Article

Interactive Decision Tree Learning and Decision Rule Extraction Based on the *ImbTreeEntropy* and *ImbTreeAUC* Packages

Krzysztof Gajowniczek * and Tomasz Ząbkowski

Department of Artificial Intelligence, Institute of Information Technology,
Warsaw University of Life Sciences-SGGW, 02-776 Warsaw, Poland; tomasz_zabkowski@sggw.edu.pl
* Correspondence: krzysztof_gajowniczek@sggw.edu.pl

Abstract: This paper presents two new R packages *ImbTreeEntropy* and *ImbTreeAUC* for building decision trees, including their interactive construction and analysis, which is a highly regarded feature for field experts who want to be involved in the learning process. *ImbTreeEntropy* functionality includes the application of generalized entropy functions, such as Renyi, Tsallis, Sharma-Mittal, Sharma-Taneja and Kapur, to measure the impurity of a node. *ImbTreeAUC* provides non-standard measures to choose an optimal split point for an attribute (as well the optimal attribute for splitting) by employing local, semi-global and global AUC measures. The contribution of both packages is that thanks to interactive learning, the user is able to construct a new tree from scratch or, if required, the learning phase enables making a decision regarding the optimal split in ambiguous situations, taking into account each attribute and its cut-off. The main difference with existing solutions is that our packages provide mechanisms that allow for analyzing the trees' structures (several trees simultaneously) that are built after growing and/or pruning. Both packages support cost-sensitive learning by defining a misclassification cost matrix, as well as weight-sensitive learning. Additionally, the tree structure of the model can be represented as a rule-based model, along with the various quality measures, such as support, confidence, lift, conviction, addedValue, cosine, Jaccard and Laplace.

Citation: Gajowniczek, K.; Ząbkowski, T. Interactive Decision Tree Learning and Decision Rule Extraction Based on the *ImbTreeEntropy* and *ImbTreeAUC* Packages. *Processes* **2021**, *9*, 1107. <https://doi.org/10.3390/pr9071107>

Received: 21 April 2021

Accepted: 22 June 2021

Published: 25 June 2021

Publisher's Note: MDPI stays neutral with regard to jurisdictional claims in published maps and institutional affiliations.

Copyright: © 2021 by the authors. Licensee MDPI, Basel, Switzerland. This article is an open access article distributed under the terms and conditions of the Creative Commons Attribution (CC BY) license (<https://creativecommons.org/licenses/by/4.0/>).

Keywords: classification trees; decision rules; interactive learning; machine learning

1. Introduction

Discovering knowledge from databases can be viewed as a non-trivial process for identifying valid, innovative, potentially useful and understandable patterns in data. In the majority of the existing data mining solutions, the visualization is used only at two certain phases of the mining process, i.e., in the first step (based on the original data) to perform explanatory analysis and during the last step to view the final results. Between those two phases, some automated data mining algorithm is used to extract patterns, for example, decision trees such as CART (Classification And Regression Tree) [1], C4.5 [2], CHAID (Chi-square Automatic Interaction Detection) [3] or *ImbTreeEntropy* and *ImbTreeAUC* [4].

Decision trees can be represented using a link–node diagram in which each internal node represents an attribute test, each link represents a corresponding test value or range of values and each leaf node contains a class label. The decision tree is usually built using a recursive top-down divide-and-conquer algorithm. Learning begins by considering the entire dataset, which is recursively divided into mutually exclusive subsets. The main decisions to make while learning are to decide which attribute should be used for splitting (along with the split value) and when to stop the splitting. Attribute selection is based on impurity measures, such as the Gini gain [1], information gain and gain ratio [2], or other non-standard measures presented in our previous work, such as generalized entropies Renyi, Tsallis, Sharma-Mittal, Sharma-Taneja and Kapur, as well as semi-global and global AUC measures [4].

Traditionally, decision trees are built by setting predefined parameters, running the algorithm and evaluating the tree structure created. Learning parameters are modified and the algorithm is restarted. This process is repeated until the user is satisfied with the decision tree. Often, users (experts) who are constructing decision trees have a broad knowledge in their domain but have little knowledge of decision tree construction algorithms. This is why experts do not know the exact meaning of all parameters and their influence on the created decision tree. Likewise, they have little or no knowledge of the inner workings of the algorithms involved. Due to this, the construction of a decision tree is often a trial and error process that can be very time consuming. Worse, field experts are not allowed to use their field knowledge to optimize the decision tree because the algorithm is used as a black box and does not allow them to control it. According to [5,6], there are several compelling reasons to incorporate an expert's domain knowledge and use visualization during the modeling process: (1) By providing appropriate data and knowledge visualizations, human pattern recognition abilities can be employed to increase the effectiveness of decision tree construction. (2) Through their active involvement, experts have a deeper understanding of the resulting decision tree. (3) By obtaining intermediate results from the algorithm, experts can provide domain knowledge (e.g., on important variables) to focus on further exploration of the algorithm. The use of domain knowledge has been identified as a promising approach to reduce computational effort and to avoid over-fitting. (4) An interactive construction process improves modeling efficiency, improves users' understanding of the algorithm and gives them more satisfaction when the problem is solved.

Recently, industry has recognized the inability to incorporate field knowledge into automated decision tree generation; it is believed that interactive learning can remove this gap. Some new methods have recently appeared [7,8] that try to more significantly involve the user in the data mining process and using visualization more intensively [9]; this new kind of approach is called visual data mining. In this work, we present *ImbTreeEntropy* and *ImbTreeAUC* packages (both packages are available at <https://github.com/KrzyGajow/ImbTreeEntropy> (accessed on 15 April 2021) and <https://github.com/KrzyGajow/ImbTreeAUC> (accessed on 15 April 2021)), which integrate automatic algorithms, interactive algorithms and visualization methods. The main contributions of the article can be summarized as follows:

- We implemented an interactive learning process that allows for constructing a completely new tree from scratch by incorporating specific knowledge provided by the expert;
- We implemented an interactive learning process by enabling the expert to make decisions regarding the optimal split in ambiguous situations;
- Both algorithms allow for visualizing and analyzing several tree structures simultaneously, i.e., after growing and/or when pruning the tree;
- Both algorithms support tree structure representation as a rule-based model, along with various quality measures;
- We show the implementation of a large collection of generalized entropy functions, including Renyi, Tsallis, Sharma-Mittal, Sharma-Taneja and Kapur, as the impurity measures of the node in the *ImbTreeEntropy* algorithm;
- We employed local, semi-global and global AUC measures to choose the optimal split point for an attribute in the *ImbTreeAUC* algorithm.

The remainder of this paper is organized as follows: Section 2 presents an overview of the similar research problems regarding interactive learning regarding both machine learning algorithms and decision tree algorithms. In Section 3, the theoretical framework of the proposed *ImbTreeEntropy* and *ImbTreeAUC* algorithms is presented. Section 4 presents various decision rule measures. Section 5 outlines the experiments and presents the discussion of the results. The paper ends with concluding remarks in Section 6.

2. Literature Review

Decision trees are based on conditional probabilities that allow for generating rules. A rule is a conditional statement that can be understood easily by humans and used within a database to identify a set of records. In some data mining applications, forecast accuracy is the only thing that really matters. It may not be important to know how the model works. In other cases, being able to explain the reason for a decision can be crucial. Decision trees are ideal for this type of application. There is a large number of commonly used measures of significance and interestingness for generated decision rules, such as support, confidence, lift, conviction, addedValue, cosine, Jaccard and Laplace. A good overview of different decision rule measures is provided by [10,11]. Some other applications of significance and interestingness measures are provided in [12] as an extension of unsupervised learning guided by some constraints and, in [13], as the application of the entropy measures to evaluate variables' importances.

A computer is often used to construct classification trees. Among the many packages/libraries, the most distinguished are those implemented in software such as R, Matlab, Orange, WEKA, SPSS and Statistica. Some solutions try to involve the expert more in this building process via interactivity. In the literature review, we focus only on the latter.

Currently, there are two commercial software programs supporting interactivity while constructing the tree. Orange software [14] provides a widget for manual construction and/or editing of classification trees by choosing the desired split attribute along with the split point at each step. The main drawback is that everything is done manually without a list of the possible splitting cut-offs and corresponding impurity measures (e.g., gain ratio or entropy). SAS software [15] provides a more easily comprehended solution. The interactive decision tree window provides the main menu with items, such as a subtree assessment plot, classification matrix and tree statistics. The most important (and most similar to our approach) is a table with competing rules, which displays the alternative and unused splitting variables.

The solution proposed by [16] enables the interactive building of decision trees using circle segments [17], which were later replaced by pixel-oriented bar charts [5]. The interactivity was expanded by allowing the computer to suggest a division to the user, allowing the computer to expand the subtree and automatically expanding the tree. Authors in [6] proposed similar interaction mechanics. Unfortunately, no support for pruning, optimization or decision tree analysis is available. In [18], users can automatically trim the decision tree after learning. However, the expert is again not involved in the pruning process, making it impossible to add domain knowledge. Software produced by [19] enables the construction of decision trees in the following approaches: node splitting (growing), leaf node label removal, leaf node labeling, removal of all descendant elements of a node (pruning) and node inspection. Unfortunately, this solution is completely manual and has no algorithmic support. The software produced by [20,21] allows for interactive learning by painting the area of the star coordinate chart and assigning it with an appropriate class label. This approach is fully manual without algorithmic support, similarly to the method proposed by [22]. Works by [23–25] allow for constructing bivariate decision trees and drawing a split polygon and split lines by visualizing them as a scatter chart. In [23], users are not algorithmically supported. The software produced by [23–25] supports users algorithmically by suggesting the best dividing line.

The frameworks most similar to our solution are the frameworks presented in [7,8]. BaobabView, presented in [7], supports field experts in the growing, pruning, optimization and analysis of decision trees. In addition, they present a scalable decision tree visualization that is optimized for exploration. In [8], the authors expanded MIME [26], which is an existing platform for pattern discovery and exploration, enabling tree construction from scratch and customizing existing trees by presenting good and troublesome splits in the data.

Finally, it worth mentioning some of the most recent works in this field. In [27], the authors focused on the construction of strong decision rules to strengthen the existing

decision tree classifier to improve the classification accuracy. The evaluation result is obtained using the inference engine with forward reasoning searches of the rules until the correct class is determined. In [28], so-called human-in-the-loop machine learning was proposed to elicit labels for data points from experts or to provide feedback on how close the predicted results are to the target. This simplifies all the details of the decision-making process of the expert. The experts have the option to additionally produce decision rules describing their decision making; the rules are expected to be imperfect but do give additional information. In particular, the rules can extend to new distributions and hence enable significantly improving the performance for cases where the training and testing results differ.

3. The Algorithm for Interactive Learning

In the following subsections, whenever it is possible, in the rounded brackets, there are calls of the certain parameters of the algorithm along with the arguments corresponding to a given situation. For more details regarding the theoretical aspects of the presented *ImbTreeEntropy* and *ImbTreeAUC* packages, please see our previous work [4].

3.1. Notations

In supervised learning problems, we receive a structured set of n labeled observations $D = \{(x_i^T, y_i)\}_{i=1}^n$. Each data point (x_i^T, y_i) consists of an object $x \in X^p$ from a given p -dimensional input space $X^p = \{X_1, \dots, X_l, \dots, X_p\}$ and has an associated label y_i , where y_i is a real value for the regression or (as in this paper) a category for the classification task, i.e., $Y \in \{1, \dots, j, \dots, k\}$. On the basis of a set of these observations, which is usually referred to as training data, supervised learning methods try to deduce a function that can successfully determine the label y^* of some previously invisible input x^* .

3.2. Algorithm Description

The main functions of the *ImbTreeEntropy* and *ImbTreeAUC* packages supporting interactivity are *ImbTreeEntropyInter()* and *ImbTreeAUCInter()*. At the very beginning, both functions call the *StopIfNot()* function to check whether all parameters take appropriate values, e.g., if there are no missing values or if combinations of some parameters are valid [4]. Next, an $n \times k$ probability matrix is created by the *AssignProbMatrix()* function. This structure is required for AUC-based learning (*ImbTreeAUC*) and for storing final results for both packages. The last initial step calls the *AssignInitMeasures()* function, which computes and assigns initial values of the required measures, such as AUC, entropy, depth of the tree, number of observations in the dataset or a misclassification error [4].

After all the preparation steps are finished, the main building function is called (Algorithm 1), which is the same for both packages. In comparison to the standard *BuildTree()* function, which constructs the tree in an automated manner, this time, there are some differences in lines 4–9. In standard learning, the *BestSplitGlobal()* function calculates the best attribute for splitting (along with the best split point) of a given node. Then, the *IfImprovement()* function checks to see whether it is possible to perform a split, i.e., if the improvement is greater than the cp parameter, if both possible nodes (siblings) have more than $minobs$ instances or if both children are pure (should be converted into leaves). In interactive learning mode, the *BestSplitGlobalInter()* function provides statistics for all splitting points and for all attributes. Suppose that there are three continuous attributes having four distinct values. Therefore, the algorithm has to perform $s-1$ operations, where s is the number of distinct values of the given attribute. As a result, this function outputs a table with $3 \times 3 = 9$ (nine possible splits at a given time) rows along with the corresponding statistics needed for the tree building.

Based on this table, the *IfImprovement()* function, similar to automatic learning, assesses which of these possible splits meet the predefined conditions. If this set of possible splits is not null (sixth line), then the *ClassProbLearn()* function has to recognize which interactive learning type should be performed. There are three different types. The first allows for

building the entire tree from the scratch, enabling the expert to make a decision during each partition. The remaining two types allow for making a decision only in ambiguous situations.

How this ambiguity is defined? Let us consider the following examples with two ambiguity meanings. For the first one, let us assume that we have a multiclass classification problem with four classes and we would like to split a particular node. In the left child, the estimated probabilities of the two of them assigned by a particular node are similar, e.g., 0.44 and 0.46. The rest of the classes have marginal probabilities, e.g., 0.06 and 0.04. In the second node, these probabilities are equal to 0.48, 0.47, 0.02 and 0.03. The automotive algorithm would choose the second class (left child) or the first class (right child) as a label. The point is that such a division leads to a situation where one observation can change the probabilities and even the final class label. For the expert, it may be better to choose a different split (cut-off or even the attribute), admittedly with a lower information gain or gain ratio, but with a more pronounced difference in the probabilities. Therefore, if the automotive algorithm generates a division such that one class will be elevated (simple decision), we leave it like that, and only if such divisions with disputable probabilities are possible, then the expert makes the decision.

Algorithm 1: BuildTreeInter algorithm.

Input: *node* (*N*), *Yname*, *Xnames*, *data*, *depth*, *levelPositive*, *minobs*, *type*, *entropypar*, *cp*, *ncores*, *weights*, *AUCweight*, *cost*, *classThreshold*, *overfit*, *ambProb*, *topSplit*, *varLev*, *ambClass*, *ambClassFreq*

Output: Tree (*T*)

```

/1/  node.Count = nrow(data) // number of observations in the node
/2/  node.Prob = CalcProb() // assign probabilities to the node
/3/  node.Class = ChooseClass() // assign class to the node
/4/  splitrule = BestSplitGlobalInter() // calculate statistics of all possible best local splits
/5/  isimpr = IfImprovement(splitrule) // check if the possible improvements are greater than the
threshold cp
/6/  if isimpr is NOT NULL then do
/7/    ClassProbLearn() // depending on the input parameters set up learning type
/8/    splitrule = InteractiveLearning(splitrule) // start interactive learning (see Algorithm 2)
/9/  end
/10/ if stop = TRUE then do // check if stopping criteria are met
/11/   UpdateProbMatrix() // update global probability matrix
/12/   CreateLeaf() // create leaf with various information
/13/   return N
/14/ else do
/15/   BuildTreeInter(left N) // build recursively tree using left child obtained based on the
splitrule
/16/   BuildTreeInter(right N) // build recursively tree using right child obtained based on the
splitrule
/17/ end
/18/ return T

```

The second meaning of ambiguity is related to the number of observations (frequencies) of a given class in a node. Following the previous example with the four classes, let us assume that class frequencies in the entire dataset having 100 instances are: 10 (10%), 15 (15%), 35 (35%) and 40 (40%). Later, let us assume that the second class is very important for us and we would like to build a model that allows for classifying instances from this class without any errors. Of course, it might happen that the best possible split has the highest information gain/gain ratio improvement, but it splits observations from this class into both children. In this interactive learning type, the expert can indicate which classes are of importance and how many observations (measured in frequencies) should fall into a node in order to make a decision; otherwise, the algorithm will make a decision

based on the best split. Let us assume that the expert set up the following thresholds: 50%, 0%, 100% and 100%, which means that the decision has to be made only when, in a particular node, there are more instances than $10 * 50\% = 5$, $15 * 0\% = 0$, $35 * 100\% = 35$ and $40 * 100\% = 40$. This means that the third and fourth classes do not have any influence on the interactivity, while the expert will make a decision whenever there are any observations from the second class.

Coming back to Algorithm 1, line 8 calls the *InteractiveLearning()* function, whose body is presented in Algorithm 2. At the end, this function outputs the decision made by the expert. Depending on the type of learning, during the first step, the function assesses whether interactive learning can be performed, i.e., whether there are any possible splits that are under or above the predefined thresholds (see the previous two paragraphs). If not, there is no interactivity and as a split rule, the best split rule is outputted (third line); this is exactly the same behavior as in the standard automatic learning. Next, all possible splits are sorted in descending order based on the gain ratio; this operation causes the expert to see the trees' structures, starting from the best one. Before the entire learning phase, the expert has to decide whether the possible best splits are derived at the attribute level (higher) or the split point for each attribute (lower), along with the *topSplit* parameter, which is the number of best splits, i.e., final trees structure to be presented. If the *varLev* parameter is set to "True," this means that the expert gets the best splits, one for each variable, while "False" means that for the best splits, it might happen that the expert receives *topSplit* splits coming from only one variable.

Algorithm 2: Interactive learning algorithm.

Input: *tree* (*T*), *splitRule*, *Yname*, *Xnames*, *data*, *depth*, *levelPositive*, *minobs*, *type*, *entropyPar*, *cp*, *ncores*, *weights*, *AUCweight*, *cost*, *classThreshold*, *overfit*, *ambProb*, *topSplit*, *varLev*, *ambClass*, *ambClassFreq*

Output: Decision regarding *splitRule*

```

/1/  stopcond = StopCond(splitRule) // check if interactive learning should be performed
/2/  if stopcond = TRUE then do // if possible splits are below or above the thresholds, no
interaction required
/3/    splitrule = Sort(splitRule) // take the best split by sorting in descending order splits
according to the gain ratio
/4/    return splitrule
/5/  end
/6/  splitruleInter = Sort(splitRule) // sort in descending order splits according to the gain ratio
/7/  if varLev = TRUE then do
/8/    splitruleInter = BestSplitAttr(splitruleInter, topSplit) // take the topSplit best splits of each
attribute
/9/  else do
/10/   splitruleInter = BestSplitAll(splitruleInter, topSplit) // take the globally best topSplit splits
/11/ end
/12/ writeTree(T) // create output file with the initial tree (current tree structure)
/13/ for each splitruleInter do
/14/   Tree = Clone(T) // deep clone of the current tree structure
/15/   BuildTree(Tree, left N) // build recursively tree using left child obtained based on the
splitruleInter
/16/   BuildTree(Tree, right N) // build recursively tree using right child obtained based on the
splitruleInter
/17/   if overfit then do
/18/     PruneTree(Tree) // prune tree if needed
/19/   end
/20/   writeTree(Tree) // create output file with the possible tree
/21/ end
/22/ splitrule = ChooseSplit(splitruleInter) // choose the desired split
/23/ return splitrule

```

Before starting the main loop (lines 13–22), for comparison purposes, the current structure of the tree is presented (12th line). This is very important because line 14 creates a deep clone of the current tree structure, which will be further expanded by the *BuildTree()* function. In other words, the main loop will create a couple of new trees (based on different possible splits) each time, starting from the current structure. Unlike the *BuildTreeInter()* function presented in Algorithm 1, the *BuildTree()* function from lines 15 and 16 expands the current tree structure in the standard automatic manner. Other existing solutions allow only for making a decision at this point, while our solution shows the expert what the entire tree structure would be if they made a particular decision. After the growing phase, it is possible to prune the tree only if the input parameter *overfit* does it globally (see Algorithm 1 in [4]). After all steps are completed, the final possible tree structure is outputted. When all possible trees are built, the algorithm stops and waits for the decision provided by the expert (22nd line). It should be noted that some tree structures would be the same as the current structure. The reason is that the split is valid only if this split and/or the created descendants choose a different class label. Therefore, after growing, the tree might shrink. It might happen that the expert (possibly by mistake) chooses this particular split. Next, the decision is exported to Algorithm 1, and if possible, the new current tree structure is created by adding the chosen split. The entire algorithm is called recursively (collaborating with the expert) until the final tree is built. Additionally, for the benefit of readers, the process of interactive learning is presented in Figure 1.

Figure 1. Flowchart of the interactive learning algorithm.

3.3. Software Description

Both packages have two main components: a set of functions that allows for building the tree in standard R-like console mode, and a set of functions that allows for deploying the Shiny web application, which incorporates all package functionalities in a user-friendly environment. In order to start interactive learning in the console mode, the user has to call either *ImbTreeEntropyInter()* or *ImbTreeAUCInter()* functions, along with the corresponding parameters (we list additional parameters required for interactive learning below, other parameters are described in [4]):

- *ambProb*: Ambiguity threshold for the difference between the highest class probability and the second-highest class probability per node, below which the expert has to make a decision regarding the future tree structure. This is a logical vector with one element, it works when the *ambClass* parameter is NULL.

- *topSplit*: Number of best splits, i.e., final tree structures to be presented. Splits are sorted in descending order according to the gain ratio. It is a numeric vector with one element.
- *varLev*: Decision indicating whether possible best splits are derived at the attribute level or the split point for each attribute.
- *ambClass*: Labels of classes for which the expert will make a decision during the learning. This character vector of many elements (from 1 up to the number of classes) should have the same number of elements as the vector passed to the *ambClassFreq* parameter.
- *ambClassFreq*: Classes frequencies per node above which the expert will make a decision. This numeric vector of many elements (from 1 up to the number of classes) should have the same number of elements as the vector passed to the *ambClass* parameter.

When using the user-friendly environment, the expert should use the fourth main panel (out of five) called “Interactive Model.” After importing the data, in the “Fit Model” sub-panel, there are additional parameters and a section (comparing to the standard learning) specifying how to understand an ambiguous situation (Figure 2).

Figure 2 consists of two panels, (a) and (b), showing the Interactive Model panel settings.

Panel (a) shows the following settings:

- Top splits**: A numeric input field with the value 2.
- Attribute level**: A dropdown menu with the value TRUE.
- Ambiguity type**: A dropdown menu with the value Probability.
- Ambiguous probability**: A numeric input field with the value 1.

Panel (b) shows the following settings:

- Provide Tree number**: A dropdown menu with the value 1.
- Confirm Decision and Resume Learning**: A button.

Figure 2. (a,b) Additional parameters and sections in the Interactive Model panel.

First, the user has to specify the number of best splits to be presented and indicates whether possible best splits are derived at the attribute level or the split point for each attribute. The ambiguous type chooses between the “Probability” or “Class”. The ambiguous probability decides the threshold for the difference between the highest class probability and the second-highest class probability per node, below which the expert has to make a decision regarding the subsequent tree structure. When the ambiguous type is set to “Class”, an additional “Ambiguity matrix” sub-panel appears where the user has to decide the ambiguous classes, i.e., labels of classes for which the expert will make a decision during the learning. Ambiguous class frequencies are frequencies per node, above which the expert will make a decision.

During the learning, just below the “Start learning” button, an additional section appears that allows for choosing the desired tree structure and resumes the learning (Figure 2b).

4. Extracting Decision Rules

Decision trees and decision rules are very common knowledge representations that are used in machine learning. Algorithms for learning decision trees are simple to implement and relatively fast. Algorithms for learning decision rules are more complex, but the resulting rules may be easier to interpret. To exploit the advantages of both knowledge representations, learned decision trees are often converted into rules. This subsection presents a collection of measures of significance and interestingness of each extracted rule, which can also be found in association rule mining (the most established measures are listed below; however, the package provides more measures that can be easily derived based on the measures listed below).

The frequency of the rule $R : A \rightarrow Y$ (e.g., $A : x^{(1)} \leq 2 \wedge x^{(2)} > 3, Y : y = 0$) is measured with its support in the dataset. The support gives the proportion of observations (described by the attributes vector x^T) that fall into the following rule:

$$\text{Sup}(A \rightarrow Y) = \frac{1}{n} \sum_{i=1}^n I(x_i^T \in A \wedge y_i = j), \quad (1)$$

where I is an indicator function. Confidence is defined as the probability of seeing the rule's consequent under the condition that the transactions also contain the antecedent:

$$\text{Conf}(A \rightarrow Y) = \frac{1}{\sum_{i=1}^n I(x_i^T \in A)} \sum_{i=1}^n I(x_i^T \in A \wedge y_i = j), \quad (2)$$

The next measure is the lift, which measures how many times more often A and Y occur together than expected if they were statistically independent:

$$\text{Lift}(A \rightarrow Y) = \frac{\text{Conf}(A \rightarrow Y)}{\sum_{i=1}^n I(y_i = j)}, \quad (3)$$

Conviction was developed as an alternative to confidence, which was found to not adequately capture the direction of associations. Conviction compares the probability that A appears without Y if they were dependent on the actual frequency of the appearance of A without Y :

$$\text{Conviction}(A \rightarrow Y) = \frac{1 - \sum_{i=1}^n I(y_i = j)}{1 - \text{Conf}(A \rightarrow Y)}, \quad (4)$$

A null-invariant measure for dependence is the Jaccard similarity between A and Y , which is defined as:

$$\text{Jaccard}(A \rightarrow Y) = \frac{\text{Sup}(A \rightarrow Y)}{\sum_{i=1}^n I(x_i^T \in A) + \sum_{i=1}^n I(y_i = j) - \text{Sup}(A \rightarrow Y)}, \quad (5)$$

5. Results

5.1. Interactive Example

In this section, we show illustrative examples of how to build decision trees in an interactive manner using the three approaches described earlier, i.e., building an entire tree from scratch and using two types of ambiguous situations.

The example was based on the commonly known Iris data set (three classes) that was downloaded from the UCI Machine learning repository [29]. The data consists of four continuous features and a single three-class output that identifies the type of an iris plant. The data set is balanced and it contains 150 observations, i.e., 50 instances for each class. Examples are presented based on the *ImbTreeEntropy* package (see Figure 3); however, the usage of the *ImbTreeAUC* package would be the same.

First, we built the tree from scratch. After uploading the file, the target variable was chosen as "Species," which had three labels: "setosa," "versicolor" and "virginica." All four variables were chosen as possible splitting attributes: "Sepal.Length," "Sepal.Width," "Petal.Length" and "Petal.Width." Parameter *topSplit* was set to 3. All the remaining parameters used their default values, e.g., Shannon entropy, depth equals 5 and minimal number of observation per node equals 10. Next, depending on the used mode, the user ran the *ImbTreeEntropyInter()* function (standard console mode) or pushed the start button on the Shiny application. In console mode, the program created *topSplit* + 1 trees (in the chosen working directory), with the following convention: *tree0* presented the current tree structure from which possible trees were built, *tree1*, ..., *tree_{topSplit}*. The Shiny application produced the output presented in Figure 3.

Current Tree. Accuracy: 33.33%.

levelName	Decision	Number	Count	Class	Probability	Leaf
1	Root		1	150	setosa (0.33, 0.33, 0.33)	NA

Tree1. Accuracy: 96.00%.

levelName	Decision	Number	Count	Class	Probability	Leaf
1	Root		1	150	setosa (0.33, 0.33, 0.33)	
2	--Petal.Length <= 1.9	*NOW*	2	50	setosa (1.00, 0.00, 0.00)	*
3	°--Petal.Length > 1.9	*NOW*	3	100	versicolor (0.00, 0.50, 0.50)	
4	--Petal.Width <= 1.7		6	54	versicolor (0.00, 0.91, 0.09)	*
5	°--Petal.Width > 1.7		7	46	virginica (0.00, 0.02, 0.98)	*

Tree2. Accuracy: 96.00%.

levelName	Decision	Number	Count	Class	Probability	Leaf
1	Root		1	150	setosa (0.33, 0.33, 0.33)	
2	--Petal.Width <= 0.6	*NOW*	2	50	setosa (1.00, 0.00, 0.00)	*
3	°--Petal.Width > 0.6	*NOW*	3	100	versicolor (0.00, 0.50, 0.50)	
4	--Petal.Width <= 1.7		6	54	versicolor (0.00, 0.91, 0.09)	*
5	°--Petal.Width > 1.7		7	46	virginica (0.00, 0.02, 0.98)	*

Tree3. Accuracy: 88.67%.

levelName	Decision	Number	Count	Class	Probability	Leaf
1	Root		1	150	setosa (0.33, 0.33, 0.33)	
2	--Sepal.Length <= 5.4	*NOW*	2	52	setosa (0.87, 0.12, 0.02)	
3	--Petal.Length <= 1.6		4	40	setosa (1.00, 0.00, 0.00)	*
4	°--Petal.Length > 1.6		5	12	versicolor (0.42, 0.50, 0.08)	*
5	°--Sepal.Length > 5.4	*NOW*	3	98	virginica (0.05, 0.45, 0.50)	
6	--Petal.Length <= 4.7		6	43	versicolor (0.12, 0.88, 0.00)	*
7	°--Petal.Length > 4.7		7	55	virginica (0.00, 0.11, 0.89)	*

(a)

Current Tree. Accuracy: 63.33%.

levelName	Decision	Number	Count	Class	Probability	Leaf
1	Root		1	150	setosa (0.33, 0.33, 0.33)	
2	--Sepal.Length <= 5.4		1	2	52	setosa (0.87, 0.12, 0.02)
3	--Petal.Length <= 1.6		2	4	40	setosa (1.00, 0.00, 0.00)
4	°--Petal.Length > 1.6		2	5	12	versicolor (0.42, 0.50, 0.08)
5	°--Sepal.Length > 5.4		1	3	98	virginica (0.05, 0.45, 0.50)

Tree1. Accuracy: 88.67%.

levelName	Decision	Number	Count	Class	Probability	Leaf
1	Root		1	150	setosa (0.33, 0.33, 0.33)	
2	--Sepal.Length <= 5.4		1	2	52	setosa (0.87, 0.12, 0.02)
3	--Petal.Length <= 1.6		2	4	40	setosa (1.00, 0.00, 0.00)
4	°--Petal.Length > 1.6		2	5	12	versicolor (0.42, 0.50, 0.08)
5	°--Sepal.Length > 5.4		1	3	98	virginica (0.05, 0.45, 0.50)
6	--Petal.Length <= 4.7	*NOW*	6	43	versicolor (0.12, 0.88, 0.00)	*
7	°--Petal.Length > 4.7	*NOW*	7	55	virginica (0.00, 0.11, 0.89)	*

Tree2. Accuracy: 89.33%.

levelName	Decision	Number	Count	Class	Probability	Leaf
1	Root		1	150	setosa (0.33, 0.33, 0.33)	
2	--Sepal.Length <= 5.4		1	2	52	setosa (0.87, 0.12, 0.02)
3	--Petal.Length <= 1.6		2	4	40	setosa (1.00, 0.00, 0.00)
4	°--Petal.Length > 1.6		2	5	12	versicolor (0.42, 0.50, 0.08)
5	°--Sepal.Length > 5.4		1	3	98	virginica (0.05, 0.45, 0.50)
6	--Petal.Width <= 1.7	*NOW*	6	52	versicolor (0.10, 0.83, 0.08)	*
7	°--Petal.Width > 1.7	*NOW*	7	46	virginica (0.00, 0.02, 0.98)	*

Tree3. Accuracy: 88.00%.

levelName	Decision	Number	Count	Class	Probability	Leaf
1	Root		1	150	setosa (0.33, 0.33, 0.33)	
2	--Sepal.Length <= 5.4		1	2	52	setosa (0.87, 0.12, 0.02)
3	--Petal.Length <= 1.6		2	4	40	setosa (1.00, 0.00, 0.00)
4	°--Petal.Length > 1.6		2	5	12	versicolor (0.42, 0.50, 0.08)
5	°--Sepal.Length > 5.4		1	3	98	virginica (0.05, 0.45, 0.50)
6	--Sepal.Width <= 3.3	*NOW*	6	87	virginica (0.00, 0.49, 0.51)	*
7	--Petal.Length <= 4.7		12	37	versicolor (0.00, 1.00, 0.00)	*
8	°--Petal.Length > 4.7		13	50	virginica (0.00, 0.12, 0.88)	*
9	°--Sepal.Width > 3.3	*NOW*	7	11	setosa (0.45, 0.09, 0.45)	*

(c)

Current Tree. Accuracy: 51.78%.

levelName	Decision	Number	Count	Class	Probability	Leaf
1	Root		1	150	setosa (0.33, 0.33, 0.33)	NA
2	--Sepal.Length <= 5.4		1	2	52	setosa (0.87, 0.12, 0.02)
3	°--Sepal.Length > 5.4		1	3	NA	(NA) NA

Tree1. Accuracy: 52.44%.

levelName	Decision	Number	Count	Class	Probability	Leaf
1	Root		1	150	setosa (0.33, 0.33, 0.33)	
2	--Sepal.Length <= 5.4		1	2	52	setosa (0.87, 0.12, 0.02)
3	--Petal.Length <= 1.6	*NOW*	4	40	setosa (1.00, 0.00, 0.00)	*
4	°--Petal.Length > 1.6	*NOW*	5	12	versicolor (0.42, 0.50, 0.08)	*
5	°--Sepal.Length > 5.4		1	3	NA	(NA)

Tree2. Accuracy: 51.78%.

levelName	Decision	Number	Count	Class	Probability	Leaf
1	Root		1	150	setosa (0.33, 0.33, 0.33)	NA
2	--Sepal.Length <= 5.4		1	2	52	setosa (0.87, 0.12, 0.02)
3	°--Sepal.Length > 5.4		1	3	NA	(NA) NA

Tree3. Accuracy: 51.78%.

levelName	Decision	Number	Count	Class	Probability	Leaf
1	Root		1	150	setosa (0.33, 0.33, 0.33)	NA
2	--Sepal.Length <= 5.4		1	2	52	setosa (0.87, 0.12, 0.02)
3	°--Sepal.Length > 5.4		1	3	NA	(NA) NA

(b)

Current Tree. Accuracy: 62.44%.

levelName	Decision	Number	Count	Class	Probability	Leaf
1	Root		1	150	setosa (0.33, 0.33, 0.33)	
2	--Sepal.Length <= 5.4		1	2	52	setosa (0.87, 0.12, 0.02)
3	--Petal.Length <= 1.6		2	4	40	setosa (1.00, 0.00, 0.00)
4	°--Petal.Length > 1.6		2	5	12	versicolor (0.42, 0.50, 0.08)
5	°--Sepal.Length > 5.4		1	3	98	virginica (0.05, 0.45, 0.50)
6	--Sepal.Width <= 3.3		3	6	87	virginica (0.00, 0.49, 0.51)
7	°--Sepal.Width > 3.3		3	7	NA	(NA)

Tree1. Accuracy: 87.11%.

levelName	Decision	Number	Count	Class	Probability	Leaf
1	Root		1	150	setosa (0.33, 0.33, 0.33)	
2	--Sepal.Length <= 5.4		1	2	52	setosa (0.87, 0.12, 0.02)
3	--Petal.Length <= 1.6		2	4	40	setosa (1.00, 0.00, 0.00)
4	°--Petal.Length > 1.6		2	5	12	versicolor (0.42, 0.50, 0.08)
5	°--Sepal.Length > 5.4		1	3	98	virginica (0.05, 0.45, 0.50)
6	--Sepal.Width <= 3.3		3	6	87	virginica (0.00, 0.49, 0.51)
7	--Petal.Length <= 4.7	*NOW*	12	37	versicolor (0.00, 1.00, 0.00)	*
8	°--Petal.Length > 4.7	*NOW*	13	50	virginica (0.00, 0.12, 0.88)	*
9	°--Sepal.Width > 3.3		3	7	NA	(NA)

Tree2. Accuracy: 87.78%.

levelName	Decision	Number	Count	Class	Probability	Leaf
1	Root		1	150	setosa (0.33, 0.33, 0.33)	
2	--Sepal.Length <= 5.4		1	2	52	setosa (0.87, 0.12, 0.02)
3	--Petal.Length <= 1.6		2	4	40	setosa (1.00, 0.00, 0.00)
4	°--Petal.Length > 1.6		2	5	12	versicolor (0.42, 0.50, 0.08)
5	°--Sepal.Length > 5.4		1	3	98	virginica (0.05, 0.45, 0.50)
6	--Sepal.Width <= 3.3		3	6	87	virginica (0.00, 0.49, 0.51)
7	--Petal.Width <= 1.7	*NOW*	12	46	versicolor (0.00, 0.91, 0.09)	*
8	°--Petal.Width > 1.7	*NOW*	13	41	virginica (0.00, 0.02, 0.98)	*
9	°--Sepal.Width > 3.3		3	7	NA	(NA)

Tree3. Accuracy: 85.78%.

levelName	Decision	Number	Count	Class	Probability	Leaf
1	Root		1	150	setosa (0.33, 0.33, 0.33)	
2	--Sepal.Length <= 5.4		1	2	52	setosa (0.87, 0.12, 0.02)
3	--Petal.Length <= 1.6		2	4	40	setosa (1.00, 0.00, 0.00)
4	°--Petal.Length > 1.6		2	5	12	versicolor (0.42, 0.50, 0.08)
5	°--Sepal.Length > 5.4		1	3	98	virginica (0.05, 0.45, 0.50)
6	--Sepal.Width <= 3.3		3	6	87	virginica (0.00, 0.49, 0.51)
7	--Petal.Length <= 5.7	*NOW*	12	17	versicolor (0.00, 0.88, 0.12)	*
8	°--Sepal.Length > 5.7	*NOW*	13	70	virginica (0.00, 0.40, 0.60)	*
9	--Petal.Length <= 4.7		26	22	versicolor (0.00, 1.00, 0.00)	*
10	°--Petal.Length > 4.7		27	48	virginica (0.00, 0.12, 0.88)	*
11	°--Sepal.Width > 3.3		3	7	NA	(NA)

(d)

levelName	Decision	Number	Count	Class	Probability	Leaf
1	Root		1	150	setosa (0.33, 0.33, 0.33)	
2	--Sepal.Length <= 5.4		1	2	52	setosa (0.87, 0.12, 0.02)
3	--Petal.Length <= 1.6		2	4	40	setosa (1.00, 0.00, 0.00)
4	°--Petal.Length > 1.6		2	5	12	versicolor (0.42, 0.50, 0.08)
5	°--Sepal.Length > 5.4		1	3	98	virginica (0.05, 0.45, 0.50)
6	--Sepal.Width <= 3.3		3	6	87	virginica (0.00, 0.49, 0.51)
7	--Petal.Width <= 1.7		4	12	46	versicolor (0.00, 0.91, 0.09)
8	°--Petal.Width > 1.7		4	13	41	virginica (0.00, 0.02, 0.98)
9	°--Sepal.Width > 3.3		3	7	11	setosa (0.45, 0.09, 0.45)

(e)

Figure 3. (a–e) Exemplary steps for interactive tree learning from scratch using the Iris data set.

In Figure 3a, we see the tree structure at this point, i.e., the root without any divisions. Since all classes had the same frequencies, the accuracy was 33.33%. In all cases, the first column called “levelName” shows the entire tree structure. Column “Decision” provides information about where the expert has to make a decision using the following label “*NOW*”. Once the decision is made, this label is replaced with the number indicating when the user made a decision. “Number” presents the node number, assuming that the full tree with all nodes was built. “Count” provides the number of observations in a given node. The “Class” column presents the predicted class based on the “Probability” vector presented on the right. Values in this vector are always sorted alphabetically based on the class labels; here we have “setosa”, “versicolor” and “virginica”. The last column indicates whether a particular node is considered a leaf (star) or an intermediate node (empty). As we can see, the first two trees provided an accuracy of 96%, while the last one produced an accuracy of 88.67%. Since we used *varLev* set to “TRUE”, we see three different attributes splitting the root (the best possible split for each variable). We chose the last tree by inputting the number 3 into the function.

Then, we moved to step two presented in Figure 3b. The previous decision was labeled with the number 1. At this point, the accuracy of the current tree was 51.78%. The reason was that the algorithm works in recursive manner, meaning that at this point, we expanded the left part of the tree from node number 2. The sibling of this node (number 3) was “under construction”, waiting for its turn and when calculating the overall accuracy, it was assumed that for this sibling node, we took the majority class. Based on the current structure, it was possible to expand the tree only in the case of the first possible split. The remaining two possible splits caused tree shrinkage because otherwise, all descendants would choose the same label as the predicted class (please see Section 3.2). Therefore, the chosen split was the first structure.

After moving to the right side of the structure (third node) and updating the information for the node, the tree being generated had an accuracy of 63.33%, which can be seen in Figure 3c. First, two possible splits generated only one division, while the third one split the considered node and then later, the left child was also partitioned using the “Petal.Length” attribute. Interestingly, the local best partition (splits are sorted by gain ratio) did not indicate the best global accuracy. The first tree had the best gain ratio with a global accuracy of 88.67%; however, the second-best local split in terms of the gain ratio had a bigger overall accuracy of 89.33%. Let us assume that this time, the user selected the third split.

During the fourth step, the considered node was node number 6 (Figure 3d). It could use the same variable as previously seen in Figure 3c. Since this was the best split in terms of the gain ratio, it was displayed first, and at the same time, as was stated previously, it can be seen that after the decision, the entire structure starting from this node was built automatically. From these three possible splits, we selected the second one, which provided the best overall accuracy of 87.78%.

After the decision, it turned out that interactive learning was over, leading to the structure presented in Figure 3e. After updating the information regarding node number 7, the overall accuracy of the final tree was 88.67%. The final tree structure had three intermediate and five terminal nodes. Based on this structure, the *ExtractRules()* function generated decision rules (in the form of a table), along with the corresponding measures, which are presented in Figure 4. Each rule was of the form “conjunction of some conditions → class label”. Next, on the right, there are accuracy measures, such as: length of the rule, Support, Confidence or Lift.

	Rule	Length	Support	Confidence	Lift	Conviction	AddedValue	Cosine	Jaccard	Laplace	Leverage
1	Sepal.Length <= 5.4 & Petal.Length <= 1.6 -> setosa	2	0.26666667	1.0000000	3.0000000	Inf	0.66666667	0.8944272	0.80000000	0.9534884	0.177777778
2	Sepal.Length <= 5.4 & Petal.Length > 1.6 -> versicolor	2	0.04000000	0.5000000	1.5000000	1.3333333	0.16666667	0.2449490	0.10714286	0.46666667	0.013333333
3	Sepal.Length > 5.4 & Sepal.Width <= 3.3 & Petal.Width <= 1.7 -> versicolor	3	0.28000000	0.9130435	2.739130	7.6666667	0.5797101	0.8757605	0.77777778	0.8775510	0.177777778
4	Sepal.Length > 5.4 & Sepal.Width <= 3.3 & Petal.Width > 1.7 -> virginica	3	0.26666667	0.9756098	2.926829	27.333333	0.6422764	0.8834522	0.78431373	0.9318182	0.175555556
5	Sepal.Length > 5.4 & Sepal.Width > 3.3 -> setosa	2	0.03333333	0.4545455	1.363636	1.222222	0.1212121	0.2132007	0.08928571	0.4285714	0.008888889

Figure 4. Decision rules generated from the final tree structure built from scratch.

We then moved to an example incorporating ambiguity to the learning, which was defined by the difference in the probabilities between the highest class probability and the second-highest class probability per node, below which the expert had to make a decision regarding the subsequent tree structure. We set up this threshold at 0.40 (the *ambProb* parameter in console mode or “Ambiguous probability” in Shiny). The remaining parameters stayed the same. After starting the learning, the first step generated the results presented in Figure 5a.

Current Tree. Accuracy: 33.33%.					
levelName	Decision	Number	Count	Class	Probability Leaf
1 Root		1	150	setosa (0.33, 0.33, 0.33)	NA
Tree1. Accuracy: 96.00%.					
levelName	Decision	Number	Count	Class	Probability Leaf
1 Root		1	150	setosa (0.33, 0.33, 0.33)	
2 --Petal.Length <= 1.9	*NOW*	2	50	setosa (1.00, 0.00, 0.00)	*
3 *--Petal.Length > 1.9	*NOW*	3	100	versicolor (0.00, 0.50, 0.50)	
4 --Petal.Width <= 1.7		6	54	versicolor (0.00, 0.91, 0.09)	*
5 *--Petal.Width > 1.7		7	46	virginica (0.00, 0.02, 0.98)	*
Tree2. Accuracy: 96.00%.					
levelName	Decision	Number	Count	Class	Probability Leaf
1 Root		1	150	setosa (0.33, 0.33, 0.33)	
2 --Petal.Width <= 0.6	*NOW*	2	50	setosa (1.00, 0.00, 0.00)	*
3 *--Petal.Width > 0.6	*NOW*	3	100	versicolor (0.00, 0.50, 0.50)	
4 --Petal.Width <= 1.7		6	54	versicolor (0.00, 0.91, 0.09)	*
5 *--Petal.Width > 1.7		7	46	virginica (0.00, 0.02, 0.98)	*
Tree3. Accuracy: 88.67%.					
levelName	Decision	Number	Count	Class	Probability Leaf
1 Root		1	150	setosa (0.33, 0.33, 0.33)	
2 --Sepal.Length <= 5.4	*NOW*	2	52	setosa (0.87, 0.12, 0.02)	
3 --Petal.Length <= 1.6		4	40	setosa (1.00, 0.00, 0.00)	*
4 *--Petal.Length > 1.6		5	12	versicolor (0.42, 0.50, 0.08)	*
5 *--Sepal.Length > 5.4	*NOW*	3	98	virginica (0.05, 0.45, 0.50)	
6 --Petal.Length <= 4.7		6	43	versicolor (0.12, 0.88, 0.00)	*
7 *--Petal.Length > 4.7		7	55	virginica (0.00, 0.11, 0.89)	*

(a)

(b)

Figure 5. (a,b) Exemplary steps for interactive tree learning using the first type of ambiguity for the Iris data set.

As we can see, each partition generated an ambiguous situation. The first possible split generated nodes with the following probability vectors: (1.00, 0.00, 0.00) and (0.00, 0.50, 0.50). Theoretically, in the first node, there was no ambiguity since the difference in the top two probabilities was $1.00 - 0.00 = 1.00$, which is above the threshold. However, in the second node, the difference was $0.50 - 0.50 = 0.00$, which is below the threshold. A similar situation was observed for other possible splits. After choosing, similar to the previous example, in the third partition, it appears that the learning had finished. First, the algorithm tried to split node number 2, which produced node numbers 4 and 5. Node number 4 was automatically converted into a leaf (since it was pure), while node number 5 could not be divided due to the minimal number of observations being set at five (this node had seven instances). On the right part of the tree, the algorithm tried to split node number 3. This time, the possible splits did not generate an ambiguous situation since the probability differences for the best split were $0.88 - 0.12 = 0.76$ (sixth node) and $0.89 - 0.11 = 0.78$ (seventh node). The algorithm stopped at these levels because possible nodes coming from these two nodes would choose the same class label, i.e., “versicolor” for node number 6 and “virginica” for node number 7.

Finally, we considered the last type of ambiguity. Instead of the *ambProb* parameter, we used the *ambClass* parameter to indicate which class labels were of importance. We selected “versicolor” and “virginica” classes with the corresponding frequencies of 0.30 and 0.10 (*ambClassFreq* parameter). The remaining parameters stayed the same. Since both classes appeared in the root, the first step was the same as in the previous two simulations; see Figure 6a. After choosing the third possible split, the user had partially built the structures to consider, as presented in Figure 6b. As we can see, the left part of each tree was finished (nodes 2, 4 and 5). The reason was that in this part of the tree, there were 52 observations in total, where 45 instances came from the class “setosa” ($52 * 0.87 \approx 45$), 6 instances

came from the class “versicolor” ($52 * 0.12 \approx 6$) and 1 observation came from the class “virginica” ($52 * 0.02 \approx 1$). For the last two classes, we set the thresholds in such a manner that the decision would be made only if there were more than $50 * 0.30 = 15$ (“versicolor”) or $50 * 0.10 = 5$ (“virginica”) instances in the given node. These conditions were not met. Starting from node number 3, it was assumed that the user would have to make a few decisions since the majority of instances from the considered classes fell into this part of the tree $98 * 0.45 \approx 44$ (“versicolor”) or $98 * 0.50 = 49$ (“virginica”). From these three possible splits, we chose the second one with the conditions “Petal.Width ≤ 1.7 ” and “Petal.Width > 1.7 ”.

Figure 6. (a–c) Exemplary steps for interactive tree learning using the second type of ambiguity for the Iris data set.

Likewise to the first ambiguity type, the algorithm stopped at those levels because possible nodes coming from these two nodes would provide the user partitions with the same labels; see Figure 6c. The obtained structure had an accuracy of 89.33%. For comparison purposes, in Figure 7, we present the tree structure that was built in a fully automated manner, achieving an accuracy of 96.0%.

	levelName	Decision	Number	Count	Class	Probability Leaf
1	Root		1	150	setosa (0.33, 0.33, 0.33)	
2	--Petal.Length <= 1.9		2	50	setosa (1.00, 0.00, 0.00)	*
3	*--Petal.Length > 1.9		3	100	versicolor (0.00, 0.50, 0.50)	
4	--Petal.Width <= 1.7		6	54	versicolor (0.00, 0.91, 0.09)	*
5	--Petal.Width > 1.7		7	46	virginica (0.00, 0.02, 0.98)	*

Figure 7. Final tree structure using fully automated learning for the Iris data set.

As presented in Figure 7, predicting Setosa is very easy; therefore, we proposed the following adjustment using a misclassification cost matrix: versicolor had a cost of 5 for virginica, while virginica had a cost of 10 for versicolor. As a result, the overall accuracy improved to 95.33% and the AUC reached 96.54%. The tree structure that was the result of interactive learning with the cost matrix is presented in Figure 8.

	levelName	Decision	Number	Count	Class	Probability	Leaf
1	Root		1	150	virginica	(0.11, 0.32, 0.58)	
2		!--Petal.Length <= 1.9	2	50	setosa	(1.00, 0.00, 0.00)	*
3		*--Petal.Length > 1.9	3	100	virginica	(0.00, 0.35, 0.65)	
4		!--Petal.Length <= 4.7	6	45	versicolor	(0.00, 0.96, 0.04)	*
5		*--Petal.Length > 4.7	7	55	virginica	(0.00, 0.06, 0.94)	*

Figure 8. Final tree structure using fully automated learning and a misclassification cost matrix for the Iris data set.

5.2. Comparison with Benchmarking Algorithms

In this section, we discuss the results of a performance comparison between the obtained trees from Section 5.1 and the following benchmarking algorithms implemented in R software:

- *Rpart*—package for recursive partitioning for classification, regression and survival trees;
- *C50*—package that contains an interface to the C5.0 classification trees and rule-based models based on Shannon entropy;
- *Ctree*—conditional inference trees in the *party* package.

The results are presented in Table 1. It shows the performance of the following trees:

- o Three interactive trees trained with *ImbTreeEntropy*;
- o *ImbTreeAUC* and *ImbTreeEntropy* trees trained in a fully automated manner (best results after the tuning using a grid search of the possible hyper-parameters) using a 10-fold cross-validation;
- o Three benchmarks, i.e., *C50*, *Ctree* and *Rpart* algorithms, that were trained in a fully automated manner (best results after the tuning using grid search of the possible hyper-parameters) using a 10-fold cross-validation.

Table 1. Comparison between authors' algorithms, including interactive learning, and other R packages for classification.

Algorithm	Accuracy Train	AUC Train	Accuracy Valid	AUC Valid	Number of Leaves
Interactive <i>ImbTreeEntropy</i> (Figure 3e)	0.887	0.960	-	-	5.0
Interactive <i>ImbTreeEntropy</i> (Figure 5b)	0.887	0.938	-	-	5.0
Interactive <i>ImbTreeEntropy</i> (Figure 6c)	0.893	0.938	-	-	4.0
<i>ImbTreeAUC</i>	0.976 (∓ 0.006)	0.987 (∓ 0.006)	0.960 (∓ 0.047)	0.968 (∓ 0.037)	3.9 (∓ 0.316)
<i>ImbTreeEntropy</i>	0.961 (∓ 0.005)	0.971 (∓ 0.004)	0.947 (∓ 0.053)	0.959 (∓ 0.041)	3.0 (∓ 0.000)
<i>C50</i>	0.976 (∓ 0.008)	0.990 (∓ 0.003)	0.953 (∓ 0.055)	0.971 (∓ 0.045)	4.1 (∓ 0.316)
<i>Ctree</i>	0.976 (∓ 0.005)	0.989 (∓ 0.003)	0.960 (∓ 0.034)	0.971 (∓ 0.031)	3.3 (∓ 0.483)
<i>Rpart</i>	0.973 (∓ 0.007)	0.986 (∓ 0.008)	0.953 (∓ 0.032)	0.961 (∓ 0.028)	4.0 (∓ 0.000)

To measure the performance, we used accuracy, AUC and the number of leaves in the tree. For the interactive trees, the accuracy and AUC on the validation sample were not provided as the whole dataset was used for the training, which reflects a typical situation where an expert uses the complete dataset to discover the dependencies in the data. The results in Table 1 show that the interactive trees became less accurate; however, we believe

that those tree structures can meet expectations for interpretability as fit for the purpose because they were being prepared by an expert in the field.

6. Conclusions

In this paper, we presented two R packages that allow a field expert to integrate the automatic and interactive construction of a decision tree, along with visualization functionalities (visualize and analyze several tree structures simultaneously). *ImbTreeEntropy* and *ImbTreeAUC* provide algorithms to construct a decision tree from scratch or make a decision regarding the optimal split in ambiguous situations defined in two ways, i.e., the difference between top two class probabilities or class frequencies per node, above which the expert will make a decision. After the learning is finished, the final tree structure might be represented as a rule-based model, along with various quality measures. It should be noted that, in conjunction with the functionalities presented in this paper, the user can employ cost-sensitive learning by defining a misclassification cost matrix and weight-sensitive learning, as well as optimization of the thresholds, where posterior probabilities determine the final class labels such in a way that misclassification costs are minimized [4].

Through the numerical simulations, we showed that interactive learning produced good tree structures with few errors compared to the standard automatic algorithm. By adjusting the constructed trees, the user could intelligently reduce or expand the tree size based on their domain knowledge. Although the trees became less accurate, they also became more readable and understandable to users.

Future research will be focused on extending the algorithm to allow for non-binary splits for both nominal and numerical attributes, where for the latter one, we would like to add discretization functionalities. Finally, we would like to provide user functionality that allows for shrinking or expanding an external tree structure (after uploading the file) that was constructed in advance using either automated or interactive learning.

Author Contributions: K.G. prepared the simulation and analysis and wrote Sections 1–6 of the manuscript; T.Z. wrote Sections 1 and 6 of the manuscript. All authors read and approved the final manuscript. All authors have read and agreed to the published version of the manuscript.

Funding: This research received no external funding.

Conflicts of Interest: The authors declare no conflict of interest.

References

1. Breiman, L.; Friedman, J.H.; Olshen, R.A.; Stone, C.J. *Classification and Regression Trees*; Wadsworth Statistics/Probability Series; CRC Press: Boca Raton, FL, USA, 1984. [\[CrossRef\]](#)
2. Quinlan, J.R. Induction of decision trees. *Mach. Learn.* **1986**, *1*, 81–106. [\[CrossRef\]](#)
3. Kass, G.V. An Exploratory Technique for Investigating Large Quantities of Categorical Data. *Appl. Stat.* **1980**, *29*, 119. [\[CrossRef\]](#)
4. Gajowniczek, K.; Zabkowski, T. ImbTreeEntropy and ImbTreeAUC: Novel R packages for decision tree learning on the imbalanced datasets. *Electronics* **2021**, *10*, 657. [\[CrossRef\]](#)
5. Ankerst, M.; Ester, M.; Kriegel, H.-P. Towards an effective cooperation of the user and the computer for classification. In *KDD'00: Proceedings of the Sixth ACM SIGKDD International Conference on Knowledge Discovery and Data Mining, Boston, MA, USA, 20–23 August 2000*; ACM: New York, NY, USA, 2000; pp. 179–188. [\[CrossRef\]](#)
6. Liu, Y.; Salvendy, G. Design and evaluation of visualization support to facilitate decision trees classification. *Int. J. Hum-Comput. Stud.* **2007**, *65*, 95–110. [\[CrossRef\]](#)
7. Van den Elzen, S.; van Wijk, J.J. BaobabView: Interactive construction and analysis of decision trees. In *Proceedings of the 2011 IEEE Conference on Visual Analytics Science and Technology (VAST), Providence, RI, USA, 23–28 October 2011*; pp. 151–160. [\[CrossRef\]](#)
8. Pauwels, S.; Moens, S.; Goethals, B. Interactive and manual construction of classification trees. *BENELEARN* **2014**, *2014*, 81.
9. Poulet, F.; Do, T.-N. Interactive Decision Tree Construction for Interval and Taxonomical Data. In *Visual Data Mining. Lecture Notes in Computer Science 2008*; Simoff, S.J., Böhlen, M.H., Mazeika, A., Eds.; Springer: Berlin/Heidelberg, Germany, 2008; Volume 4404. [\[CrossRef\]](#)
10. Tan, P.-N.; Kumar, V.; Srivastava, J. Selecting the right objective measure for association analysis. *Inf. Syst.* **2004**, *29*, 293–313. [\[CrossRef\]](#)
11. Geng, L.; Hamilton, H.J. Interestingness measures for data mining. *ACM Comput. Surv.* **2006**, *38*, 9. [\[CrossRef\]](#)

12. Gajowniczek, K.; Liang, Y.; Friedman, T.; Ząbkowski, T.; Broeck, G.V.D. Semantic and Generalized Entropy Loss Functions for Semi-Supervised Deep Learning. *Entropy* **2020**, *22*, 334. [CrossRef] [PubMed]
13. Gajowniczek, K.; Ząbkowski, T. Generalized Entropy Loss Function in Neural Network: Variable's Importance and Sensitivity Analysis. In *Proceedings of the 21st EANN (Engineering Applications of Neural Networks)*; Iliadis, L., Angelov, P.P., Jayne, C., Pimenidis, E., Eds.; Springer: Cham, Germany, 2020; pp. 535–545. [CrossRef]
14. Demsar, J.; Curk, T.; Erjavec, A.; Gorup, C.; Hocevar, T.; Milutinovic, M.; Mozina, M.; Polajnar, M.; Toplak, M.; Staric, A.; et al. Orange: Data Mining Toolbox in Python. *J. Mach. Learn. Res.* **2013**, *14*, 2349–2353.
15. SAS 9.4. Available online: <https://documentation.sas.com/?docsetId=emref&docsetTarget=n1gvjknzxd2a2n12f7co56t1vsu.htm&docsetVersion=14.3&locale=en> (accessed on 3 March 2021).
16. Ankerst, M.; Elsen, C.; Ester, M.; Kriegel, H.-P. Visual classification. In *KDD 99: Proceedings of the Fifth ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*; ACM: New York, NY, USA, 1999; pp. 392–396. [CrossRef]
17. Ankerst, M.; Keim, D.A.; Kriegel, H.-P. Circle segments: A technique for visually exploring large multidimensional data sets. In *Proceedings of the Visualization, Hot Topic Session, San Francisco, CA, USA, 27 October–1 November 1996*.
18. Liu, Y.; Salvendy, G. Interactive Visual Decision Tree Classification. *Human-Computer Interaction*. In *Interaction Platforms and Techniques*; Springer: Berlin/Heidelberg, Germany, 2007; pp. 92–105. [CrossRef]
19. Han, J.; Cercone, N. Interactive construction of decision trees. In *Proceedings of the 5th Pacific-Asia Conference on Knowledge Discovery and Data Mining, Hong Kong, China, 16–18 April 2001*; Springer: London, UK, 2001; pp. 575–580. [CrossRef]
20. Teoh, S.T.; Ma, K.-L. StarClass: Interactive Visual Classification Using Star Coordinates. In *Proceedings of the 2003 SIAM International Conference on Data Mining, San Francisco, CA, USA, 1–3 May 2003*; pp. 178–185. [CrossRef]
21. Teoh, S.T.; Ma, K.-L. PaintingClass. In *KDD'03: Proceedings of the Ninth ACM SIGKDD International Conference on Knowledge Discovery and Data Mining, Washington, DC, USA, 24–27 August 2003*; ACM: New York, NY, USA, 2003; pp. 667–672. [CrossRef]
22. Liu, D.; Sprague, A.P.; Gray, J.G. PolyCluster: An interactive visualization approach to construct classification rules. In *Proceedings of the International Conference on Machine Learning and Applications, Louisville, KY, USA, 16–18 December 2004*; pp. 280–287. [CrossRef]
23. Ware, M.; Frank, E.; Holmes, G.; Hall, M.; Witten, I.H. Interactive machine learning: Letting users build classifiers. *Int. J. Hum.-Comput. Stud.* **2001**, *55*, 281–292. [CrossRef]
24. Poulet, F.; Recherche, E. Cooperation between automatic algorithms, interactive algorithms and visualization tools for visual data mining. In *Proceedings of the Visual Data Mining ECML, the 2nd International Workshop on Visual Data Mining, Helsinki, Finland, 19–23 August 2002*.
25. Do, T.-N. *Towards Simple, Easy to Understand, and Interactive Decision Tree Algorithm*; Technical Report; College of Information Technology, Cantho University: Can Tho, Vietnam, 2007.
26. Goethals, B.; Moens, S.; Vreeken, J. MIME. In *Proceedings of the 17th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining—KDD'11, San Diego, CA, USA, 21–24 August 2011*; pp. 757–760. [CrossRef]
27. Sheeba, T.; Reshmy, K. Prediction of student learning style using modified decision tree algorithm in e-learning system. In *Proceedings of the 2018 International Conference on Data Science and Information Technology (DSIT'18), Singapore, 20–22 July 2018*; pp. 85–90. [CrossRef]
28. Nikitin, A.; Kaski, S. Decision Rule Elicitation for Domain Adaptation. In *Proceedings of the 26th International Conference on Intelligent User Interfaces (IUI'21), College Station, TX, USA, 14–17 April 2021*; pp. 244–248. [CrossRef]
29. Iris dataset. UCI Machine Learning Repository. Available online: <https://archive.ics.uci.edu/ml/index.php> (accessed on 10 October 2020).