 \times 1 + 0.3 \times 0.6 + 0.1 \times 0.5 \\ &\quad + 0.4 \times 0.3 = 0.55. \end{aligned}$$

It is worth noting two special OWA operators are equal to MAX and MIN:

- If $w^* = (1, 0, \ldots, 0)$, then

$$F(a_1, \ldots, a_n) = max\{a_1, \ldots, a_n\}. \tag{6}$$

- If $w_* = (0, 0, \ldots, 1)$, then

$$F(a_1, \ldots, a_n) = min\{a_1, \ldots, a_n\}. \tag{7}$$

The main factor in determining which aggregation should be used is the relationship between the criteria involved. Compensation has the property that a higher degree of

satisfaction of one of the criteria can compensate for a lower degree of satisfaction of another criterion. w^* means full compensation (*or*) and w_* means no compensation (*and*). Normally, an OWA operator lies in between these two extremes. An OWA operator with much of nonzero weights near the top will be more *or* than *and*.

2.3 Pattern Trees

A pattern tree is a tree which propagates fuzzy terms using different fuzzy aggregations. Each pattern tree represents a structure for an output class in the sense that how the fuzzy terms aggregate to predict such a class. The output class is located at the top as the root of this tree. The fuzzy terms of input variables are on different levels (except for the top) of the tree. They use fuzzy aggregations (as presented in Sect. 2.2) to aggregate from the bottom to the top (root).

Assume two fuzzy variables A and B each have two fuzzy linguistic terms A_i and B_i, $i = \{1, 2\}$, and the task is to classify the data samples to either class X or Y. The primitive pattern trees for class X are shown in Fig. 1. Each primitive tree consists of only one leaf node (fuzzy set) and it uses such a leaf node to predict the output class (X). Typically, primitive pattern trees do not lead to high prediction accuracy, and multi-level pattern trees are required to maintain satisfactory performance. In fact, multi-level trees are built via the aggregation of primitive trees. For example, the two level pattern tree for class X as shown in Fig. 2 is built using the primitive trees in Fig. 1. Let *candidate trees* be trees which begin as primitive trees and aggregate with other trees (termed *low level trees*) to generate more optimal pattern trees in terms of the similarity measure to the fuzzy values of output. A candidate tree $B_1 \Rightarrow X$ aggregates with a low level tree $A_2 \Rightarrow X$ using the *and* operator to lead to a new candidate tree $B_1 \wedge A_2 \Rightarrow X$. This new candidate tree then aggregates with another low level tree $A_1 \Rightarrow X$ using the *or* operator to generate $(B_1 \wedge A_2) \vee A_1 \Rightarrow X$ (as shown in Fig. 2). The low level trees are so called due to the fact that they always have less levels, and thus are shallower than the candidate trees. In this example, the low level trees are primitive trees only. However, it is not always the case as low level trees may be multi-level pattern trees as well (so long as they are shallower than the candidate trees).

For a classification application which involves several output classes, the worked model should have as many pattern trees as the number of output classes, with each pattern tree representing one class. When a new data sample is tested over a pattern tree, it traverses from the bottom to the top and finishes with a truth value, indicating the degree to which this data sample belongs to the output class of this pattern tree. The output class with the maximal truth value is chosen as the prediction class. For

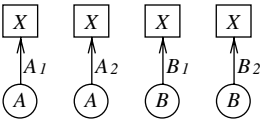

Fig. 1 Primitive pattern trees

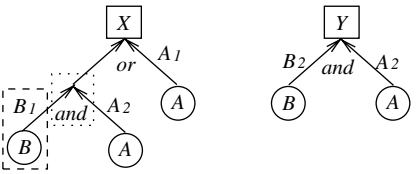

Fig. 2 Two example pattern trees

example, consider that a fuzzy data $A_1 = 0.8$, $A_2 = 0.2$, $B_1 = 0$, and $B_2 = 1$ is given for classification. As the truth values of this data over pattern trees for class X and Y are 0.8 and 0.2 respectively, X is chosen as the output class.

Conventional fuzzy rules can be extracted from pattern trees. For example, the following rules can be obtained from the pattern trees in Fig. 2.

$$Rule1 : IF\ A = A_1\ THEN\ class = X \tag{8}$$

$$Rule2 : IF\ A = A_2\ AND\ B = B_1\ THEN\ class = X \tag{9}$$

$$Rule3 : IF\ A = A_2\ AND\ B = B_2\ THEN\ class = Y \tag{10}$$

In addition to the conventionally used fuzzy aggregations MIN/MAX, pattern trees can use any aggregations as described in Sect. 2.2.

3 Proposed Pattern Tree Induction Method

Without losing generality, assume a dataset has n input variables $A_i, i = 1, 2, \ldots, n$ and one output variable B. For simplicity, further assume that both the input and output variables have m fuzzy linguistic terms denoted as A_{ij} and $B_j, i = 1, 2, \ldots, n$, and $j = 1, 2, \ldots, m$. That is, each data in the dataset is represented by a fuzzy membership value vector of dimension $(n + 1) \times m$. The task is to build m pattern trees for the m output classes (fuzzy terms). This section proposes a pattern tree induction method in pseudo code to build a pattern tree for class B_1. The pattern trees for other classes can be built following the same procedure.

Class TreeNode shows the structure of *TreeNode*, which represents leaf nodes (such as the dashed box) and internal nodes (the dotted box) in Fig. 2. *is Leaf* indicates if a node is a leaf node or an internal node. *aggreVals* stores the aggregated values on this node. Note that it stores the fuzzy values of the fuzzy set if the node is a leaf node. *similarity* is the similarity measure between *aggreVals* and the output class (B_1 in this case), and *parent* points to its parent node.

Class 1 TreeNode{

```
1:    boolean isLeaf
2:    double[] aggreVals
3:    double similarity
4:    TreeNode parent
5: }
```

Each node is further classified into either a leaf node or an internal node according to the status of *ifLeaf*. Leaf nodes and internal nodes are represented by two subclasses of *TreeNode*, namely the *LeafNode* and the *InternalNode*, as shown in classes 2 and 3 respectively. For leaf nodes, *usedAttribute* and *usedTerm* store the indices of the used fuzzy set. For instance, the use of $usedAttribute = 1$ and $usedTerm = 2$ stands for the use of fuzzy term A_{12}. For internal nodes, *aggreOpe* indicates which aggregation operator is used, *lambda* is the value associated with *aggreOpe* if *aggreOpe* = *WA* or *OWA*. *children* points to this node's child nodes.

Class 2 LeafNode extends TreeNode{

```
1:    int usedAttribute
2:    int usedTerm
3: }
```

Class 3 InternalNode extends TreeNode{

```
1:    int aggreOpe
2:    double lambda
3:    TreeNode[] children
4: }
```

Algorithm 4 is the main algorithm which constructs a pattern tree for a class. In particular, it finds and returns the root of the best pattern tree in terms of the similarity measure between the tree and the output class B_1. This algorithm takes two extra input arguments apart from the training dataset, namely the *numCandidateTrees* and *numLowLevelTrees*, indicating how many candidate trees and low level trees are used for generating optimal pattern trees. In particular, a vector consisting of *nm* primitive pattern trees is built by method *buildPrimitiveTrees()*. These primitive pattern trees are sorted in descending order according to the similarity measure to the fuzzy values of output class B_1. A copy of such vector *nodes* is made and is trimmed to size *numCandidateTrees*. In other words, the *numCandidateTrees* primitive trees which have the highest similarities are chosen to form a repository of candidate trees. $buildPatternTree()$ is then invoked to find the most optimal tree and the root node of this tree is returned.

Algorithm 5 shows how to build a pattern tree. It takes five input arguments: a vector of candidate tree, a vector of primitive trees, a vector of low level trees, the number of low level trees, and the level of the constructed tree so far. Candidate

Algorithm 4 main()

Input: A fuzzified dataset ds, int $numCandidateTrees$, and int $numLowLevelTrees$
Output: The root of the constructed pattern tree for class B_1

```
1: Vector primitiveTrees =buildPrimitiveTrees(ds);
2: primitiveTrees.sort()
3: Vector nodes = primitiveTrees.clone()
4: nodes.setSize(numCandidateTrees)
5: TreeNode root = buildPatternTree(nodes, primitiveTrees, null, numLowLevelTrees, 0)
6: return root
```

Algorithm 5 buildPatternTree()

Input: Vector *nodes*, Vector *primitiveTrees*, Vector *lowLevelTrees*, int *numLowLevelTrees*, and int *level*
Output: the root of the constructed pattern tree

```
 1: double maxSim = nodes[0]. similarity
 2: Vector candidateTrees = new Vector()
 3: Vector thisLowLevelTrees = lowLevelTrees.clone()
 4: for i = 0 to numCandiateTrees do
 5:     Vector aggregatedPrimitives = aggregate(nodes[i], primitiveTrees)
 6:     Vector aggregatedLowLevels = aggregate(nodes[i], thisLowLevelTrees)
 7:     lowLevelTrees.addAll(aggregatedPrimitives)
 8:     lowLevelTrees.addAll(aggregatedLowLevels)
 9:     candidateTrees.addAll(aggregatedPrimitives)
10:     candidateTrees.addAll(aggregatedLowLevels)
11: end for
12: lowLevelTrees.sort()
13: lowLevelTrees.removeDuplicates()
14: lowLevelTrees.setSize(numLowLevelTrees)
15: candidateTrees.sort()
16: candidateTrees.removeDuplicates()
17: candidateTrees.setSize(numCandidateTrees)
18: double newMaxSim = candiateTrees[0].similarity
19: if newMaxSim > maxSim then
20:     return buildPatternTree(candidateTrees, primitiveTrees,
        lowLevelTrees, numLowLevelTrees, level + 1)
21: else
22:     return nodes[0]
23: end if
```

trees are initially *numCandidateTrees* primitive trees which have the highest similarities (see Algorithm4). They are used to aggregate with the low level trees in a parallel manner. The vector of primitive trees remains the same in building the optimal pattern tree. The number of low level trees specifies how many low level trees are maintained to aggregate with the candidate trees, such that their aggregation may lead to higher similarities. The level of the tree indicates how many levels of the tree have been reached so far. The higher the level, the more complex the pattern tree. *buildPatternTree*() invokes itself when building the pattern tree. Each invocation results in the number of the level being increased by 1.

As candidate trees *nodes* are sorted in a descending order with respect to similarities, *maxSim* is the highest similarity among all the candidate trees. The candidate trees *candidateTrees* for the next level is initiated to be an empty vector. *thisLowLevelTrees* is cloned from *lowLevelTrees* to perform the aggregations with candidate trees in this invocation time. Within the **for** loop at lines 4 – 11, all current candidate pattern trees are aggregated with *primitiveTrees* and *lowLevelTrees* resulting in two aggregated vectors of trees (*aggregatedPrimitives* and *aggregatedLowLevels*) respectively. All these trees are added into vectors *lowLevelTrees* and *candidateTrees*. After the loop, both the *lowLevelTrees* and *candidateTrees* are sorted and the duplicate pattern trees are removed. They are then trimmed to the size of *numLowLevelTrees* and *numCandidateTrees* respectively.

Algorithm 6 aggregate()

Input: TreeNode $candidateTree$ **and Vector** $trees$
Output Aggregated pattern trees
1: Vector $optimalTrees$ = new Vector()
2: **for** $i = 0$ **to** $trees.size()$ **do**
3: **if** (!isSubSet($trees[i]$, $candidateTree$)) **then**
4: TreeNode $optimalTree$ = aggregateOptimal($candidateTree$, $trees[i]$)
5: $optimalTrees$.add($optimalTree$)
6: **end if**
7: **end for**
8: return $optimalTrees$

Now $newMaxSim$ is the highest similarity among all the current candidate trees. If it is greater than the previous one $maxSim$, the method buildTree() is kept on invoking to next level with updated candidate trees ($candidateTrees$) and low level trees ($lowLevelTrees$). Otherwise, the tree building process stops and the root of the pattern tree which has the highest similarity is returned.

The method *aggregate()* is given in algorithm 3. It takes a pattern tree $candidate$ and a vector of pattern trees $trees$ as inputs and outputs a vector of aggregated pattern trees. In particular, for each pattern tree $trees[i], i = 0, \ldots, trees.size()$, in $trees$, if $trees[i]$ is not a subset of $candidateTree$, then it is used to aggregate with $candidate$ by invoking $aggregateOptimal()$ method. The condition of non-subset prevents repeated parts existing in the same pattern tree. Note that $trees[i]$ can be either a primitive tree or a multiple level tree. The aggregated results are added into vector $optimalTrees$ and the vector is returned. Note that the $optimalTrees$ remains the same size as $trees$.

The method *aggregateOptimal()* is given in algorithm 3. It takes two pattern trees $candidate$ and $tree$ as inputs and outputs an optimal aggregated pattern tree. In particular, among all possible aggregation operators, this method chooses the optimal one to aggregate $candidate$ with $tree$. The optimal aggregated tree is constructed and returned.

The method *applyAggregation()* is given in algorithm 3. It shows how to calculate the aggregated values when the aggregation operator and two vectors of fuzzy term values are given. This method additionally requires the use of the class fuzzy values if the aggregation operator is OWA or WA.

Method *calLambda()* calculates a singleton value λ at lines 10 and 15 so that the vector of $aggregatedVals[i]$ at line 12 and 17 has the closest distance from $classVals$ in the sense of the mean square error.

Figure 3 summarizes the process of building a pattern tree. Each node T_i, $i = 0, 1, \ldots$, represents a pattern tree, with the horizontal axis indicating the trees's level, and the vertical axis indicating its similarity to the output. The level and similarity represent the tree's complexity and performance respectively. The goal is to find a pattern tree which has a high similarity with a low level. Assume initially three primitive trees T_0, T_1 and T_2 are available and they are denoted as level 0 trees. Let $numCandidateTrees = 2$ and $numLowLevelTrees = 3$, the nodes within the ellipses and rectangles represent candidate and low level trees respectively. In particular, T_0 and T_1 are candidate trees, and T_0, T_1 and T_2 are low level trees at

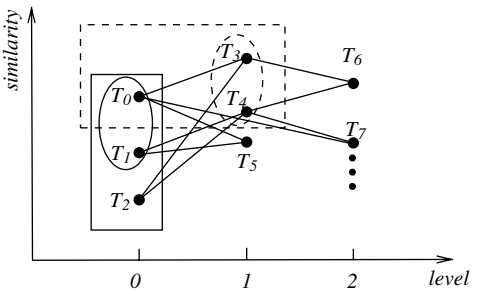

Fig. 3 Summary of building a pattern tree

level 0. Each candidate tree aggregates with each low level tree to generate a level 1 tree: T_0 aggregates with T_1 to T_5, with T_2 to T_3; T_1 aggregates with T_0 to T_5, with T_2 to T_4. As T_3 and T_4 are level 1 trees and they have the highest similarities, they are chosen as candidate trees at level 1. The low level trees are updated to T_3, T_0 and T_4 which have the highest similarities.

The process is carried out until the highest similarity at one level decreases compared to that in the previous level (e.g., T_6's similarity is less than T_3), or no candidate tree is available for further process. As the size of used fuzzy terms increases exponentially with the growth of the level, a threshold tree level can be set in practice to limit the over-growth of the pattern tree. In particular, the pattern tree induction stops when the tree reaches the threshold tree level, or the highest similarity decreases from one level to the next, whichever happens first. Note that the value of threshold tree level depends on the number of input variables. If the variables are all relevant for the classification, the more input variables are, the higher value the threshold should be.

Generally, the higher values of $numCandidateTrees$ and $numLowLevelTrees$ are, the more thoroughly the search goes and the less likely the search is to be trapped in a local optimum. High values may lead to an exhaustive search in building a globally optimal pattern tree, which is unfortunately not practical for

Algorithm 7 aggregateOptimal()

Input: TreeNode $candidate$ and TreeNode $tree$
Output: An optimal aggregated pattern tree
1: double $maxSim$ = -1
2: double[] $candidateVals$ = $candidate$.aggregatedVals
3: double[] $treeVals$ = $tree$.aggregatedVals
4: **for** $i = 1$ **to** $numAggreOpes$ **do**
5: $tempAggreVals$ = applyAggregation(i, $candidateVals$, $treeVals$, B_1.fuzzyVals)
6: $tempSim$ = similarity($tempAggreVals$, B_1.fuzzyVals)
7: **if** $maxSim < tempSim$ **then**
8: $maxSim = tempSim$
9: $aggregationOpe = i$
10: **end if**
11: **end for**
12: TreeNode $parent$ = new InternalNode($aggregationOpe$, $candidate$, $tree$)
13: return $parent$

Algorithm 8 applyAggregation()

Input: Int $aggOpe$**, double[]** $aggreVals1$**, double[]** $aggreVals2$**, and double[]** $classVals$
(required if the operator is WA or OWA)
Output: Aggregated values $aggregatedVals$

```
1: if aggOpe!= OWA or aggOpe!= OWA then
2:   for i = 1 to sizeOf(aggreVals1) do
3:     aggregatedVals[i] =
                 aggOpe(aggreVals1[i], aggreVals2[i])
4:   end for
5: else ifaggOpe == OWA then
6:   for i = 1 to sizeOf(aggreVals1) do
7:     maxVals[i] = max{aggreVals1[i], aggreVals2[i]}
8:     minVals[i] = min{aggreVals1[i], aggreVals2[i]}
9:   end for
10:  λ = calLambda(maxVals, minVals, classVals)
11:  for i = 1 to sizeOf(aggreVals1) do
12:    aggregatedVals[i] = λ * maxVals[i]+

                        (1 − λ) * minVals[i]
13:  end for
14: elss ifaggOpe == WA then
15:  λ = calLambda(aggreVals1, aggreVals2, classVals)
16:  fori = 1 to sizeOf(aggreVals1) do
17:    aggregatedVals[i] = λ * aggreVals1[i]+
                        (1 − λ) * aggreVals2[i]
18:  end for
19: end if
```

NP hard problems. In this paper, the setting of $numCandidateTrees = 2$ and $numLowLevelTrees = 3$ is used through out all examples and experiments to trade off the search effort and the capacity of escaping from local optima.

The configuration of $numCandidateTrees = 1$ and $numLowlevelTrees = 0$ forms specific pattern trees. In this configuration, only primitive trees (no low level trees) are considered to aggregate the candidate tree. The generated pattern trees have one and only one fuzzy set at each level except for level 0 (bottom level), forming snake-like trees such as in Fig. 4. They are denoted as *simple pattern trees* for later reference, in distinction with normal pattern trees. In this paper, two small examples are given to illustrate the pattern tree induction. The first one uses the simple pattern tree configuration with $numCandidateTrees = 1$ and $numLowlevelTrees = 0$, while the second uses the configuration of $numCandidateTrees = 2$ and $numLowlevelTrees = 3$.

Example 2. Assume an artificial dataset has two input variables A and B, with each having two fuzzy linguistic terms A_i and B_i, $i = 1, 2$. Also assume this dataset has two output classes X and Y. All the fuzzy membership values are shown in Table 3. To simplify the representation, only the construction of pattern tree for class X is presented and the MIN/MAX, OWA, and WA aggregations are considered here. The process of building the pattern tree is shown in Fig. 4. The pattern trees constructed in the process are denoted as T_i, $i = 0, \ldots, 9$. The are marked in the order of the time that the trees are constructed. That is, the trees with lower indices

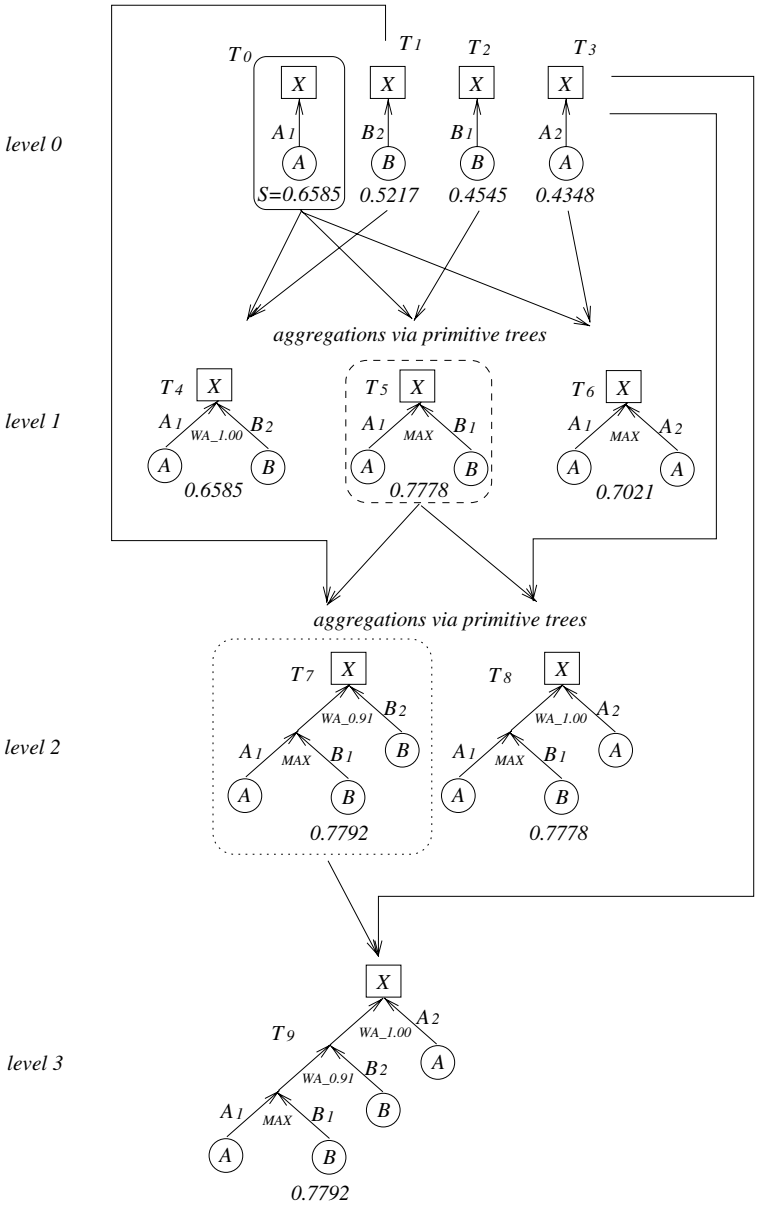

Fig. 4 Pattern tree construction for example 2

are constructed earlier than the ones with higher indices. The candidate trees at each level are surrounded by rectangular boxes with rounded corners. To distinguish them at different levels, solid, dashed, and dotted lines are used for level 0, 1, and 2 respectively. The number shown below each tree node is the similarity measure to

Table 3 An artificial dataset

A		B		Class	
A_1	A_2	B_1	B_2	X	Y
0.8	0.2	0.0	1.0	0.9	0.1
0.9	0.1	0.1	0.9	0.8	0.2
0.5	0.5	0.9	0.1	0.7	0.3
0.2	0.8	0.8	0.2	0.9	0.1
0.4	0.6	0.4	0.6	0.3	0.7
0.3	0.7	0.5	0.5	0.1	0.9

the fuzzy values of output class X. In particular, the process of building a pattern tree includes the following steps:

1. Build four primitive pattern trees T_0, T_1, T_2 and T_3, with the similarities calculated from (1) being 0.6585, 0.5217, 0.4545, and 0.4348 respectively.
2. Assign T_0 which has the highest similarity measure as the candidate tree at level 0.
3. Now consider the tree aggregation from level 0 to level 1. Try the rest of the primitive pattern trees T_1, T_2 and T_3 in turn to aggregate with T_0 using different aggregation operators MIN, MAX, OWA, and WA. The resulted similarity measures are shown in Table 4
4. Choose the aggregated tree T_5 as the new candidate tree at level 1. This is because the aggregation of T_0 and T_2 with MAX operator results in the highest similarity of 0.7778 which is greater than the similarity produced by T_0 alone. Also, the aggregated values of T_5, $(A_1\ MAX\ B_1) = \{0.8, 0.9, 0.9, 0.8, 0.4, 0.5\}$, are stored in the leaf node for further aggregation.
5. Now consider the tree aggregation from level 1 to level 2. Try the rest of the primitive trees T_1 and T_3 in turn to aggregate with T_5 using different aggregation operators. The resulting similarity measures are shown in Table 5.
6. Choose the aggregated tree T_7 as the new candidate tree at level 2, as this aggregation results in the highest similarity 0.7792 which is greater than the similarity produced by T_5. Note that a $\lambda = 0.91$ is stored in the node along with the WA operator. Also, the aggregated values of $T_7 = \{0.8185, 0.9000, 0.8259, 0.7444, 0.41856, 0.5\}$ are stored in this node for further aggregation.
7. Now consider the tree aggregation from level 2 to level 3. Try the rest of the primitive tree T_3 to aggregate with T_7 using different aggregation operators. None of the resulting similarity measures are greater than the current 0.7792. The pattern tree building process stops and the pattern tree T_7 is returned.

Table 4 Similarity measures of aggregations in step 3

	T_1	T_2	T_3
MIN	0.5610	0.3000	0.3500
MAX	0.6087	0.7778	0.7021
OWA	0.6065	0.7740	0.6815
WA	0.6585	0.6543	0.6356

Table 5 Similarity measures of aggregations in step 5

	T_1	T_3
MIN	0.5349	0.4762
MAX	0.7500	0.7143
OWA	0.7609	0.7109
WA	0.7792	0.7778

It is worth noting that each primitive tree is being checked to see if it is a subset tree of the candidate tree before the aggregation happens. For example, at level 1 for candidate tree T_5, only T_1 and T_3 among all primitive trees are allowed to aggregate. This is because other primitive trees have already appeared in T_5.

Example 3. Following the conditions given in the previous example, the construction of pattern tree using the configuration of $numCandidateTrees = 2$ and $numLowlevelTrees = 3$ is shown in Fig. 5. The candidate trees at each level are surrounded by rectangular boxes with rounded corners, while the low level trees are surrounded by rectangular boxes. Again, solid, dashed, and dotted lines are used for level 0, 1, and 2 respectively. Note that no low level trees are available at level 0. The building process is described by the following steps:

1. Build four primitive pattern trees T_0, T_1, T_2, and T_3, with the similarities calculated from (1) being 0.6585, 0.5217, 0.4545, and 0.4348 respectively.
2. At level 0, assign T_0 and T_1 which have the highest similarities as the candidate trees. There is no low level tree at this level.
3. Now consider the tree aggregation from level 0 to level 1. For the first candidate tree T_0, try the rest of the primitive pattern trees T_1, T_2, and T_3 in turn to aggre-

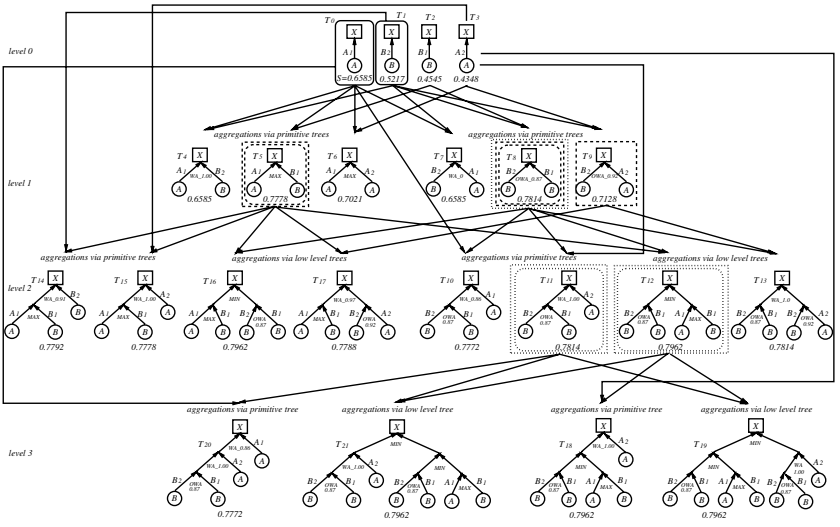

Fig. 5 Pattern tree construction for example 3

gate with, the best aggregated pattern trees are T_4, T_5 and T_6, with similarities being 0.6585, 0.7778, and 0.7021. As there is no low level trees available, no aggregations between the candidate tree and low level trees happen.

4. For the second candidate tree T_1, try the rest of the primitive pattern trees T_0, T_2 and T_3 in turn to aggregate with, the best aggregated pattern trees are T_7, T_8 and T_9, with similarities being 0.6585, 0.7814, and 0.7128. again, no aggregations between the candidate tree and low level trees happen.
5. At level 1, assign T_8 and T_5 which are level 1 trees and have the highest similarities as the candidate trees. Also, assign T_8, T_5 and T_9 which have the highest similarities as the low level trees.
6. Now consider the tree aggregation from level 1 to level 2. For the first candidate tree T_8, try the primitive pattern trees T_0 and T_3 which do not appear in the candidate tree. The resulted aggregated trees are T_{10} and T_{11}, with similarities being 0.7772 and 0.7814 respectively. Then the low level trees T_5 and T_9 are used to aggregate with T_8, resulting in T_{12} and T_{13} with similarities being 0.7962 and 0.7814.
7. Similarly for the second candidate tree T_5 at level 1, four aggregated trees T_{14}, T_{15}, T_{16}, and T_{17} are generated, with similarities being 0.7792, 0.7778, 0.7962, and 0.7788 respectively.
8. This process is carried out in the same manner from level 2 to level 3. T_{18}, T_{19}, T_{20}, and T_{21} are constructed. As none of them has a higher similarity than T_{12}, the building process stops and T_{12} is returned.

It is worth noting that the duplication removal is performed in selecting the candidate trees and low level trees. For example, at level 2, although T_{16} has a high similarity measure, it is not selected as a candidate tree or low level tree as tree T_{12} is identical to it, and also T_{12} is constructed earlier. It can be shown from Fig. 4 and Fig. 5 that, the search space for example 2 is a sub-space of that for example 3. Generally, the higher values of $numCandidateTrees$ and $numLowlevelTrees$, the more of the search space the building process reaches. On the other hand, the configuration of $numCandidateTrees = 1$ and $numLowlevelTrees = 0$ attempts to find an optimal simple tree with little computation effort.

4 Comparison to SBM, WSBM and Decision Trees

As the fuzzy subsethood based method (SBM) [1] and its extension, the weighted subsethood based method (WSBM) [12], are two specific cases of patterns trees (see below), they are chosen along with the well-known fuzzy decision tree induction [18] to compare with the proposed pattern tree induction method. For a clear representation of the comparison, the small artificial dataset as shown in Table 3 is used to generate fuzzy rules for SBM, WSBM, decision trees induction, and pattern trees induction.

4.1 SBM

The subsethood based method consists of three main steps:

1. Classifying training data into subgroups according to class values, with each group having the data which prefer voting for one class.
2. Calculating *fuzzy subsethood* values of a certain output class to each input fuzzy term. A fuzzy subsethood value of X with regard to A_1, $Sub(X, A_1) = \frac{X \cap A_1}{X}$, represents the degree to which X is a subset of A_1 (see [1, 12]), where $\cap$ is a t-norm operator (MIN is used).
3. Creating rules based on fuzzy subsethood values. In particular, for each input variable, the fuzzy term that has the highest subsethood value (must be greater than or equal to a pre-specified threshold value $\alpha \in [0, 1]$, 0.9 used in [1]) will be chosen as an antecedent for the resulting fuzzy rules.

Regardless of the subgrouping process and the use of *subsethood* rather than *similarity* measure, the SBM always generates a pattern tree which has only one level. In this tree, the fuzzy terms which have the greatest subsethood values (must be greater than or equal to α) per input variable are aggregated via a t-norm operator (MIN) to predict a class concerned.

For the artificial dataset, two groups are created. The first has the first four data points, which prefer voting for class X rather than Y, and the second has the remaining two data points. Assume class X is considered, the subsethood values of X to A_i, B_i, $i = 1, 2$, are calculated as 0.6970, 0.4848, 0.4848, and 0.6061 respectively. If α is set to 0.9 as in [1], no fuzzy rules (pattern trees) can be generated. However, for comparison purposes, α is set to 0.6 to generate the pattern tree for class X as shown in Fig. 6. This figure also shows the pattern tree for Y which is constructed in the same manner.

4.2 WSBM

WSBM is a weighted version of SBM. It uses a certain weighting strategy to represent fuzzy terms rather than choosing the one which has the greatest subsethood value per variable. In particular, the weight for fuzzy term A_i is defined as

$$w(X, A_i) = \frac{Sub(X, A_i)}{max_{j=1,2} Sub(X, A_j)}. \tag{11}$$

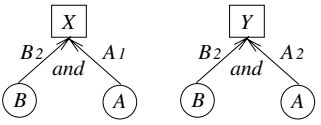

Fig. 6 Pattern trees generated by SBM using the artificial dataset

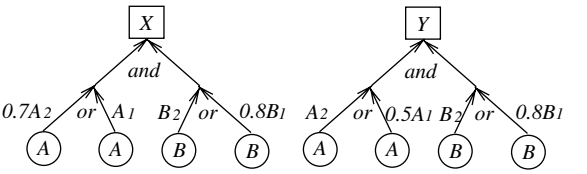

Fig. 7 Pattern trees generated by WSBM using the artificial dataset

Using this equation, the weights for A_1, A_2, B_1 and B_2 in the classification of X are calculated as 1, 0.7, 0.8, and 1 respectively. The fuzzy rules generated by WSBM form fixed structured pattern trees as shown in Fig. 7.

Unlike the proposed pattern tree induction method which can generate different structured patter trees, SBM and WSBM generate trees with fixed structures. In particular, SBM trees have only one level, on which the fuzzy terms which have the highest subsethood values per input variable aggregate via a t-norm operator. WSBM trees have two levels. On the bottom level, all fuzzy terms (with different weights) for each variable aggregate via a t-conorm operator, on the top level the aggregated values further aggregate via a t-norm operator. The fixed structures of SBM and WSBM may prevent them from obtaining high accuracy in classification applications (see Sect. 4.6).

4.3 Fuzzy Decision Tree Induction

The decision tree induction [10] is a classic machine learning method. It has been extended to induce fuzzy decision tree by Yuan and Shaw [18], where the fuzzy entropy is used to guide the search of most effective branches. A fuzzy decision tree as shown in Fig. 8 can be built using [18] over the artificial dataset. It has a root on the top and several leaf nodes on the bottom. When a new data sample needs to be classified by the fuzzy decision tree, it traverses from the root to the bottom. The output class of a leaf node in the bottom, which the data sample reaches with the highest truth value, is chosen as the prediction class.

For the same training dataset, fuzzy decision tree induction may generate different results with different numbers of minimal data points per leaf node, which are used as criteria to terminate the tree building process. Considering only six data samples are available, this can be set to 1 to 6. Among all these, the best result (as shown in Fig. 8, when the number of leaf nodes is set to 1 or 2) has been chosen for comparison in Sect. 4.6.

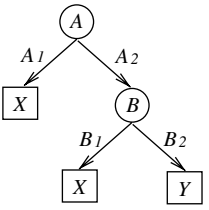

Fig. 8 A decision tree generated by fuzzy decision tree induction using the artificial dataset

4.4 Relation between Fuzzy Decision Trees and Pattern Trees

A fuzzy decision tree can be converted to a set of pattern trees whose size is equal to the number of output classes. In particular, the easiest way is to convert a fuzzy decision tree to a set of semi-binary pattern trees. That is, there are maximally two branches allowed for each node (except for the root) in the pattern tree. The conversion process is outlined as follows. For each fuzzy rule (a branch from the root to a leaf node) in a decision tree, the input fuzzy terms are connected by a t-norm operator (usually MIN) in different levels of the pattern tree. A fuzzy rule consisting of n fuzzy terms results in a $(n-1)$-level binary pattern tree as the bottom level contains two fuzzy terms. The fuzzy rules which have the same classification class are connected by a t-conorm (usually MAX) at the top level (level n) to construct a pattern tree for this output class. The number of levels of the generated semi-binary pattern trees remains the same as the fuzzy decision tree, regardless whether the decision tree is binary or not. Fig. 2 shows the pattern trees which are equivalent to the decision tree as shown in Fig. 8. They both have the same rule base as listed in (8) – (10).

The conversion from a decision tree to pattern trees is not unique. For example, an alternative conversion from Fig. 8 is shown in Fig. 9, which can be represented by two rules as following.

$$Rule1 : IF A = A_1 \; OR \; B = B_1 \; THEN \; class = X, \tag{12}$$

$$Rule2 : IF A = A_2 \; AND \; B = B_2 \; THEN \; class = Y. \tag{13}$$

Note that these two fuzzy rules are functionally equal to rules (8), (9) and (10). Such a conversion is closely related to Zadeh's work on compactification [20]. When the size of fuzzy terms for each variable increases, the conversion of a fuzzy decision tree to multi-branch pattern trees are desirable and the work on that is on-going.

On the other hand, pattern trees can be converted to a decision tree. The fuzzy decision tree shown in Fig. 8 can be converted from either Fig. 2 or Fig. 9.

It is worth noting that decision trees and pattern trees are different in terms of four aspects: 1) the former focus on separating data samples which have *different* output classes, while the latter focus on representing the structures of data samples which have the *same* output classes; 2) for each internal node, the former consider *all* fuzzy terms of the chosen input variable, whist the latter only consider *one*; 3) the former normally make use of MIN and MAX aggregations *only*, while the latter can use *any* aggregations as described in Sect. 2.2; and 4) the tree induction methods are completely different, with the former based on the heuristics of *entropy measure* [18] while the latter on the heuristics of *similarity measure*.

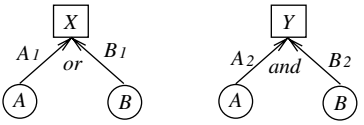

Fig. 9 Alternative pattern trees converted from the decision tree in Fig. 8

4.5 Advantages of Trees

Both decision trees and pattern trees can be converted to conventional fuzzy rules. In terms of the computational complexity, both decision trees and patterns trees have the advantage over conventional rule bases. An example is given below to show how pattern trees can simplify the computation.

Definition 2. Let assume fuzzy variable A, B, C, D, E, and F with each having two fuzzy linguistic terms A_i, B_i, C_i, D_i, E_i and F_i, $i = \{1, 2\}$. One pattern tree associated with the output class X is shown in Fig. 10. The directly extracted fuzzy rule from the pattern tree can be written as

$$\begin{aligned} IF\ (((A = A_1)\ OR\ (B = B_2))\ AND\ (C = C_1))\ AND \\ (((D = D_1)\ AND\ (E = E_2))\ OR\ (F = F_2)) \\ THEN\ class = X, \end{aligned} \tag{14}$$

which is equivalent to the conventional fuzzy rules (without using the OR operator explicitly within a rule):

$$IFA = A_1\ AND\ C = C_1\ AND\ D = D_1\ AND\ E = E_2 \\ THEN\ class = X \tag{15}$$

$$IFA = A_1\ AND\ C = C_1\ AND\ F = F_2 \\ THEN\ class = X \tag{16}$$

$$IFB = B_2\ AND\ C = C_1\ AND\ D = D_1\ AND\ E = E_2 \\ THEN\ class = X \tag{17}$$

$$IFB = B_2\ AND\ C = C_1\ AND\ F = F_2 \\ THEN\ class = X \tag{18}$$

If a data sample needs to be classified with the conventional fuzzy rules, the four rules should fire in turn, leading to the evaluation of $A = A_1$, $B = B_2$, $C = C_1$, $D = D_1$, $E = E_2$, $F = F_2$, the *and* operator, and the *or* operator to be 2, 2, 4, 2, 2, 2, 10 and 3 times respectively. However, for the computation upon the pattern tree directly, they are only computed 1, 1, 1, 1, 1, 1, 3 and 2 times respectively.

From this example, it can be concluded that although the pattern trees (or decision trees) may share the same fuzzy rule base with a conventional rule model, it

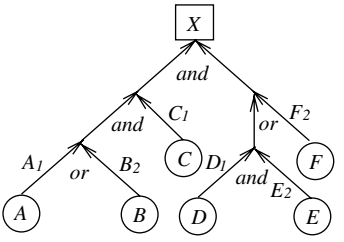

Fig. 10 A example pattern tree

can significantly reduce the computation time in the test stage. This advantage is especially important when quick response is required for fuzzy applications.

4.6 Pattern Trees and the Comparison Results

The aggregations of MIN/MAX, algebraic AND/OR, Šukasiewicz, EINSTEIN, OWA, and WA are considered in building a pattern tree using the proposed method. Two similarity measures as shown in (1) and (3) are used and the best result (using 3) is reported. The threshold tree level for pattern tree induction is set to 3. Both simple pattern tree (with $numCandidateTrees = 1$ and $numLowlevelTrees = 0$) and pattern tree (with $numCandidateTrees = 2$ and $numLowlevelTrees = 3$) configurations are used, resulting in simple pattern trees and pattern trees as shown in Fig. 11 and Fig. 12 respectively. It is worth noting that, if only MIN/MAX are allowed in the simple pattern tree induction, the generated trees using similarity defined in either (1) or (3) are exactly the same as the trees shown in Fig. 9, with *and* being MIN and *or* being MAX.

Now apply this artificial dataset to the rules (or trees) generated by SBM, WSBM, fuzzy decision trees, simple pattern trees, and pattern trees, the results including the number of correctly predicted data samples (No.), the root mean square error (RMSE) of the predictions for class X, Y, and their average (ARMSE) are shown in Table 6.

It is clear that the pattern trees perform the best, in terms of both the correctly predicted number and the mean square error, among all the methods. Simple pattern trees perform slightly worse, but still better than the fuzzy decision tree. This example shows that SBM and WSBM cannot get good results as their fixed pattern tree structures prevent them appropriately representing the structure of the dataset. Fuzzy decision trees can generate different tree structures. However, they lack some candidate search spaces represented by the t-conorm aggregations of fuzzy terms of different variables. For instance, in this example, Fig. 8 considers whether A_1, A_2, $A_1 \vee A_2$, and $A_2 \wedge B_1$ etc. are important or not for the classification, but not $A_1 \vee B_1$ explicitly, thus failing to find such an important rule. In contrast, the proposed pattern tree induction method explicitly considers both t-norm and t-conorm aggregations of fuzzy terms in building trees. Thus, it is more likely to find optimal solutions.

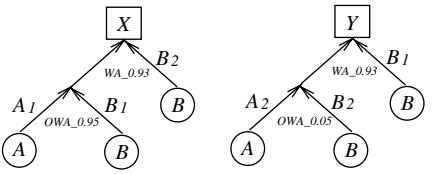

Fig. 11 Pattern trees generated using simple pattern tree configuration

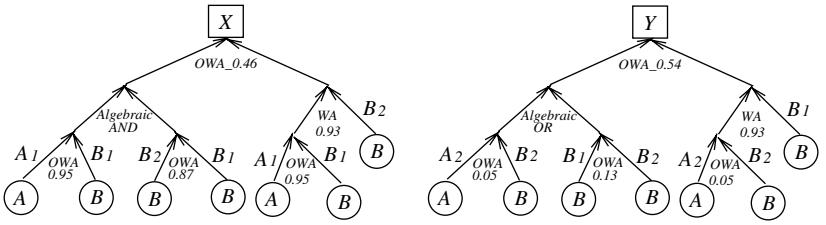

Fig. 12 Pattern trees generated using pattern tree configuration

Table 6 Results of SBM, WSBM, Fuzzy decision tree, and patter trees using the artificial dataset

	No.	RMSE(X)	RMSE(Y)	ARMSE
SBM	4	0.3916	0.2	0.2958
WSBM	4	0.2386	0.3435	0.2911
Fuzzy decision tree	5	0.2	0.2	0.2
Simple pattern trees	6	0.1956	0.1956	0.1956
Pattern trees	6	0.1715	0.1715	0.1715

5 Experimental Results using Saturday Morning Problem Dataset and UCI Datasets

To demonstrate the effectiveness of the proposed pattern tree induction method, Sect. 5.1 presents comparative studies with the fuzzy subsethood based method (SBM) [1], the weighted subsethood based method (WSBM) [12], and the fuzzy decision tree [18] using the Saturday Morning Problem dataset [18, 1, 12]. Section 5.2 presents more comparison to fuzzy decision trees over the datasets obtained from UCI machine learning repository [8], which include Iris-Plant, Wisconsin Breast Cancer, Glass identification, Diabetes, and Wine Recognition. Note that the results of SBM and WSBM over the Iris-Plant dataset, which were reported in [12], are also provided here for comparison.

5.1 Saturday Morning Problem

The Saturday Morning Problem (SMP) dataset has 16 data samples, with each having 4 input variables, *outlook, temperature, humidity* and *wind*. Each variable has fuzzy terms as follows: *outlook = {sunny, cloudy, rain}, temperature ={hot, mild, cool}, humidity ={humid, normal}* and *wind ={windy, not-windy}*. The classification result is one of the plans to be taken: *class ={volleyball, swimming, weight-lifing}*.

Table 7 shows the best performances of SBM, WSBM, fuzzy decision tree, simple pattern trees, and pattern trees respectively. Note that the threshold pattern tree level is set to be 3. No. is the number of correctly predicted data points. RMSE(1), RMSE(2) and RMSE(3) are the root mean square errors for class *volleyball, swimming* and *weight-lifting* respectively and ARMSE is the average of these three.

Table 7 Performance of SBM, WSBM, Fuzzy decision trees, and patter trees using SMP dataset

	No.	RMSE(1)	RMSE(2)	RMSE(3)	ARMSE
SBM	15	0.2046	0.1620	0.1323	0.1663
WSBM	15	0.2818	0.2480	0.4398	0.3232
Fuzzy decision tree	13	0.2681	0.1369	0.1953	0.2001
Simple pattern trees	14	0.3279	0.1750	0.2669	0.2566
Pattern trees	15	0.2078	0.0995	0.2256	0.1776

Among the comparison, SBM obtains the best result, in terms of both the number of correctly predicted data points and the root mean square errors (indicating the differences between the predicted values and the actual values). This is because a default fuzzy rule has been introduced in SBM method to predict class *weight-lifting*, as it cannot generate any "meaningful" rules for this class. This certainly helps in the example, but may not be useful in other datasets – especially when there are more than one class which SBM cannot generate any "meaningful" rules for.

WSBM seems get a good result in terms of the correctly predicted number, it however has the greatest ARMSE among all methods. This reveals that the average difference between the predicted values and the actual values is the highest.

Fuzzy decision tree has a low average RMSE, but only correctly predicts 13 out of 16 data samples. Simple pattern tree predicts 14 correctly, but with a higher average RMSE. Pattern trees perform very well in terms of both correctly predicted number and ARMSE. It is worth noting that, unlike SBM method, three pattern trees are constructed here for three classes respectively. With respect to finding a correct pattern for each class, pattern trees perform nearly the same as SBM in finding the first one, but much better in the second. However, the default rule which SBM uses outperforms pattern trees for the third class.

5.2 UCI Machine Learning Datasets

The datasets of Iris-Plant, Wisconsin Breast Cancer, Glass Identification, Diabetes, and Wine Recognition obtained from UCI machine learning repository [8], which have been widely used as benchmarks in classification applications, are summarized in Table 8.

For all datasets except Iris-Plant, a simple fuzzification method based on six evenly distributed trapezoidal membership functions for each input variable is used to transform the crisp values into fuzzy values. To be comparable to the result reported in [12], Iris-Plant dataset uses three evenly distributed trapezoidal fuzzy sets for each variable.

All datasets (*ds*) are divided into training datasets and test ones. Assume every dataset is labeled for data samples, the training sets (*ds-odd*) contain the odd numbered data samples and the test sets (*ds-even*) contain the even numbered ones. The performances of the SBM and WSBM (only for Iris-Plant dataset),

Table 8 Summary of Iris-Plant, Wisconsin Breast Cancer, Glass Identification, Diabetes, and Wine Recognition Datasets

Name	Number of data	Number of inputs	Number of classes
Iris-Plant	150	4	3
Wisconsin Breast Cancer	699	9	2
Glass identification	214	9	6
Diabetes	768	8	2
Wine recognition	178	13	3

fuzzy decision tree (FDT), simple pattern trees (SimPT), pattern trees with level no more than 5 (PTLevel5) and pattern trees with level no more than 9 (PTLevel9) over different combinations of training-test sets for different datasets are shown in Tables 9, 10, 11, 12 and 13. PTLevel5 trees maintain a good comprehensibility as they have maximal $2^5 = 32$ leaf nodes. However, PTLevel9 trees may be too complex as they have maximal $2^9 = 512$ leaf nodes, although the trees usually have much fewer leaf nodes than this maximum. As the Iris-Plant dataset has only 4 input variables, there is no need to set the threshold tree level to 5. Actually the maximal level the trees reach is no more than 3.

All the results reported for fuzzy decision trees are the best results among the use of different numbers of leaf nodes. The MIN/MAX, algebraic AND/OR, Šukasiewicz, EINSTEIN, OWA, and WA are considered in building pattern trees. Note that $*$ in Table 9 indicates the results are not available in [12].

PTLevel9 trees outperform the other trees over all five datasets. They obtain the highest prediction accuracies in all experiments, except for the training-test sets being ds and ds in Glass Identification and Diabetes datasets. On the other hand, FDT nearly performs the worst except in Diabetes dataset, in which it outperforms SimPT. SimPT performs roughly the same to PTLevel5 – both of them perform better than FDT, but worse than PTLevel9 trees.

It is worth noting that pattern trees perform in a consistent way for different combinations of training and test datasets, while fuzzy decision trees do not. This can be seen from the Glass Identification and Diabetes datasets. Fuzzy decision trees generate large differences in classification accuracy between the first (or the second) combination of the training-test datasets and the third one, due to the over-fitting problem. The reason is that decision tree induction considers only a portion of the whole training dataset in choosing the branches at low levels of trees. The lack of using the whole training dataset inevitably prevents the method finding better

Table 9 Prediction accuracy of SBM, WSBM, fuzzy decision trees, simple pattern trees, and pattern trees using Iris-Plant dataset

Training	Testing	SBM	WSBM	FDT	SimPT	PTLevel3
ds-odd	ds-even	80%	93.33%	97.33%	97.33%	97.33%
ds-even	ds-odd	78.67%	93.33%	97.33%	98.67%	98.67%
ds	ds	*	*	97.33%	97.33%	97.33%

Table 10 Prediction accuracy of fuzzy decision trees, simple pattern trees, and pattern trees using Wisconsin Breast Cancer dataset

Training	Testing	FDT	SimPT	PTLevel5	PTLevel9
ds-odd	ds-even	93.70%	94.84%	94.84%	95.41%
ds-even	ds-odd	95.71%	96.57%	95.42%	96.57%
ds	ds	96.85%	97.42%	97.13%	98.14%

tree structures for all the dataset. In contrast, pattern trees make use of the whole data in building each level of the tree, which ensures the tree to keep good generality for classifications. Therefore, even complex pattern trees do not suffer from over-fitting.

Simple pattern trees usually have compact structures, and they can be simpler than fuzzy decision trees. For example, Fig. 13 shows the decision tree (with prediction accuracy of 92.13%) generated using Wine Recognition dataset with training set being *ds-odd* and test set being *ds-even*. The ellipses are the input variables and the rectangles are the output classes (0, 1, or 2). Note that the empty rectangles mean no decision class is available. $Fi, i = 0, \ldots, 5$, are the fuzzy terms associated with each input variable.

Figure 14 shows the three simple pattern trees (with prediction accuracy of 93.25%) generated using the same training dataset. In terms of the size of leaf nodes, the three simple pattern trees have $6 \times 3 = 18$ leaf nodes in total as each pattern tree per class has 6 fuzzy terms, while the decision tree has 26. Figure 15 shows the constructed level5PT tree (with prediction accuracy of 94.38%) for class 0, which

Table 11 Prediction accuracy of fuzzy decision trees, simple pattern trees, and pattern trees using Glass Identification dataset

Training	Testing	FDT	SimPT	PTLevel5	PTLevel9
ds-odd	ds-even	55.14%	61.68%	62.61%	62.61%
ds-even	ds-odd	57.94%	55.14%	58.87%	60.74%
ds	ds	87.75%	71.02%	70.09%	72.89%

Table 12 Prediction accuracy of fuzzy decision trees, simple pattern trees, and pattern trees using Diabetes dataset

Training	Testing	FDT	SimPT	PTLevel5	PTLevel9
ds-odd	ds-even	75.26%	72.65%	76.82%	77.60%
ds-even	ds-odd	74.48%	72.13%	74.21%	75.26%
ds	ds	91.15%	75.52%	75.39%	76.30%

Table 13 Prediction accuracy of fuzzy decision trees, simple pattern trees, and pattern trees using Wine Recognition dataset

Training	Testing	FDT	SimPT	PTLevel5	PTLevel9
ds-odd	ds-even	92.13%	93.25%	94.38%	96.62%
ds-even	ds-odd	91.01%	97.75%	97.75%	97.75%
ds	ds	97.75%	98.31%	97.75%	98.31%

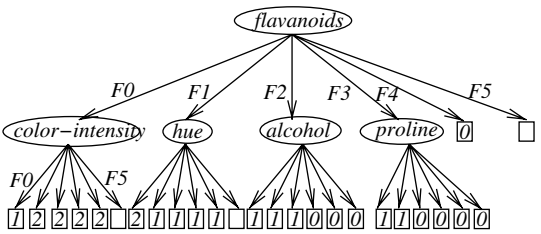

Fig. 13 Decision tree generated using Wine Recognition dataset

has 9 leaf nodes. Its complexity is still acceptable although it is more complex than fuzzy decision trees or simple pattern trees.

The complexity can also be compared in the form of rule representation. For example, if only class 0 is considered, the rules extracted from the decision tree, simple pattern trees, and pattern trees are listed in (19), (20) and (21) respectively. It can be seen that the simple pattern tree has the simplest form.

$$\begin{aligned}
&(IF\ flavanoids = F_2\ AND\ alcohol = F_3)\ OR\\
&(IF\ flavanoids = F_2\ AND\ alcohol = F_4)\ OR\\
&(IF\ flavanoids = F_2\ AND\ alcohol = F_5)\ OR\\
&(IF\ flavanoids = F_2\ AND\ proline = F_2)\ OR\\
&(IF\ flavanoids = F_2\ AND\ proline = F_3)\ OR\\
&(IF\ flavanoids = F_2\ AND\ proline = F_4)\ OR\\
&(IF\ flavanoids = F_2\ AND\ proline = F_5)\ OR\\
&(IF\ flavanoids = F_5)\\
&THEN\ class = 0
\end{aligned} \tag{19}$$

$$\begin{aligned}
&(((((IF\ flavanoids = F_3\ OWA\ malic\text{-}acid = F_1)\ OWA\\
&proline = F_3)\ OWA\ proline = F_4)\ OWA\\
&proline = F_5)\ WA\ alcalinity = F_1)\\
&THEN\ class = 0
\end{aligned} \tag{20}$$

$$\begin{aligned}
&(((((IF\ flavanoids = F_3\ OWA\ malic\text{-}acid = F_1)\ Luc_OR\\
&proline = F_3)\ OWA
\end{aligned}$$

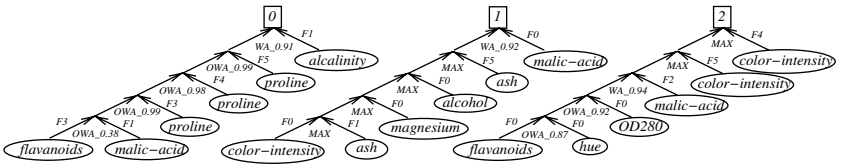

Fig. 14 Simple pattern trees generated using Wine Recognition dataset

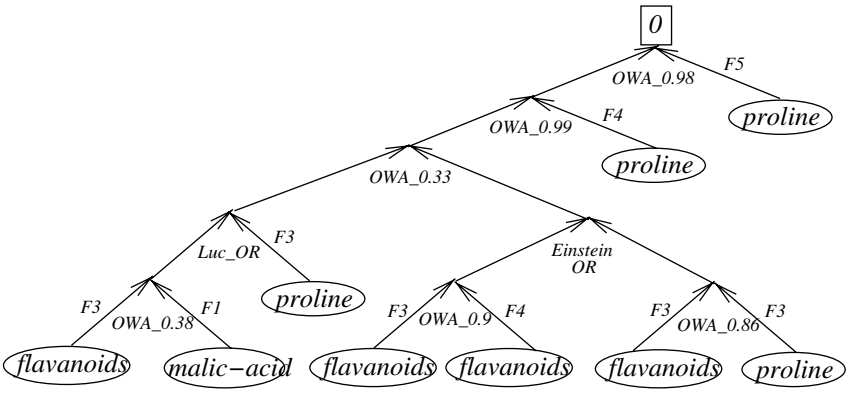

Fig. 15 A pattern tree generated using Wine Recognition dataset for class 0

$$\begin{aligned}((flavanoids = F_3\ OWA flavanoids = F_4)\ Einstein\_OR \\ (flavanoids = F_3\ OWA\ proline = F_3))) \\ OWA\ proline = F_4)\ OWA\ proline = F_5) \\ THEN\ class = 0 \end{aligned} \tag{21}$$

With respect to choosing appropriate methods for real world applications, PTLevel9 trees are favored if performance is a critical factor. The drawback is that these trees may compromise the comprehensibility, as the number of leaf nodes tends to be large. If, however, the comprehensibility is critical for the solution to be considered, SimPT trees are a good choice.

6 Weighted Pattern Trees

The classification using pattern trees discussed in Sect. 2.3 is based on the assumption that all pattern trees each have the same confidence on predicting a particular class, though it is not always the case in real world applications. *Weighted trees* are introduced to resolve this problem. For each tree, the similarity of such tree to the output class is served as a degree of confidence, to reflect how confident to use this tree to predict such a class. For example, if the two trees in Fig. 2 have similarities of 0.1 and 0.8 respectively, they can be called weighted pattern trees with weights of 0.1 and 0.8. The prediction using weighted pattern trees is the same as pattern trees, except that the final truth values are multiplied by the weights of trees. As an example, let's revise the classification problem in Sect. 2.3; consider classifying the fuzzy data $A_1 = 0.8$, $A_2 = 0.2$, $B_1 = 0$, and $B_2 = 1$ over pattern trees (with weights of 0.1 and 0.8) in Fig. 2, its truth values over pattern trees for class X and Y change to 0.08 and 0.16 respectively, and Y (rather than X) is therefore chosen as the output class. This reflects the fact that, if a tree has a low weight, even an input data has a high firing strength over such pattern tree, the prediction is not confident.

Note that this example is merely used to show how weighted pattern trees work. In practice, a pattern tree with weight 0.1 may not be trusted to predict a class.

The concept of weighted pattern trees is important. It offers an option to trade off the complexity and performance of pattern trees. The pattern tree building process can stop at very compact trees, if it detects that the similarities (weights) of such trees are already high enough. In addition, it enhances the comprehensibility of pattern trees. For example consider the construction of the pattern tree for class Y in Fig. 2, assume that the tree growing from the primitive tree $B_2 \Rightarrow Y$ to $B_2 \wedge A_2 \Rightarrow Y$ leads to the weight increase from 0.6 to 0.8, this gradual change can be interpreted in a comprehensible way:

$$IF\ B = B_2\ THEN\ it\ is\ possible\ that\ class = Y, \quad (22)$$

$$IF\ B = B_2\ AND\ A = A_2\ THEN\ it\ is\ very\ possible\ that\ class = Y, \quad (23)$$

if users pre-define semantic ranges of weights, say *less possible*: [0, 0.3), *possible*: [0.3, 0.7), and *very possible*: [0.7, 1]. Thus, the graduate change of confidence of pattern trees can be monitored from the pattern tree induction process. This provides a very transparent way for fuzzy modeling.

7 Experimental Results using BT Dataset

In this section, different variants of pattern trees, namely simple pattern trees, weighted simple pattern trees, pattern trees, and weighted pattern trees, are applied to a sample customer satisfaction dataset from BT. This dataset has a total of 26 input parameters representing ease of contact, problem understanding, service quality, repair time, and overall event handling. Among the input parameters, 6 are numerical parameters and the rest 20 are category ones, with the number of possible values being from 2 up to 17. The output parameter consists of 7 classes reflecting varying degrees of customer satisfaction.

The BT customer satisfaction dataset has 16698 data points in total. Let *ds*, *ds-odd* and *ds-even* be the datasets which contains the whole, the odd numbered, and the even numbered data points respectively. The number of data per class for these three datasets are shown in Table 14, with ci, $i = 0, \ldots, 6$ standing for class i. As can be seen, this dataset is not well balanced as the number of data per class varies significantly. The experiments of (weighted) pattern trees are carried out in three combinations of training-test datasets, namely, *odd-even, even-odd*, and *ds-ds*. In all experiments, a simple fuzzification method based on three evenly distributed trapezoidal membership functions for each numerical input parameter is used to transform the crisp values into fuzzy values. All aggregations as listed in Table 2 are allowed in pattern trees. The similarity measure as shown in (3) is used.

Table 14 Number of data per class for ds, ds-odd and ds-even datasets

	c0	c1	c2	c3	c4	c5	c6
ds	1895	7289	4027	382	1361	853	891
ds-odd	949	3659	1990	197	660	448	446
ds-even	946	3630	2037	185	701	405	445

7.1 Prediction accuracy and overfitting

The prediction accuracy and rule number of the fuzzy decision trees (FDT) with respect to the minimal number of data per leaf node (used as criteria to terminate the training), over different combinations of training-test sets are shown in Fig. 16. The prediction accuracy of pattern trees (PT) and weighted pattern trees (WPT) with respect to different tree levels, over different combinations of training-test sets is shown in Fig. 17.

The experiments show that weighted pattern trees and pattern trees perform roughly the same. In fact, the former slightly outperform the latter. Table 15 shows the highest prediction accuracy of fuzzy decision trees, (weighted) simple pattern trees and (weighted) pattern trees over different combinations of training-test sets. Both weighted and unweighted pattern trees can obtain higher prediction accuracy than fuzzy decision trees in *odd-even* and *even-odd* combinations. However, if considering *ds-ds* combination, fuzzy decision trees perform much better. This just reflects the overfitting of fuzzy decision trees, since fuzzy decision trees generate large differences in classification accuracy between the *odd-even*, *even-odd* combinations and *ds-ds* one. The reason is that decision tree induction considers only a portion of the whole training dataset in choosing the branches at low levels of trees. The lack of using the whole training dataset inevitably prevents the method finding generalized tree structures for all the dataset. In contrast, pattern trees make use of the whole data in building each level of the tree, which ensures the tree to keep good generality for classifications. Therefore, even complex pattern trees do not suffer from over-fitting.

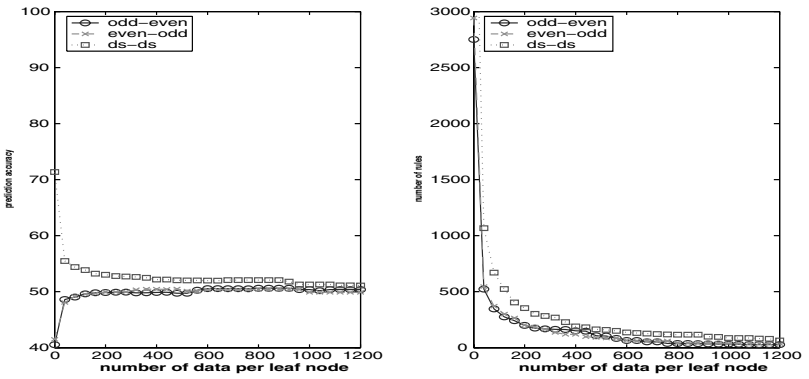

Fig. 16 Prediction accuracy and rule number of fuzzy decision trees with different number of data per leaf node

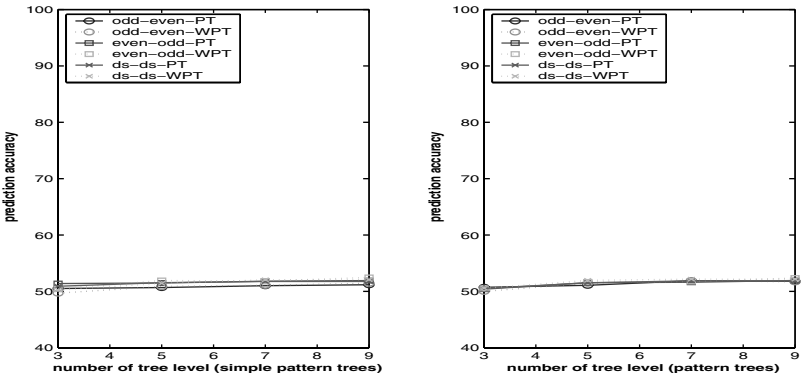

Fig. 17 Prediction accuracy of pattern trees and weighted pattern trees with different tree levels

In addition, the experiments show that (weighted) pattern trees tend to converge to a accuracy rate when the number of tree level becomes large. It has no trend of overfitting. This property is essential to ensure a stable, compact and effective fuzzy model for the problem at hand. In fact, (weighted) pattern trees with two or three level perform very well for all conducted experiments. That means, pattern trees which consist of maximal $2^3 = 8$ leaf nodes can perform well, in contrast to tens, or even hundred rules used in fuzzy decision trees. This provides a superb solution to achieve a highly effective as well as compact fuzzy model.

7.2 Approximate accuracy

Section 7.1 presents the prediction accuracy of trees in a very strict way. That is, if and only if a data is predicted exactly as its class, this prediction is counted as a correct one. In other words, there is no distinction between "close" errors and "gross" errors. In BT customer dataset, this distinction is necessary as it reflects how far the prediction is away from the actual class. It is much worse if a data of class 0 is mis-predicted to class 5 rather than to class 1. To resolve this problem, three accuracy estimations, namely *accuracy 1*, *accuracy 2*, and *accuracy 3* are employed to estimate prediction accuracy which has no tolerance (the same as the one used in Sect. 7.1), tolerance of adjacent mis-prediction, and tolerance of mis-prediction

Table 15 Highest prediction accuracy of fuzzy decision tree, pattern trees, and weighted pattern trees

	FDT	SimPT		PT	
		no weight	weight	no weight	weight
odd-even	50.62%	51.19%	51.45%	51.89%	51.92%
even-odd	50.47%	51.92%	52.47%	51.93%	52.37%
ds-ds	71.36%	51.82%	52.09%	51.88%	52.18%

within two closest neighbor classes in either direction, respectively. For example in the BT dataset, the mis-prediction of a class 0 data to class 2 is still counted as a correct prediction in the estimation of *accuracy 3*, although it is not counted in *accuracy 1* and *accuracy 2*.

Table 16 shows the highest prediction accuracy of fuzzy decision trees, (weighted) simple pattern trees and (weighted) pattern trees over odd-even combination of training-test sets. Both weighted and unweighted pattern trees can obtain higher prediction accuracy than fuzzy decision trees in estimation of accuracy 1 and 2. In estimation of accuracy 3, weighted pattern trees perform the best, and fuzzy decision trees outperform unweighted pattern trees. Generally, accuracy 2 and 3 are very consistent with accuracy 1. Pattern trees with a high value of accuracy 1 usually have high values of accuracy 2 and 3. This table also shows that both fuzzy decision trees and pattern trees can obtain over 80% prediction accuracy if the closest error can be tolerated.

7.3 Interpretation of pattern trees

Each pattern tree can be interpreted as a general rule. Considering building level 5 simple pattern trees using odd dataset, 7 simple pattern trees can be obtained, with each representing one output class. Fig. 18 shows the tree for class 0. The ellipses are the input parameters and the rectangle is the output class 0. Over each branch, i and $Fi, i = 0, \ldots$, are category values and fuzzy terms associated with each input parameter. All aggregators as shown in Table 2 are allowed to be used in pattern trees. For example, *A_AND* is algebraic AND, and *WA_0.84* is weighted average with weight vector $w = (0.84, 0.16)$.

Fig. 18 roughly indicates that one example combination yielding highly satisfied customers are: no call re-routing, fast fault reporting time, high technician competence, being well-informed through the repair process, and high satisfaction with company/product in general. Here, we say roughly, as we use different aggregations such as weighted average (WA), ordered weighted average (OWA), algebraic and (A_AND) etc. rather than simple AND.

These 7 pattern trees obtains an accuracy of 51.46%. In particular, the confusion table is shown in Table 17, where SA and SP are number of data for actual and predicted classes respectively.

Table 16 Highest prediction accuracy of fuzzy decision trees, pattern trees, and weighted pattern trees over odd-even training-test combination

	FDT	SimPT		PT	
		no weight	weight	no weight	weight
accuracy 1	50.62%	51.19%	51.45%	51.89%	51.92%
accuracy 2	84.02%	84.08%	84.68%	84.44%	84.82%
accuracy 3	92.13%	91.74%	92.70%	91.85%	92.29%

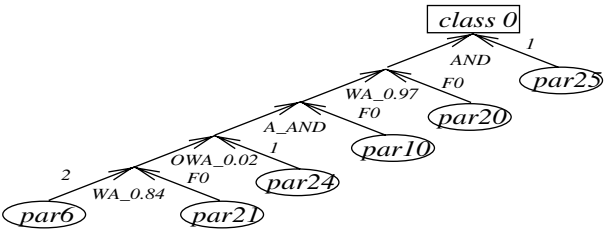

Fig. 18 Pattern tree for class 0 using *odd* dataset

7.4 Limitation

It is a little strange that no prediction is made to *c0* for all test data in Table 17. It can be seen that nearly all data (884 out of 946 in fact) with class 0 are mis-classified to class 1. A first intuition is to raise the weight of pattern tree for class 0 . However, this does not work; the raise does not only lead to the data of class 0 to be classified correctly, but also lead to the majority of data of class 1 to be classified as class 0. Considering that there are 3630 data of class 1 and only 946 data of class 0 in even dataset, the raise of weight for class 0 tree would therefore cause more mis-classifications. This can be seen in Fig. 19, where the fired values of first 50 data points per class in *even* dataset over pattern trees constructed from *odd* dataset are shown. The real class line indicates the real classes of the data; for example, data numbered from 0 to 49 have class 0, and those from 50 to 99 have class 1.

The phenomena of no prediction on particular classes also occurs in fuzzy decision trees. Considering the highest accuracy of 50.62% which fuzzy decision trees can obtain over odd-even combination, the confusion table in this case is shown in Table 18, where no data is predicted to *c3*, *c4* or *c5*.

An interesting experiment is carried out trying to improve the prediction accuracy for class 0 in Table 17. The data of classes 0 and 1 in odd dataset are selected as a new training dataset, and the data which are of classes 0 and 1 in even dataset and are classified as class 1 in Table 17 are selected as a new test dataset. Both fuzzy decision trees and pattern trees are applied to the new training data and tested over the new test data. Surprisingly, they obtain the same highest accuracy of 78.76%. Table 19 shows the confusion table, which only has one data

Table 17 Confusion table for pattern tree prediction using odd-even combination

		Prediction							
		c0	c1	c2	c3	c4	c5	c6	*SA*
Actual	c0	0	884	51	0	3	2	6	946
	c1	0	3283	309	0	10	15	13	3630
	c2	0	1122	751	1	53	72	38	2037
	c3	0	59	94	0	6	20	6	185
	c4	0	122	395	1	30	104	49	701
	c5	0	50	142	0	23	120	70	405
	c6	0	35	129	0	37	131	113	445
	SP	0	5555	1871	2	162	464	295	8349

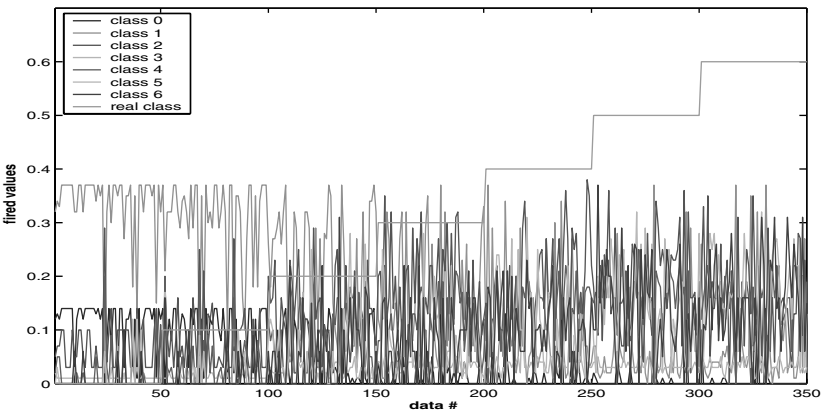

Fig. 19 Fired values of first 50 data points per class in *even* dataset over pattern trees constructed from *odd* dataset

Table 18 Confusion table for fuzzy decision tree prediction using odd-even combination

		Prediction							
		c0	c1	c2	c3	c4	c5	c6	SA
Actual	c0	17	777	151	0	0	0	1	946
	c1	33	2980	605	0	0	0	12	3630
	c2	9	942	1035	0	0	0	51	2037
	c3	0	49	121	0	0	0	15	185
	c4	1	84	509	0	0	0	107	701
	c5	1	26	226	0	0	0	152	405
	c6	0	31	220	0	0	0	194	445
	SP	61	4889	2867	0	0	0	532	8349

Table 19 Confusion table for prediction of both fuzzy decision trees and pattern trees using new training and test datasets

		Prediction		
		c0	c1	SA
Actual	c0	0	884	884
	c1	1	3282	3283
	SP	1	4166	4167

predicted as class 0 (and it is wrong actually). It can be concluded that the data of class 0 and class 1 can not be separated properly by either fuzzy decision trees or pattern trees.

8 Conclusions

This paper has proposed a type of tree termed *pattern trees* which make use of different aggregations. Like decision trees, pattern trees are an effective machine learning approach for classification applications. A novel pattern tree induction

method has been proposed. The comparison to other classification methods including SBM, WSBM and fuzzy decision trees using UCI datasets shows that pattern trees can obtain higher accuracy rates in classifications. It also shows that pattern trees perform more consistently than fuzzy decision trees. They are capable of finding classifiers with good generality, while fuzzy decision trees can easily fall into the trap of over-fitting.

According to two different configurations, simple pattern trees and pattern trees have been distinguished. The former not only produce high prediction accuracy, but also preserve compact tree structures, while the latter can produce even better accuracy, but as a compromise produce more complex tree structures. Subject to the particular demands (comprehensibility or performance), simple pattern trees or pattern trees provide an effective methodology for real world applications.

Weighted pattern trees have been proposed to reflect the nature that different trees may have different confidences. The experiments on British Telecom (BT) customer satisfaction dataset show that weighted pattern trees can slightly outperform pattern trees, and both are slightly better than fuzzy decision trees in terms of prediction accuracy. Finally, a limitation of pattern trees as revealed via BT dataset analysis is discussed.

Although the proposed pattern tree induction method shows very promising results, it does not mean that it cannot be improved. Other searching techniques can be used to find alternative pattern trees. For instance, Tabu search[2], which is notable for the capability of escaping from local optima, can be implemented. In addition, the underlying relation between decision trees and pattern trees needs more research work. The conversion between decision trees and pattern trees is worth investigating. Finally, research on assignment of weights to pattern trees is necessary. The current version simply makes use of similarity measures as weights. More sophisticated assignment may be more suitable and can therefore lead to higher accuracy.

Acknowledgments The authors would like to thank Marcus Thint for his helpful discussions on the work presented here.

References

1. Chen, S. M., Lee, S. H. and Lee, C. H., "A new method for generating fuzzy rules from numerical data for handling classification problems," *Applied Artificial Intelligence*, Vol. 15, pp. 645–664, 2001.
2. Glover, Fred and Laguna, Manuel, "Tabu search," *Boston : Kluwer Academic Publishers*, 1997.
3. Huang, Z. H. and Gedeon, T. D., "Pattern trees," *IEEE International Conference on Fuzzy Systems*, pp. 1784–1791, 2006.
4. Huang, Z. H., Gedeon, T. D., and Nikravesh, M.,"Pattern trees," submitted to *Transaction on Fuzzy Systems*, 2006.
5. Huang, Z. H., Nikravesh, M., Azvine, B., and Gedeon, T. D., "Weighted pattern trees: a case study with customer satisfaction dataset," to appear in *World Congress of the International Fuzzy Systems Association (IFSA)*, 2007.

6. Kóczy, L. T., Vámos, T. and Biró, G., "Fuzzy signatures," *EUROFUSE-SIC*, pp. 210–217, 1999.
7. Mendis, B. S. U., Gedeon T. D., and Kóczy, L. T., "Investigation of aggregation in fuzzy signatures," *3rd International Conference on Computational Intelligence, Robotics and Autonomous Systems*, 2005.
8. Newman, D. J., Hettich, S., Blake, C. L. and Merz, C.J., UCI Repository of machine learning databases [http://www.ics.uci.edu/ mlearn/MLRepository.html], 1998.
9. Nikravesh, M., "Soft computing for perception-based decision processing and analysis: web-based BISC-DSS," *Studies in Fuzziness and Soft Computing*, Vol. 164, pp. 93–188, Springer Berlin/Heidelberg, 2005.
10. Quinlan, J. R., "Decision trees and decision making," *IEEE Transactions on Systems, Man, and Cybernetics*, Vol. 20, No. 2, pp. 339–346, 1994.
11. Raju, G. V. S. and Kisner, R. A., "Hierarchical fuzzy control," *International Journal Control*, Vol. 54, No. 5, pp. 1201–1216, 1991.
12. Rasmani, K. A. and Shen, Q., "Weighted linguistic modelling based on fuzzy subsethood values," *IEEE International Conference on Fuzzy Systems*, Vol. 1, pp. 714–719, 2003.
13. Schweizer, B. and Sklar, A., "Associative functions and abstract semigroups," *Publ. Math. Debrecen*, Vol. 10, pp. 69–81, 1963.
14. Wang, L. X. and Mendel, J. M., "Generating fuzzy rules by learning from examples," *IEEE Transactions on Systems, Man, and Cybernetics*, Vol. 22, No. 6, pp. 1414–1427, 1992.
15. Wang, L. X. and Mendel, J. M., "Fuzzy Basis functions, universal approximation, and orthogonal least-squares learning," *IEEE Transactions on Neural Networks*, Vol. 3, No. 5, pp. 807–814, 1992.
16. Wong, K. W, Gedeon, T. D. and Kóczy L. T., "Construction of fuzzy signature from data: an example of SARS pre-clinical diagnosis system," *IEEE International Conference on Fuzzy Systems*, Vol. 3, pp. 1649–1654.
17. Yager R. R., "On ordered weighted averaging aggregation operators in multicritera decison making," *IEEE Transactions on Systems, Man and Cybernetics*, Vol. 18, pp. 183–190, 1988.
18. Yuan, Y. and M. J. Shaw, "Induction of fuzzy decision trees," *Fuzzy Sets and Systems*, Vol. 69, No. 2, pp. 125–139, 1995.
19. Zadeh, L. A., "Fuzzy sets," *Information and Control*, Vol. 8, pp. 338–353, 1965.
20. Zadeh, L. A., "A fuzzy-algorithmic approach to the definition of complex or imprecise concepts," *International Journal of Man-Machine Studies*, Vol. 8, pp. 249–291, 1976.