BULGARIAN ACADEMY OF SCIENCES

CYBERNETICS AND INFORMATION TECHNOLOGIES • Volume **22**, No 2
Sofia • 2022 Print ISSN: 1311-9702; Online ISSN: 1314-4081
DOI: 10.2478/cait-2022-0016

Enhancing Weak Nodes in Decision Tree Algorithm Using Data Augmentation

Youness Manzali[1], *Mohamed El Far*[1], *Mohamed Chahhou*[2], *Mohammed Elmohajir*[2]

[1]LPAIS Laboratory, Faculty of Sciences, USMBA Fez, Morocco
[2]Faculty of sciences, UAE, Tetouan, Morocco
E-mails: younes.manzali@usmba.ac.ma mohamed.elfar@usmba.ac.ma mchahhou@hotmail.com m.elmohajir@ieee.ma

Abstract**:** *Decision trees are among the most popular classifiers in machine learning, artificial intelligence, and pattern recognition because they are accurate and easy to interpret. During the tree construction, a node containing too few observations* (*weak node*) *could still get split, and then the resulted split is unreliable and statistically has no value. Many existing machine-learning methods can resolve this issue, such as pruning, which removes the tree's non-meaningful parts. This paper deals with the weak nodes differently; we introduce a new algorithm Enhancing Weak Nodes in Decision Tree* (*EWNDT*)*, which reinforces them by increasing their data from other similar tree nodes. We called the data augmentation a virtual merging because we temporarily recalculate the best splitting attribute and the best threshold in the weak node. We have used two approaches to defining the similarity between two nodes. The experimental results are verified using benchmark datasets from the UCI machine-learning repository. The results indicate that the EWNDT algorithm gives a good performance.*

Keywords**:** *Decision tree; virtual merging node; weak nodes; nodes similarity; data augmentation.*

1. Introduction

The Decision Tree (DT) algorithm is a well-known method for representing classifiers due to its simplicity, interpretation, and ease of use. The decision tree algorithm starts by using the whole data in the root node, then it divides the data into two subsets if the tree is binary or more than two otherwise based on the values of one or more attributes. The DT algorithm recursively divides each subset of examples into smaller ones, and it continues this process until all subsets contain a single class. The final subsets form the leaf nodes of the resulting tree. The classification process starts at the root node and follows the decision nodes' directions until it reaches a leaf. A weak node can be found during tree construction, with very few observations left; this confirms that as long as we go down in the tree, the reliability of further splitting nodes decreases due to the small sample size. To treat this issue, the literature on decision trees presents several methods. The most well-known among them is pruning. Pruning is a method that reduces the complexity of the tree by eliminating the non-useful parts to avoid over-fitting the dataset. Pruning the tree is an effective way to improve such a model's efficiency and classification accuracy. There are two types of pruning:

- Pre-pruning: during the construction, it stops the subdivision of the nodes based on some stopping criteria.
- Post-pruning: remove some part of the tree structure by recursive partitioning retrospectively.

Mahmood mentions in [7] that pre-pruning saves time by avoiding creating branches that will not be used in the final optimal tree. The challenge in this approach is to find an appropriate stopping rule [15]. Post-pruning consists of two phases: the construction and pruning phases. First, it allows growth to its full extent and then post-prunes the overfitted tree. Post-pruning methods require more time to build additional branches that are subsequently discarded. Still, this cost is offset against benefits because, in practice, post-pruning methods give better performance than pre-pruning. Pruning a decision tree is an effective technique to solve the problem of Overfitting. Pruned decision trees have a more straightforward structure and are expected to have higher generalization ability at the expense of classification accuracy. Their advantages have attracted the attention of many researchers, who have proposed several methods. However, the trade-off between structural simplicity and classification accuracy has not been well solved. This paper proposes another technique for dealing with weak nodes; we reinforce them by augmenting their data from another similar node. The rest of the paper is organized as follows. In Section 2, we give an overview of the related works. Section 3 presents in detail our contribution and the decision tree algorithm. In Section 4, we experimentally compare our algorithm with four state-of-the-art decision tree algorithms and other learning algorithms for the AUC (Area Under the ROC Curve) metric. Finally, Section 5 closes the paper and proposes future work.

2. Related work

Merging branches or nodes in decision trees is well-known and has been studied by several researchers. D. Ignatov and A. Ignatov [4] propose a new method for merging similar nodes where the similarity is calculated using two-sample test statistics. The merging process is executed after each splitting iteration. The test of similarity between two nodes compares the distribution of labels. The nodes from different branches are merged if the difference is statistically insignificant. This procedure resolves the classical problem of decision trees (progressive decrease of data quantity in the leaf nodes). It produces a more general structure (a directed acyclic graph), which can be extremely deep. Wu and Shi [13] propose merging identical subtrees. Two subtrees are identical if each subtree's root has the same splitting attribute and some corresponding branches of all the subtrees are similar. The algorithm reduces the size of the tree because it decreases the number of leaves and the depth of the final tree.

Y a n g, W a n g and Z h u [14] compare the complexity of a decision tree before and after merging branches, and present an algorithm for merging branches MID based on the support vector machine margin enlargement. This approach merges the pairs of nodes with the smallest distance between their data. H u, L i u and Y a n [3] propose a new merging branches algorithm, EPMID, based on information gain and predictability. The algorithm is built using the classical decision tree, but it uses the pre-pruning approach to merge the nodes with equal predictability. Two nodes have equal predictability if they have the same prediction class and the difference between their conditional probabilities of belonging to the same class is below a threshold defined by the user. J o o s t de N i j s [2] seeks to overcome some disadvantages in the decision tree by replacing it with a graph or DAG (a Directed Acyclic Graph) in which he merges branches of the tree that have a similar structure. V o l e s u and U t h e r [11] demonstrate that merging several nodes with a common parent into a single node could improve model accuracy. T a n and D o w e [10] resolve the problem of encoding internal repeated structures by introducing dynamic attributes in decision graphs.

L u n a et al. [27] propose a new algorithm, addTree, that creates a weak learner at each node, using gradient boosting on the entire dataset rather than using only the current node data. The algorithm allows the remaining data to influence the choice of the current split but with a potentially different weight. P f a h r i n g e r, H o l m e s and K i r k b y [8] propose an algorithm named HOT for Hoeffding Option Trees that creates optional tree branches simultaneously, replacing the branches with a lousy performance by optional ones. The algorithm's time complexity is high due to the construction of optional tree branches, while the precision has been dramatically improved. In [12], authors propose a new algorithm, Size Constrained Decision Tree (SCDT), which constructs a decision tree with a given number of leaf nodes. This approach allows dealing with the problem of tree complexity while remaining efficient. C o s t a et al. [23] propose an algorithm named the SVFDT algorithm for Strict Very Fast Decision Tree, which avoids excessive tree growth by applying additional rules to hold tree growth like the following assumptions:

1. A leaf node should split only if a minimum uncertainty of class assumption is associated with the examples.

2. All leaf nodes should observe a similar number of examples to be turned into split nodes.

3. According to previous statistics, the feature used for splitting should have a minimum relevance.

G a r c í a-M a r t í n et al. [24] extend the Hoeffding Trees with $n_{\min}$ adaptation. They propose a new algorithm that defines a unique and adaptive value for $n_{\min}$ on each leaf to check for possible division. This method allows the tree to grow faster on branches with clear divisions while delaying divisions on more uncertain branches. By retarding the growth of those branches with insufficient confidence, the algorithm saves a significant amount of energy on unnecessary tasks, with only minor effects on accuracy. G a n a i e, T a n v e e r and S u g a n t h a n [25] present a new approach for generating the oblique decision trees. At each non-leaf node, they use Bhattacharyya distance with a randomly selected feature subset to split the training data into two categories. They use a Twin Bounded Support Vector Machine (TBSVM) to get two clustering hyperplanes so that each hyperplane is closer to the data points of one group and as far as possible from the data points of the other group. Each non-leaf node is split to generate the decision tree based on these hyperplanes. Yang and Fong [26] propose an incremental optimization mechanism to solve imperfect data stream, Overfitting, and imbalanced class distribution problems. The mechanism is called Optimized Very Fast Decision Tree (OVFDT). OVFDT is a pioneer model with an incremental optimization mechanism that seeks to balance accuracy and tree size for data stream mining. It operates incrementally by a test-then-train approach. Three types of functional tree leaves improve the accuracy with which the tree model predicts a new data stream in the testing phase.

3. Proposed method

3.1. Example illustrating the problem of weak nodes

When trained to a significant depth, DT potentially has a high enough model complexity to achieve near-perfect predictions, and the existence of weak nodes can explain this. Therefore, these nodes will have a negative impact on the performance of the tree. So maximizing the contribution of each branch of the decision tree to optimal decision making becomes of high priority. Let us take an example of a weak node. During the tree construction, in a high depth, we consider a node containing three examples, two of them are positively labelled, and one is negatively labelled. Although this number of examples is very small, this node still needs to be divided; in reality, this number of examples is not enough to make a good decision. Moreover, the branch rooted with this node has no value statistically. Below we present an example of a binary decision tree. Given an input of three examples, the classifier follows the branch based on the condition satisfied by the splitting attribute until a leaf is reached, which specifies the prediction.

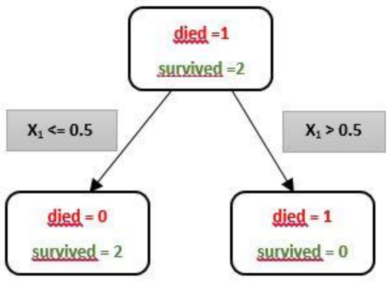

Fig. 1. Example of a node containing three examples

It seems that the classifier works perfectly since it splits the node into two pure nodes, but the question to ask is whether if the number of examples three is enough to make a reliable prediction. These three examples may be outliers, and therefore this type of node will be of poor quality, and subsequently, it does not generalize well; therefore, it will harm the tree performance. Here is the graphical representation in Figs 2-3.

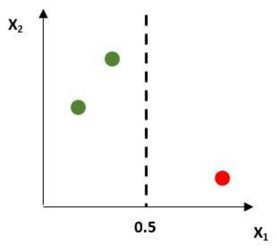

Fig. 2. The graphical representation of the node from Fig. 1

Now suppose an individual with x_1= 0.6 is labelled positive so that this classifier misclassifies this individual.

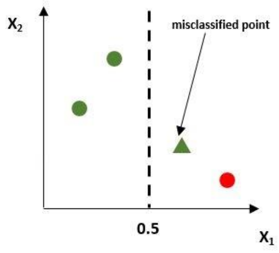

Fig. 3. Example of misclassification point

This is normal because the number of examples in the node is too low; making the prediction of class labels of this region is very difficult, since a smaller leaf makes the model more prone to capturing noise in training data. For this purpose, if we increase this node's data from another similar node's data, the prediction would become more robust. After the node's data augmentation, the splitting attribute and the best threshold for this attribute are recalculated. The threshold can be modified (Fig. 4), or the splitting attribute can be modified (Fig. 5). Thus, the node becomes more robust, and the decision becomes reliable.

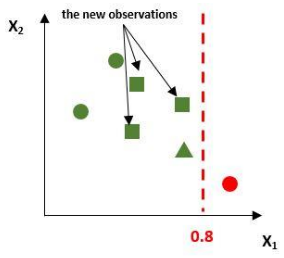

Fig. 4. The threshold correction of the node after data augmentation

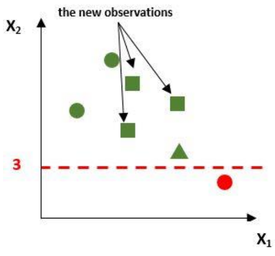

Fig. 5. Changing the splitting attribute after increasing data

3.2. Decision tree algorithm

A decision tree is a classifier that divides data recursively into homogeneous subsets to form classes. It is a supervised learning algorithm used in discrete or continuous data for classification or regression. Several algorithms have been presented until now, but the best known are C4.5, CART, QUEST, and CHAID.

3.2.1. C4.5 Algorithm

C4.5 algorithm [9] recursively splits a dataset of samples using a breadth-first or depth-first approach until all data subsets belong to one class. DT algorithm would begin by placing all the samples in the root node; the root node would then be placed in the fringe. The fringe is the set of nodes that still need to be divided further. We remove a node from the fringe at each step, and we create his children nodes and add them to the fringe, and we repeat this process until the fringe becomes empty.

3.2.2. CART

CART [1] stands for Classification and Regression Trees. It is distinguished by the fact that it constructs binary trees; specifically, each internal node has exactly two child nodes. The splits are selected using the Twoing criteria [22], which is a measure that evaluates the goodness of a splitting value. It measures the difference in probability that a category appears in the left descendant rather than the right descendant node, and the obtained tree is pruned. CART can handle both numeric and categorical data, efficiently handling outliers.

3.2.3. QUEST

Loh and Shih proposed QUEST [6] (Quick Unbiased Efficient Statistical Tree) in 1997. It is a tree-based classification algorithm that requires the target variable to be continuous. The computation speed of this algorithm is higher than this of other methods. The QUEST tree algorithm is known by the fact that it is not biased in the selection of split attributes, as opposed to the CART algorithm, which is biased towards the selection of the splitting attributes that allow more splits and those which have more missing values. The QUEST algorithm is more suitable for multiple categorical variables but can only perform binary classification. In the case of multiclass classification, it merges the classes into two superclasses.

3.2.4. CHAID

Kass proposed CHAID [5] (CHi-square Automatic Interaction Detection) in 1980. CHAID uses multi-way splits by default (multi-way splits mean that the current node is split into more than two nodes). It also prevents overfitting problems. A node is only split if a significance criterion is fulfilled.

3.3. The EWNDT Algorithm

The EWNDT algorithm is a modified version of the C4.5 Algorithm [2] using the breadth-first approach. Our main contribution is to increase weak node data (i.e., nodes with a number of examples less than a given thresholdβ, e.g., $\beta = 5$) when building trees. If a node is identified as weak, it should not be placed in the fringe but in a specific set named $pausedNodeSet$, containing all the weak nodes. Once the initial construction of the tree is finished, we start processing the weak nodes and iterate over all nodes in the $pausedNodeSet$. At each iteration, a node N would be removed from the $pausedNodeSet$; then, we search for similar nodes to N in the tree already built. Finally, when similar nodes are identified, we merge their data with the current node N and continue the construction process until the set of weak nodes becomes empty. The similarity between the two nodes is detailed in the next section. The exact process of our method is outlined below:

```
Algorithm EWNDT ($D$, β )
Input: Training data $D$, node's weakness indicator $\beta$
Output: A tree
Step 1.  $fringe, pausedNodeSet \leftarrow \emptyset$                // Global variables
Step 2.  if $D$ is pure OR other stopping criteria met
Step 3.      return a node with a corresponding class label
Step 4.  for all attributes $a \in D$
Step 5.      Compute information-theoretic criteria if we split on $a$
Step 6.  $a_{\text{best}} \leftarrow$ the best attribute according to the above computed criteria
Step 7.  $root \leftarrow$ create a new node child with $a_{\text{best}}$ as splitting attribute
Step 8.  Add the root to the $fringe$
Step 9.  while the $fringe$ is not empty
Step 10.     Pop a node $n$ from the fringe
Step 11.     $bestA \leftarrow$ the splitting attribute of the node $n$
Step 12.     for each possible value $v$ of $bestA$
Step 13.         $D_{\text{sub}} \leftarrow$ a subset of $D$ that have value $v$ for $bestA$
Step 14.         $child_v \leftarrow$ BuildNode ($D_{\text{sub}}$ , $\beta$)
Step 15.         Add $child_v$ as a descent from the node $n$ and label the edge
                 $\{n \rightarrow child_v\ \}\ as\ v$
Step 16. while the $pausedNodeSet$ is not empty
Step 17.     Pop a node $N$ from the $pausedNodeSet$
Step 18.     Search similar nodes to $N$
Step 19.     if more than one node is similar to $N$
Step 20.         Calculate the distance between them and $N$
Step 21.         if more than one node has the min distance of $N$
Step 22.             Merge the data of $N$ and all these nodes in $D_1$
Step 23.         else
Step 24.             CN ← the node that has min distance to $N$
Step 25.             Merge the data of $N$ and CN in $D_1$
Step 26.         Select Attribute $A_1$, which best classifies $D_1$
Step 27.         Mark the node $N$ with $A_1$ as splitting Attribute
Step 28.         Add the node $N$ to the $fringe$
Step 29.     else
Step 30.         SN ← the node which has maximum similarity to $N$
Step 31.         Merge the data of $N$ and SN in $D_2$.
Step 32.            Select Attribute $A_2$, which best classify $D_2$
Step 33.            Mark the node $N$ with $A_2$ as splitting Attribute
Step 34.            Add the node $N$ to the $fringe$
Step 35.   return root
```

```
Algorithm BuildNode ($D, \beta$ )
Input: Training data $D$, node's weakness indicator $\beta$
Output: A node,
Step 1.   if $D$ is pure OR other stopping criteria are met,
Step 2.       return a node with a corresponding class label.
Step 3.   else
Step 4.    $a_{\text{best}} \leftarrow$ the best attribute according to the above computed criteria
Step 5.       $child \leftarrow$ create a new node with $a_{\text{best}}$ as splitting attribute
Step 6.   if the number of examples of $D < \beta$
Step 7.       Add $child$ to the $pausedNodeSet$
Step 8.   else
Step 9.       Add $child$ to the $fringe$
Step 10.    return $child$
```

3.3.1. General conditions of the algorithm

The goal of the EWNDT algorithm is to increase the data of nodes marked as weak by merging their data with other similar nodes. This data merging is done virtually to determine the splitting attribute and the best threshold. Since this additional data is not kept in the child nodes, however, before merging, the following conditions must be verified:

- Two nodes belonging to the same branch (nodes on the same path from the tree root to a leaf node (Fig. 6)) cannot be merged.

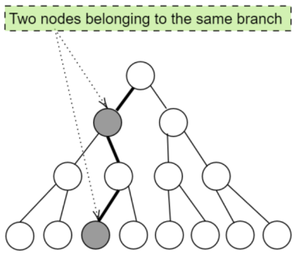

Fig. 6. Example of two nodes belonging to the same branch

- Two sibling nodes (nodes with the same parent node (Fig. 7)) cannot be merged.

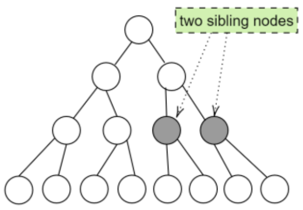

Fig. 7. Example of two sibling nodes

- If we find more than one node similar to the current one, we take the closest node (e.g., the node with the minimum distance from the current node). The method to calculate the distance between two nodes is presented in the following sub-section.
- If we find more than one similar node to the current one and these similar nodes are located at the same distance from the current one, we merge the data from all of these nodes with the current one to reselect a more reliable splitting attribute.

3.3.2. Distance between two nodes

The distance d between two nodes n_1 and n_2 is the minimum number of edges traversed from n_1 to n_2. It is calculated using the following formula:

(1) $$d = (n_1.\text{depth} - \text{ca.depth}) + (n_2.\text{depth} - \text{ca.depth}),$$

where ca is the common ancestor of the two nodes (i.e., the common node between the two branches carrying these two nodes).

The distance between two nodes n_1 and n_2 can be computed as the distance from n_1 to ca, plus the distance from ca to n_2. Fig. 8 explains the method to calculate the distance between two nodes.

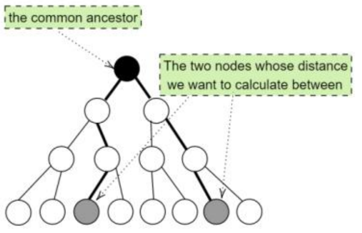

Fig. 8. The distance between two nodes

3.4. Node similarity

To merge two nodes in the tree, they must be similar, but there are several similarity criteria. We have used two approaches for comparing the similarity of the nodes:

3.4.1. Threshold approach

Two nodes, n_1 and n_2, in the tree are similar in the case of numerical attributes if they satisfy the following conditions:

- If they have the same splitting attribute.
- If they are labelled with the same class.
- If ($|n_1.\text{thresh} - n_2.\text{thresh}| \le \text{simThresh}$),

where $n_1.\text{thresh}$ and $n_2.\text{thresh}$ are the thresholds that give the best distribution of data in each node. simThresh is a parameter that measures the closeness between two thresholds (e.g., if simThresh $= 0.1$, two thresholds are considered equal if their difference is less than 0.1). The thresholds of every i-th splitting attribute z_i are normalized (i.e., its values are between 0 and 1) using the following formula:

(2) $$z_i = \frac{x_i - x_{\min}}{x_{\max} - x_{\min}},$$

where $x = (x_1, \dots, x_n)$ is the data before normalization, $x_{\max}$ is the maximum value for the attribute x_i, $x_{\min}$ is the minimum value for the attribute x_i, and z_i is now the i-th normalized data. We normalized the thresholds to facilitate the detection of similar ones to be generalized for the whole attribute space. In the case where thresholds are unnormalized, a $\mathrm{simThresh}$ value must be determined for each attribute.

Two nodes, n_1 and $n_{2,}$ in the tree are similar in the case of categorical attributes if they satisfy the following conditions:

- If they have the same splitting attribute.
- If they are labelled with the same class.
- The difference between their predictability is less than a threshold α [3],

where the predictability of a node N is defined as a conditional probability:

(3) $\mathrm{predictability}\ (N) = \mathrm{prob}(N.\mathrm{splitAttr} = N.\mathrm{thresh} \mid N.\mathrm{label})$

where $N.\mathrm{splitAttr}$ and $N.\mathrm{thresh}$ are the best attribute and threshold values to split the node N while $N.\mathrm{label}$ represents the class label with the majority vote in the node N. For example, let's take $\alpha = 0.1$. Two nodes n_1 and n_2 are merged if

(4) $\mid \mathrm{predictability}(n_1) - \mathrm{predictability}(n_2) \mid\ \leq\ 0.1.$

We only compute the predictability when the first two conditions are verified, i.e., we calculate predictability when the two nodes have the same class and the same splitting attribute.

3.4.2. Probabilistic approach

This approach does not differentiate between the numerical and the categorical attributes because it uses the predicted class probability, which represents the fraction of samples of the same class in a node. To better understand the prediction of class probabilities, we will take an example of a node containing five examples; two of them are positively labelled, and three examples are negatively labelled. So the probability of the positive class will be 2/5=0.4 while the probability of the negative one will be 3/5=0.6. Two nodes in a decision tree are similar if they satisfy the following conditions:

- If they have the same splitting attribute,
- If ($|n_1.\mathrm{prob} - n_2.\mathrm{prob} \mid\ \leq\ \mathrm{probThresh})$,

$\mathrm{probThresh}$ is a parameter that measures the similarity between two probabilities (e.g., if $\mathrm{probTresh}\ =\ 0.1$, two probabilities are considered similar if their difference is less than 0.1).

3.5. The strengths and limitations of the proposed method

When the tree is too shallow, there will be a chance that very few data points will be present at the bottom nodes, so the model could not reliably predict future cases and, thus, would have low generalizability. Hence, we augment the data of these nodes from other nodes, which are similar. This new approach allows us to build a reliable and robust splitting criteria tree. Furthermore, this will reflect on the accuracy of the tree. Our proposal's validity is tested; its advantages are demonstrated experimentally through illustrative examples and comparative analysis. The results discussion proved its usability and strengths over the existing approaches. However, the EWNDT algorithm has some weaknesses compared with other methods; the main drawback of the EWNDT algorithm is that it takes a large time to construct the tree in all datasets and generates a little more complex trees.

Time complexity of the algorithm. The algorithm builds more complex trees, but they construct more robust and reliable branches. To evaluate the time complexity of our algorithm, we start by calculating the complexity of each step. First, we consider the time complexity of the technique used in searching for the best split in the case of a numerical attribute. This technique sorts the unique values in the variable and selects a threshold between each consecutive value, and then it takes the one that minimizes the entropy. This sorting step takes time $O(n \times \log(n))$. Where n is the number of examples in the dataset, since we are doing that for all the features whose number is m, the total time complexity would be $O(m \times n \times \log(n))$. Finally, we will repeat this process for all weak nodes whose number is w then; the final complexity will be $O(m \times n \times \log(n) \times w)$.

4. Experimental results

4.1. Datasets

Experiments have been performed using twenty binary classification real-world datasets from the UCI repository to evaluate the proposed algorithm's performance. Table 1 shows the dataset, the size, the number of nominal and numerical attributes, and the percentage of examples of the minority class.

Table 1. Datasets characteristics

No	Datasets	Size	Attributes		% Min class
			Num.	Cat.	
1	transfusion	748	4	0	23.8
2	eighthr	2533	73	0	6.3
3	spambase	4601	57	0	39.4
4	Bank	41190	10	10	11.27
5	Hill_valley	606	100	0	49.7
6	ad	3263	1556	0	13.9
7	Cyl bands	540	18	20	42.2
8	Pima	768	8	0	34.9
9	adult	32561	0	14	24.1
10	caravan	5822	85	0	6.0
11	Connect 2c	67557	0	42	34.2
12	Liver	345	6	0	42.03
13	clean	6598	166	0	15.4
14	sick	2751	6	21	6.1
15	weka	310	6	0	22.3
16	spect	267	44	0	20.6
17	hypothyroid	3163	6	19	4.8
18	numerai	96320	21	0	49.5
19	kobe	25697	12	12	44.6
20	Sick-euth	2000	6	19	8.2

4.2. Parameters tuning

Table 2 summarizes the values obtained with our method and C4.5 in the parameter optimization process during the dataset training. The parameters of interest for the optimal parameterization are the max depth, the number of nodes in the constructed tree, and β the node's weakness indicator. We have used other parameters like the min sample leaf (the minimum number of samples required to be at a leaf node), but we show just the most critical parameters. We have used a grid search approach to find optimal parameters. For β, we have used the parameter space (10, 20, 30, 50, and 100), and for the maximum depth, we have used the following interval [3, 9].

Table 2. Optimal parameters for EWNDT and C4.5 for each dataset

Method	Dataset	Depth	β	Number of nodes	Dataset	Depth	β	Number of nodes
C4.5	1	5	–	31	11	9	–	423
EWNDTT		7	20	38		9	10	398
EWNDTP		7	20	37		9	10	403
C4.5	2	3	–	18	12	9	–	39
EWNDTT		4	30	21		9	20	37
EWNDTP		4	30	23		9	20	36
C4.5	3	6	–	45	13	9	–	84
EWNDTT		7	50	48		9	50	82
EWNDTP		7	50	48		9	50	81
C4.5	4	7	–	110	14	8	–	27
EWNDTT		7	50	107		8	50	27
EWNDTP		7	50	106		8	50	27
C4.5	5	8	–	19	15	3	–	6
EWNDTT		9	50	18		3	50	6
EWNDTP		9	50	20		3	50	6
C4.5	6	5	–	31	16	9	–	19
EWNDTT		7	20	38		9	20	21
EWNDTP		7	20	37		9	20	22
C4.5	7	3	–	18	17	9	–	25
EWNDTT		4	30	21		9	50	24
EWNDTP		4	30	23		9	50	26
C4.5	8	5	–	25	18	7	–	293
EWNDTT		5	30	23		7	50	271
EWNDTP		5	30	22		7	50	273
C4.5	9	9	–	193	19	9	–	166
EWNDTT		9	50	171		9	50	171
EWNDTP		9	50	175		9	50	172
C4.5	10	4	–	13	20	9	–	26
EWNDTT		4	50	12		9	50	21
EWNDTP		4	50	12		9	50	24

According to Table 2, those trees generated by EWNDT are much smaller than those generated by C4.5 in 11 of the 20 datasets; this can be explained by the fact that the data augmentation in weak nodes by similar data allows the classifier to find pure nodes faster.

4.3. Comparison with state-of-the-art methods

We have compared our two approaches (EWNDT with Threshold Similarity (ENWDTT) and EWNDT with Probability Similarity (ENWDTP)) with four state-of-the-art decision tree methods C4.5 [9], QUEST [6], CHAID [5], and CART [1]. We performed a 10-fold cross-validation. For each dataset, we show the arithmetic mean and the standard deviation of the ten runs. Table 3 presents the results of our benchmarks. Each model's performance has been evaluated using the AUC metric (area under the roc curve). We have used the AUC as a performance metric because the datasets have different distributions. Table 4 compares the different algorithms using accuracy as a performance metric. The best scores for each dataset are boldfaced.

Table 3. Evaluation of the two proposed approaches vs. four state-of-the-art decision tree algorithms using AUC score

No	C4.5	EWNDTT	EWNDTP	QUEST	CHAID	CART
1	70.6 ± 8.5	**71.9 ± 8.4**	**71.9 ± 7.4**	71.1 ± 5.1	71.0 ± 6.0	71.5 ± 8.6
2	70.3 ± 5.9	70.3 ± 5.9	**70.4 ± 5.9**	70.2 ± 5.7	70.1 ± 5.9	**70.4 ± 5.8**
3	95.5 ± 1.2	95.5 ± 1.2	95.7 ± 1.2	95.8 ± 1.3	**95.9 ± 1.2**	95.7 ± 1.1
4	83.1 ± 0.7	83.3 ± 0.8	83.7 ± 0.8	**84.1 ± 0.7**	83.2 ± 0.8	83.7 ± 0.8
5	56.3 ± 6.1	56.6 ± 4.5	56.3 ± 6.1	55.3 ± 5.1	**58.6 ± 4.7**	57.2 ± 6.1
6	90.9 ± 1.9	91.3 ± 1.6	91.4 ± 1.6	91.8 ± 1.9	91.7 ± 1.6	**92.4 ± 1.5**
7	77.3 ± 4.2	**78.3 ± 4.2**	78.0 ± 4.5	77.3 ± 4.1	**78.3 ± 4.2**	78.0 ± 4.4
8	80.4 ± 5.0	**82.4 ± 4.5**	**82.4 ± 4.0**	81.1 ± 5.0	80.9 ± 4.7	81.6 ± 4.0
9	88.1 ± 0.6	88.2 ± 0.6	**89.7 ± 0.6**	86.1 ± 0.7	85.2 ± 0.9	87.7 ± 0.6
10	73.7 ± 3.2	73.7 ± 3.2	73.7 ± 3.2	73.1 ± 3.2	73.7 ± 3.2	73.2 ± 3.2
11	82.4 ± 0.6	82.4 ± 0.6	82.4 ± 0.6	82.4 ± 0.6	82.4 ± 0.6	82.2 ± 0.6
12	68.5 ± 7.4	**72.6 ± 5.7**	71.8 ± 7.1	69.8 ± 7.3	70.5 ± 6.7	71.7 ± 7.1
13	89.3 ± 2.5	89.4 ± 2.5	89.8 ± 2.4	89.9 ± 2.7	90.3 ± 2.8	**90.7 ± 2.4**
14	89.2 ± 5.6	**90.4 ± 4.0**	**90.4 ± 4.0**	89.6 ± 4.6	90.1 ± 5.0	90.2 ± 5.2
15	89.0 ± 5.9	**89.5 ± 6.0**	**89.5 ± 6.0**	89.1 ± 5.8	88.7 ± 6.1	88.9 ± 6.0
16	60.0 ± 10.3	61.2 ± 10.6	62.3 ± 9.9	**63.0 ± 10.1**	62.3 ± 10.2	62.5 ± 9.9
17	78.0 ± 4.9	78.0 ± 4.9	79.0 ± 5.0	78.5 ± 4.8	78.2 ± 4.9	**79.1 ± 5.0**
18	52.2 ± 0.4	52.2 ± 0.4	52.2 ± 0.4	**53.0 ± 0.6**	51.7 ± 0.5	52.5 ± 0.5
19	68.6 ± 1.0	68.7 ± 1.0	68.7 ± 1.0	68.9 ± 1.2	69.3 ± 1.4	**69.6 ± 1.1**
20	87.2 ± 4.5	88.9 ± 5.6	**89.1 ± 4.9**	87.2 ± 4.7	88.9 ± 5.2	**89.1 ± 5.9**
Mean	77.5	78.2	78.4	77.9	78.0	78.4

Table 4. Evaluation of the two proposed approaches vs. four state-of-the-art decision tree algorithms using Accuracy

No	C4.5	EWNDTT	EWNDTP	QUEST	CHAID	CART
1	78.5 ± 4.5	**78.9 ± 4.4**	78.7± 4.3	78.1 ± 5.1	78.2 ± 5.6	77.9 ± 3.9
2	95.0 ± 3.8	94.7 ± 3.9	94.5 ± 3.6	94.7 ± 3.7	94.9 ± 3.6	**95.1 ± 3.5**
3	91.9 ± 1.2	92.0 ± 1.2	**92.1 ± 1.1**	90.8 ± 1.3	91.9 ± 1.4	91.6 ± 1.3
4	90.1 ± 1.5	90.6 ± 1.7	**90.7 ± 1.8**	89.9 ± 1.6	88.2 ± 1.8	90.0 ± 1.7
5	**52.9 ± 3.7**	52.8 ± 3.5	52.8 ± 3.8	52.3 ± 3.1	52.5 ± 3.9	52.2 ± 4.1
6	96.9 ± 1.9	97.3 ± 1.6	**97.4 ± 1.6**	97.5 ± 2.3	94.3 ± 2.0	96.4 ± 1.9
7	73.2 ± 3.6	**73.5 ± 3.5**	73.3 ± 3.6	73.1 ± 4.2	73.0 ± 4.0	73.2 ± 4.1
8	75.4 ± 4.3	75.7 ± 4.4	**76.2 ± 4.3**	75.1 ± 4.8	75.6 ± 4.3	75.5 ± 4.4
9	85.1 ± 2.1	85.0 ± 2.3	85.2 ± 2.2	85.3 ± 2.2	85.2 ± 2.0	**85.4 ± 1.9**
10	93.7 ± 2.8	93.7 ± 2.9	93.7 ± 2.9	93.7 ± 2.7	93.7 ± 2.6	**93.8 ± 3.3**
11	74.6 ± 5.4	78.6 ± 5.1	**78.2 ± 5.2**	77.3 ± 5.2	77.9 ± 5.3	76.8 ± 5.0
12	72.5 ± 6.4	**73.6 ± 5.7**	73.5 ± 6.1	72.8 ± 6.6	70.5 ± 6.7	71.7 ± 7.1
13	**93.3 ± 7.5**	91.4 ± 7.9	92.0 ± 7.4	91.9 ± 7.7	92.3 ± 7.8	92.7 ± 7.4
14	95.2 ± 3.6	95.4 ± 4.0	95.8 ± 4.0	94.6 ± 3.6	**96.1 ± 5.0**	95.2 ± 5.2
15	82.1 ± 7.3	83.3 ± 6.8	84.4 ± 6.5	**85.1 ± 5.9**	84.7 ± 6.1	85.0 ± 6.0
16	71.3 ± 2.5	**75.1 ± 2.7**	75.0 ± 2.8	73.0 ± 3.1	74.1 ± 2.8	73.9 ± 2.9
17	83.0 ± 3.9	84.4 ± 4.0	**84.9 ± 3.9**	83.5 ± 4.2	83.1 ± 4.4	83.8 ± **4.0**
18	52.8 ± 0.7	53.2 ± 0.6	53.3 ± 0.6	53.0 ± 0.6	**53.7 ± 0.5**	52.5 ± 0.5
19	69.5 ± 1.1	**70.7 ± 1.0**	69.7 ± 0.9	69.9 ± 1.1	69.8 ± 1.4	70.6 ± 1.1
20	90.6 ± 4.5	91.4 ± 4.6	**92.1 ± 3.9**	85.2 ± 4.7	86.7 ± 5.2	87.1 ± 5.1
Average	**80.9**	**81.6**	**81.7**	**80.8**	**80.8**	**81.0**

4.4. Comparison with different machine learning algorithms

To exhibit the worthiness of our method, we have compared it with the most well-known classification methods (Table. 5): Random Forests (RF) [16], Support vector machine (SVM) [17], k-Nearest Neighbors (KNN)[18], Logistic Regression (LR) [19] and Naive Bayes (NB) [20] presented in paper [21]. In the experiments, they use 10-fold cross-validation. Each dataset is split into three parts. 80% of the total examples in each dataset are used for training a classification model. Another 10% of the dataset examples are used for tuning the parameters of the classification model. The last 10% of the instances are used for testing only. If a classifier does not need to tune its parameters, the validation data will be merged into the training data. All classifiers use the same training and test data. We selected eight binary datasets from the used datasets and compared our method with the different algorithms.

Table 5. AUC results for different classification algorithms on eight data sets

Data sets	RF	SVM	KNN	LR	NB	EWNDTP	EWNDTT
pima	74.68	61.71	57.92	77.38	72.51	**82.40**	**82.40**
Ionosphere	**91.67**	80.56	88.89	88.89	80.56	83.30	82.2
German credit	**93.91**	64.52	69.39	65.35	74.08	81.50	80.4
splice	87.65	**92.59**	80.92	89.94	80.91	84.60	85.5
new_thyroid	87.50	**100.00**	96.88	96.88	96.88	92.40	93.6
spambase	**95.94**	89.71	66.46	64.42	59.40	95.70	95.5
phoneme	**85.86**	65.08	85.61	63.99	70.64	75.70	74.1
car	**95.08**	89.70	83.56	57.45	86.57	86.10	85.9
Average	**89.04**	80.48	78.70	75.54	77.69	85.21	84.95

4.5. Analysis of results

According to Table 3, the EWNDT algorithm with threshold similarity outperforms all the other methods with six (transfusion, bands, Pima, sick, weka, and liver) of the twenty datasets, while the difference between EWNDT and the other method in the remaining datasets is slight. The second approach that uses Probability's similarity outperforms all the other methods in seven of the twenty datasets. As we can see, the common factor between the datasets where our algorithm outperforms the other methods is that they have a few attributes. The first approach gives the best performance in numerical data sets, although the second approach does not differentiate between datasets. However, the second approach generally achieves the best performance compared to the other methods.

One can first observe from Table 4 that EWNDTP and EWNDTT are more accurate than the other algorithms. EWNDTT outperforms all the algorithms on five of the twenty datasets, and EWNDTP outperforms on seven of the twenty datasets. The differences between the two approaches and the other methods are often minor regarding the remaining datasets. As a matter of fact, EWNDT mean accuracy is better than the mean accuracy obtained by all other methods.

According to Table 5, RF has the best performance, followed by EWNDTT and EWNDTP. Moreover, we observe that EWNDTP and EWNDTP give the best performance over all other methods for the dataset Pima. In general, our approach ranks as the top one in most of the datasets.

5. Conclusion

This paper presents a new algorithm based on the Decision Tree model, which reinforces weak nodes in the tree by augmenting its data from other similar nodes using two similarity approaches. The first approach uses threshold similarity for the numerical attribute and conditional probability for the categorical attribute. The similarity criterion by the second approach uses nodes distribution. First, we have compared our method with four state-of-the-art decision tree methods (C4.5, CART, QUEST, and CHAID), which are the most popular ones. Then we have compared it to other classification algorithms. Therefore, the experimental results demonstrate that the new algorithm has fewer leaf nodes and higher classification performance. There are at least two areas for immediate future exploration.

First, we consider only the binary class in our experiment. Moreover, the problem may be more complicated if a non-binary class is considered. The second direction is to search for other similarity criteria between nodes. One more direction is when we find more than one node similar to the current node. We choose the node with the minimum distance from the current node, while other approaches can be applied.

References

1. Breiman, L., Je. Friedman, C. J. Stone, R. A. Olshen. Classification and Regression Trees. CRC Press, 1984.
2. Joost de Nijs. Decision Dags – a New Approach. Drown University, 1999.
3. Hu, D., Q. Liu, Q. Yan. Decision Tree Merging Branches Algorithm Based on Equal Predictability. – In: Proc. of International Conference on Artificial Intelligence and Computational Intelligence, Vol. **3**, 2009, pp. 214-218.
4. Ignatov, D., A. Ignatov. Decision Stream: Cultivating Deep Decision Trees. – In: Proc. of 29th IEEE International Conference on Tools with Artificial Intelligence (ICTAI'17), 2017, pp. 905-912.
5. Gordon, V. Kass. An Exploratory Technique for Investigating Large Quantities of Categorical Data. – Journal of the Royal Statistical Society: Series C (Applied Statistics), Vol. **29**, 1980, No 2, pp. 119-127.
6. Loh, Wei-Yin, Yu-Shan Shih. Split Selection Methods for Classification Trees. Sinica Statistica, 1997, pp. 815-840.
7. Gudapati, P., M. Mahmood, V. Kavuluru, M. Kuppa. A New Pruning Approach for Better and Compact Decision Trees. – International Journal on Computer Science and Engineering, Vol. **2**, 2010, pp. 2551-2558.
8. Pfahringer, B., G. Holmes, R. Kirkby. New Options for Hoeffding Trees. – In: Proc. of Australasian Joint Conference on Artificial Intelligence, Springer, 2007, pp. 90-99.
9. Ross Quinlan, J. C 4.5: Programs for Machine Learning. 1993.
10. Tan, P. J., D. L. Dowe. MML Inference of Decision Graphs with Multi-Way Joins. – In: Proc. of Australian Joint Conference on Artificial Intelligence, 2002, pp. 131-142.
11. Uther, W. T. B., M. M. Veloso. The Lumberjack Algorithm for Learning Linked Decision Forests. – In: Proc. of International Symposium on Abstraction, Reformulation, and Approximation, 2000, pp. 219-232.
12. Wu, Chia-Chi, Yen-Liang Chen, Yi-Hung Liu, Xiang-Yu Yang. Decision Tree Induction with a Constrained Number of Leaf Nodes. – Applied Intelligence, Vol. **45**, 2016, No 3, pp. 673-685.
13. Wu, X., B. Shi. New Algorithm of Simplifying the ID3 Decision Tree. – Journal of Hefei University of Technology, Vol. **27**, 2004, pp. 1565-1569.

14. Y a n g, C., X. W a n g, R. Z h u. A Strategy of Merging Branches Based on Margin Enlargement of SVM in Decision Tree Induction. – In: Proc. of IEEE International Conference on Systems, Man and Cybernetics, Vol. **1**, 2006, pp. 824-828.
15. Y a n g, S., H. F o n g. Incrementally Optimized Decision Tree for Noisy Big Data. – In: Proc. of 1st International Workshop on Big Data, Streams and Heterogeneous Source Mining: Algorithms, Systems, Programming Models and Applications, 2012, pp. 36-44.
16. B r e i m a n, L. Random Forests. – Machine Learning, Vol. **45**, 2001, No 1, pp. 5-32.
17. C o r t e s, C., V. V a p n i k. Support Vector Machine. – Machine Learning, Vol. **20**, 1995, No 3, pp. 273-297.
18. K e l l e r, J. M., M. R. G r a y, J. A. G i v e n s. A Fuzzy k-Nearest Neighbour Algorithm. – IEEE Transactions on Systems, Man, and Cybernetics, 1985, No 4, pp. 580-585.
19. W r i g h t, R. E. Logistic Regression. – In: L. G. Grim, P. R. Yarnolol, Eds. Reading and Understanding Multivariate Statistics, 1995, pp. 217-244.
20. S t o r k, D. G., R. O. D u d a, P. E. H a r t, et al. Pattern Classification. Wiley-Inter Science Publication, 2001.
21. Z h a n g, C., C. L i u, X. Z h a n g, G. A l m p a n i d i s. An Up-to-Date Comparison of State-of-the-Art Classification Algorithms. – Expert Systems with Applications, Vol. **82**, 2017, pp. 128-150.
22. S i n g h, S., P. G u p t a. Comparative Study ID3, Cart, and C4.5 Decision Tree Algorithm: A Survey. – International Journal of Advanced Information Science and Technology (IJAIST), Vol. **27**, 2014, No 27, pp. 97-103.
23. D a C o s t a, V. G. T., A. C. P. de L e o n F e r r e i r a, S. B. J u n i o r et al. Strict Very Fast Decision Tree: A Memory Conservative Algorithm for Data Stream Mining. – Pattern Recognition Letters, Vol. **116**, 2018, pp. 22-28.
24. G a r c í a-M a r t í n, E., N. L a v e s s o n, H. G r a h n et al. Energy-Aware Very Fast Decision Tree. – International Journal of Data Science and Analytics, Vol. **11**, 2021, No 2, pp. 105-126.
25. G a n a i e, M. A., M. T a n v e e r, P. N. S u g a n t h a n. Oblique Decision Tree Ensemble via Twin Bounded SVM. – Expert Systems with Applications, Vol. **143**, 2020, p. 113072.
26. Y a n g, H., S. F o n g. Incremental Optimization Mechanism for Constructing a Decision Tree in Data Stream Mining. – Mathematical Problems in Engineering, Vol. **2013**, 2013.
27. L u n a, J. M., E. D. G e n n a t a s, L. H. U n g a r et al. Building More Accurate Decision Trees with the Additive Tree. – Proceedings of the National Academy of Sciences, Vol. **116**, 2019, No 40, pp. 19887-19893.

Received: 14.03.2021; Second Version: 05.01.2022; Third Version: 05.04.2022; Accepted: 21.04.2022