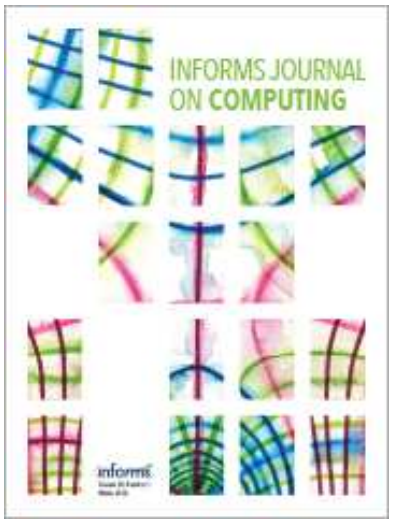


INFORMS Journal on Computing

Publication details, including instructions for authors and subscription information:
http://pubsonline.informs.org

bsnsing: A Decision Tree Induction Method Based on Recursive Optimal Boolean Rule Composition

Yanchao Liu

To cite this article:
Yanchao Liu (2022) bsnsing: A Decision Tree Induction Method Based on Recursive Optimal Boolean Rule Composition. INFORMS Journal on Computing

Published online in Articles in Advance 27 Jul 2022

. https://doi.org/10.1287/ijoc.2022.1225

Published in INFORMS Journal on Computing on July 27, 2022 as DOI: 10.1287/ijoc.2022.1225.
This article has not been copyedited or formatted. The final version may differ from this version.

http://pubsonline.informs.org/journal/ijoc

INFORMS JOURNAL ON COMPUTING
Articles in Advance, pp. 1–22
ISSN 1091-9856 (print), ISSN 1526-5528 (online)

bsnsing: A Decision Tree Induction Method Based on Recursive Optimal Boolean Rule Composition

Yanchao Liu[a]

[a] Department of Industrial and Systems Engineering, Wayne State University, Detroit, Michigan 48202
Contact: yanchaoliu@wayne.edu, https://orcid.org/0000-0002-3256-9336 (YL)

Received: September 8, 2021
Revised: March 5, 2022; May 30, 2022; June 24, 2022
Accepted: July 2, 2022
Published Online in Articles in Advance: July 27, 2022

https://doi.org/10.1287/ijoc.2022.1225

Abstract. This paper proposes a new mixed-integer programming (MIP) formulation to optimize split rule selection in the decision tree induction process and develops an efficient search algorithm that is able to solve practical instances of the MIP model faster than commercial solvers. The formulation is novel for it directly maximizes the Gini reduction, an effective split selection criterion that has never been modeled in a mathematical program for its nonconvexity. The proposed approach differs from other optimal classification tree models in that it does not attempt to optimize the whole tree; therefore, the flexibility of the recursive partitioning scheme is retained, and the optimization model is more amenable. The approach is implemented in an open-source R package named `bsnsing`. Benchmarking experiments on 75 open data sets suggest that `bsnsing` trees are the most capable of discriminating new cases compared with trees trained by other decision tree codes including the rpart, C50, party, and tree packages in R. Compared with other optimal decision tree packages, including DL8.5, OSDT, GOSDT, and indirectly more, `bsnsing` stands out in its training speed, ease of use, and broader applicability without losing in prediction accuracy.

History: Accepted by Ram Ramesh, Area Editor for Data Science & Machine Learning.
Funding: This work was supported by the National Science Foundation Division of Civil, Mechanical and Manufacturing Innovation [Grant 1944068].
Supplemental Material: Data are available at https://doi.org/10.1287/ijoc.2022.1225.

Keywords: classification trees • mixed-integer programming • statistical computing • R

1. Introduction

Classification is the task of assigning objects to one of several predefined categories. A classification tree (or a decision tree classifier) is a predictive model represented in a tree-like structure. Without making excessive assumptions about the data distribution, a classification tree partitions the input space into rectilinear (axis-parallel) regions and ultimately gives a set of *if–then* rules to classify outputs or make predictions. Starting from the root node, each internal node is split into two or more child nodes based on the input values. The split stops when some terminal condition is met. The terminal nodes of the tree are called leaves, which represent the predicted target. Cases move down the tree along branches according to their input values, and all cases reaching a particular leaf are assigned the same predicted value. The tree-like structure connects naturally to the divide-and-conquer strategy of how people judge, plan, and decide. Therefore, the technique is widely adopted for making decisions that bear substantial consequences for the decision maker. Example applications include disease diagnosis (Tanner et al. 2008, Ghiasi et al. 2020), loan approval (Mandala et al. 2012, Alaradi and Hilal 2020), and investment selection (Sorensen et al. 2000).

In this paper, we propose a classification tree induction method based on solving a mixed-integer programming (MIP) model at each split of a node. This new method can generate trees that frequently outperform trees built by other off-the-shelf tree libraries in R, the popular statistical computing system. To achieve this, the proposed MIP model explicitly maximizes the reduction in the node impurity and allows a tree node to be split by a multivariate boolean rule. Such split rules are more flexible and, in certain cases, more efficient at characterizing nonlinear patterns in data, and in the meantime, remain highly interpretable. To conquer the computational challenge, we develop an efficient implicit enumeration algorithm that solves the MIP model faster than the state-of-the-art optimization solvers. Experiments on an extensive collection of machine learning data sets suggest that the method is accurate in prediction performance and scales reasonably well on large training sets. The proposed framework is implemented in an open-source R package named `bsnsing`, available for a broad community of data science researchers and practitioners.

Published in INFORMS Journal on Computing on July 27, 2022 as DOI: 10.1287/ijoc.2022.1225.
This article has not been copyedited or formatted. The final version may differ from this version.

Whereas MIP techniques are used in various ways in the recent literature for building classifiers, our method is unique in several aspects. First, to our knowledge, it is the first MIP model that is able to maximize the Gini reduction of a split, which is known to be an effective split criterion, but, in the meantime, a nonlinear nonconvex function of the split decision. Impurity reduction is an oft-used criterion for split selection in leading decision tree heuristics for its superiority in generating well-balanced child nodes over the accuracy maximization criterion, but it has never been used in an optimization framework because of its nonconvexity. Second, the MIP model is used for rule selection at each node split, whereas other effective elements of the recursive partitioning framework, such as split point generation, early termination, and tree pruning, can be separately implemented with great flexibility. Third, along with the novel formulation, we also develop an efficient exact solution algorithm that runs faster than commercial solver codes, making the `bsnsing` package independent of any commercial optimization solvers.

The remainder of the paper is organized as follows. Section 2 reviews the literature on decision tree induction and particularly the recent literature on optimal classification tree (OCT) developments based on mixed-integer optimization. Section 3 develops the main models and algorithms that underlie the `bsnsing` package. Section 4 presents computational experiments to demonstrate the effectiveness of the proposed method and software tool. Section 5 concludes the paper with pointers for future work.

2. Related Literature

As an extremely flexible nonparametric framework, classification trees delegate a great deal of freedom to algorithm design and implementation (Tan et al. 2005). The entire search space for building the "best" tree can be enormous. Consider, for instance, splitting a node by a categorical variable consisting of 10 distinct levels. There are 115,974 nontrivial ways of splitting, that is, $B_{10} - 1$, the 10th *Bell number* minus one. Moreover, for a set of 10 single-variable split rules, there are more than 3.6 million (i.e., 10!) different ways to order them in a decision list. It is impractical to evaluate all possible splits and all possible ordering of rules. It is shown in Hyafil and Rivest (1976) that the "optimal decision tree" (ODT) problem is NP-complete, and this conclusion is corroborated in many subsequent attempts at constructing optimal decision trees using various optimization modeling techniques.

The difficulty incurred by the enormity of the search space is dealt with along three routes in the literature. The first route is via using greedy splitting methods (Breiman et al. 1984, Quinlan 1993) under the recursive partitioning framework, in which a number of candidate splits are compared and a best one is chosen to split a node. In this general paradigm, there is a great variety of algorithms addressing issues such as how the split variables are selected, how the split points are determined, and how the split quality is assessed, etc. Many efficient decision tree algorithms, including C4.5 (Quinlan 1993), CHAID (Kass 1980), CART (Breiman et al. 1984), GUIDE (Loh 2009), and the recent Bayesian-based approach (Letham et al. 2015, Yang et al. 2017) fall under this paradigm.

The second route is to trim the overall search space down to a reduced model space (as a surrogate) in which global optimization is used to find an optimal model. A popular choice of the surrogate space is the frequent item sets, for example, results from association rule mining algorithms (Agrawal et al. 1993, Liu et al. 1998, Borgelt 2012). Bertsimas et al. (2012) devise a two-step approach in which the first step is to generate an efficient frontier of L candidate item sets by solving L mixed-integer optimization (MIO) problems, one for each candidate item set, and then solve another (larger) MIO problem to rank all the candidates based on their predictive accuracy on all transactions. The top-ranked candidate is chosen as the final classifier. This approach is computationally demanding because of the attempt to build the whole classifier by solving one large MIO problem. Nijssen and Fromont (2010) note the link between decision trees over a binary feature space and the item set lattice and build a recursive tree learning algorithm, which does not invoke a numerical optimization process. Angelino et al. (2017) consider the class of rule lists assembled from premined frequent item sets and search for an optimal rule list that minimizes a regularized risk function, which is able to solve (and prove optimality for) fairly large classification instances. This stream of research is important in constructing the notion of optimality in tree learning and exploring the use of discrete optimization techniques, such as the branch-and-bound algorithm.

The third route is using exhaustive search for comprehensible rules that do not involve too many clauses. The 1R algorithm (Holte 1993) searches exhaustively the space of single-variable rules and then makes a classification or prediction based on the best rule found. Considering its simplicity, it is a surprise that it performs well on many data sets. Nonetheless, the single-variable, single-split strategy apparently sacrifices performance in cases in which more complex and subtle patterns need to be characterized. The EXPLORE algorithm (Rijnbeek and Kors 2010) performs an exhaustive search in the complete rule spaces consisting of one term, two terms, and so forth until the increment of the number of allowed terms stops giving a better performance on the validation set than the previous iteration. By searching

Published in INFORMS Journal on Computing on July 27, 2022 as DOI: 10.1287/ijoc.2022.1225.
This article has not been copyedited or formatted. The final version may differ from this version.

all possible disjunctive normal forms (DNF), the algorithm can find the best DNF rule up to a certain complexity level. An important observation given by this paper is that the phenomenon of over-searching (Quinlan and Cameron-Jones 1995), that is, the hypothesis that the more rules are evaluated, the greater the chance of finding a fluke and poorly generalizable rule, does not always hold. Even though exhaustive search is hardly viable for large cases, this gives an encouraging indication that aggressively seeking optimality on the training set does not necessarily incur overfitting if the sense of optimality is defined on a prudent metric and the tree complexity is properly regulated, an insight that is also presented in Bertsimas and Dunn (2017). In this paper, we develop a more principled exhaustive search procedure to solve the split selection problem.

Using mixed-integer optimization to tackle classification problems has been frequently investigated in recent years, following the seminar work of Bertsimas et al. (2012). Malioutov and Varshney (2013) formulate the rule-based classification problem as an integer program and show that, under certain conditions (among which an excessively large pool of boolean questions needs to be generated from the original feature set), the rules can be recovered exactly by solving the linear programming (LP) relaxation. A probabilistic guarantee of recovery is shown when the required conditions are satisfied weakly. Goh and Rudin (2014) address the problem of class imbalance in classification training and propose an MIP model to find the classification rule set that optimizes a weighted balance between positive and negative class accuracies. They develop a "characterize then discriminate" approach to decompose the problem into manageable subproblems and, hence, alleviate the computational challenge of solving the full MIP. Bertsimas and Dunn (2017) cast the problem that CART attempts to solve as a global optimization problem and instantiate the canonical problem with two MIP models, OCT for building trees using univariate splits, and OCT-H for trees of multivariate splits (separating hyperplanes). The models minimize a weighted sum of the total misclassification cost and the number of splits in the tree and are constrained by two hyperparameters, the tree depth and the leaf node size. The weighting factor in the objective function needs to be tuned via a validation set to achieve the best performance. The robust versions of these models are given in Bertsimas et al. (2019). The strengths of OCT and OCT-H complement each other, and they are able to outperform CART in many cases by significant margins. Wang et al. (2017) focus on searching for a small number of short rules (disjunctive of conjunctives, e.g., "(A and B) or C or …" kind of a rule) by approximately solving a "maximum a posteriori" problem by the simulated annealing algorithm. The authors apply the method to predict user behavior in a recommender system and report favorable performance. Verwer and Zhang (2019) formulate the OCT problem as an integer program in which the number of integer (binary) decision variables does not depend on the number of training data points (though the number of constraints does, and big-M constraints are used). The formulation is demonstrated to outperform previous OCT formulations, including Verwer and Zhang (2017) and Bertsimas and Dunn (2017), on several test data sets.

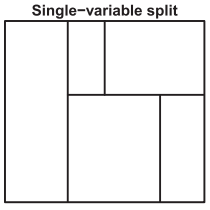
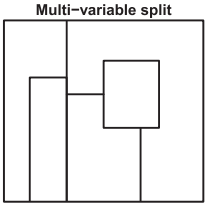
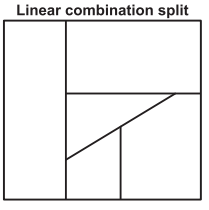
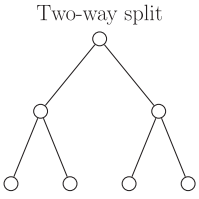
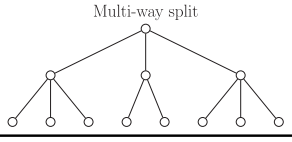
Hu et al. (2019) propose an optimal sparse decision tree (OSDT) algorithm that extends the CORELS algorithm (Angelino et al. 2017) (which creates optimal rule lists) to create optimal trees. The algorithm attempts to minimize the weighted sum of the misclassification error and the number of leaves in the tree. A specialized search procedure within a branch-and-bound framework is employed for solution. A Python program that implements OSDT is available on Hu's Github page. Lin et al. (2020) provide a general framework for decision tree optimization that is able to handle a variety of objective functions and optimize over continuous features. The authors observe orders of magnitude speedup in decision tree construction compared with the state of the art. A C++–based implementation, called generalized and scalable optimal sparse decision trees (GOSDT), as well as a Python wrapper, is available on Lin's Github page. Aglin et al. (2020) develop a DL8.5 algorithm that extends DL8 initially proposed in Nijssen and Fromont (2007). DL8.5 draws upon the association rule mining literature (Agrawal et al. 1993) and uses branch-and-bound search along with a caching mechanism to achieve a fast training speed. It is demonstrated that DL8.5 outruns the BinOCT method by orders of magnitude in training speed. DL8.5 is implemented in C++, and a Python package is publicly available (i.e., pip install dl8.5). In Section 4.2, we perform computational comparisons with DL8.5, OSDT, and GOSDT on a number of binary classification data sets to demonstrate `bsnsing`'s advantage in training speed among these latest developments in the OCT literature. Zhu et al. (2020) propose an MIP model for supervised classification by optimally organizing a support vector machine type of separating hyperplanes in a tree structure of a given depth and achieve outstanding performance on a collection of test sets. An earlier work of this kind can be found in Street (2005), in which the separating hyperplane is obtained via solving a nonlinear conconvex program. Ease of interpretation (or interpretability) of such tree-like models is clearly not an emphasis in those works. Aghaei et al. (2020) present a flow-based MIP formulation for the OCT problem. The formulation does not use big-M constraints, and hence, boasts a stronger LP relaxation than alternative formulations. The authors develop a Bender's decomposition paradigm to further improve

Published in INFORMS Journal on Computing on July 27, 2022 as DOI: 10.1287/ijoc.2022.1225.
This article has not been copyedited or formatted. The final version may differ from this version.

solution efficiency. Substantial speedup in comparison with other OCT approaches is reported.

Existing MIP-based OCT investigations invariably attempt to internalize (i.e., globally optimize) the whole process of building the (tree- or rule-based) classifier and abandon the recursive partitioning framework. Consequently, any posttraining modification, such as pruning, to the optimal tree nullifies optimality in intractable ways, so it is difficult to justify pruning or other tactics aimed at improving prediction performance. Furthermore, making all decisions regarding tree induction in a single MIP model is inevitably challenged by the dilemma that either a lot of simplifications must be imposed to the classifier to limit the size of the search space (hence, introducing high bias) or the decision model ends up being computationally prohibitive. In the author's opinion, solving an MIP usually takes too much time to warrant its inclusion in a meaningful hyperparameter search process that requires training a number of candidate classifiers. For instance, a 10-minute runtime is not uncommon for solving a moderately sized MIP model, but it feels somewhat long for a user of software tools, such as R, SAS, Stata, and IBM SPSS, which are able to produce a tree in no more than a couple of seconds (for reasonably sized data sets, such as those oft used for benchmarking in the literature). As a result, software codes for most OCT approaches, despite their potential appeal in high-stakes applications in which training/tuning time is less of a concern than other merits, such as interpretability, accuracy, and rule set sparsity, etc., are not widely picked up by the broader statistical

Table 1. Comparison of Different Tree Structures

Illustration	Characteristics	Software codes
	• Intuitive decision rules • For example, {Is `Age` ≤ 20?} • Split only produces $(n-1)$-dimensional rectilinear half spaces	ID3, C5.0, CHAID SLIQ, rpart, party
	• Intuitive decision rules • For example, {Is $20 \leq$ `Age` ≤ 25 & $19 \leq$ `BMI` ≤ 24?} • Split can generate closed and open hypercubes of any dimension	CORELS, `bsnsing`
	• Obscure decision rules • For example, {Is $0.3 *$ `Age`$-0.5 *$ `BMI` ≥ 3.3?} • Split can produce half spaces of any dimension	CART, FACT QUEST, CRUISE GUIDE, OCT-H
	• If–then rule clause • Less efficient for multiclass problem	CART, QUEST, SLIQ, rpart, `bsnsing`
	• If–then–else-if rule clause • Suits multiclass problems better • Depletes data too quickly	C5.0, CHAID, FACT CRUISE, QUEST party

Note. The proposed multivariable rectilinear splits are more efficient than single-variable splits in carving out nonlinear features and preserve interpretability better than linear combination splits.

Published in INFORMS Journal on Computing on July 27, 2022 as DOI: 10.1287/ijoc.2022.1225.
This article has not been copyedited or formatted. The final version may differ from this version.

learning or data science community. The `bsnsing` approach attempts to alleviate some of these challenges by taking a middle ground, using MIP to optimize only the split decision, whereas other aspects of tree management, such as when to stop, how to generalize candidate split points, and pruning, etc., are offloaded to the recursive partitioning framework.

Table 1 summarizes the prevalent tree structures available in software tools and how the proposed `bsnsing` package expands the landscape. Existing algorithms under the recursive partitioning framework employ either one-variable splits that are too restrictive for expressing nonlinear patterns or linear combination splits that obscure interpretation. Breiman et al. (1984) briefly entertain the idea of constructing boolean combinations of single-variable splits and recognize the associated difficulty in split search because the combinations are too many to enumerate. To our knowledge, the `bsnsing` package is the first decision tree package to implement boolean combinations of splits in which the combinatorial difficulty is mitigated by an efficient search algorithm.

3. Models and Methods

3.1. Preliminaries and the Framework Overview

Let $\mathcal{X}$ denote the input space containing all possible input vectors x. Suppose that objects characterized by x fall into J classes and let C be the set of classes, that is, $C = \{1, \ldots, J\}$. A classifier is a rule that assigns a class membership in C to every vector in $\mathcal{X}$. In other words, a classifier is a partition of $\mathcal{X}$ into J disjoint subsets $A_1, \ldots, A_j$, $\mathcal{X} = \cup_j A_j$, such that, for every $x \in A_j$, the predicted class is j. A classifier is constructed or trained by a *learning sample* $\mathcal{L}$, which consists of input data on n cases together with their actual class membership, that is, $\mathcal{L} = \{(x_1, j_1), \ldots, (x_n, j_n)\}$, where $x_i \in \mathcal{X}$ and $j_i \in \{1, \ldots, J\}$ for $i = 1, \ldots, n$. At present, we consider the binary classification problem in which $J = 2$ and call the two classes as being *positive* and *negative*, respectively, that is, $C = \{\text{Positive, Negative}\}$.

Given a learning sample $\mathcal{L}$ available at a tree node, the algorithm takes two steps to split the node. First, all input variables are coded into binary features. Each binary feature represents a question that demands a yes/no answer based on the value of the original variable. We call this process *binarization* of the input space. The outcome of this step includes (1) an n-by-m matrix B consisting of 0/1 entries, in which m is the total number of binary features created in the process, and (2) the original binary response vector y in the sample $\mathcal{L}$.

The second step determines a boolean OR clause to split the node. Here, a boolean OR clause refers to a set of general questions joined by the logical OR operator, for example, {Is Age > 35 or Age ≤ 28 or BMI ≥ 30?}. If a case answers yes to any question in the clause, it is classified as a positive case; otherwise, it is classified as a negative case. The selection of questions into the clause is the decision to be made here, which is formulated as a combinatorial optimization problem via mixed-integer programming. Figure 1 demonstrates how `bsnsing` handles these steps.

3.2. Feature Binarization

We very briefly outline the default feature binarization approach implemented in `bsnsing`. It is not the emphasis of this paper, and it is extensible by other developers. For a numeric variable, samples are sorted based on its value. If the two classes are perfectly separable, that is, the minimum value of one class is greater than the maximum value of the other class, the split point is returned, and both child nodes are marked as a leaf. Otherwise, the sorted list of samples is scanned twice in the sequential and reverse order to find "greater-than" and "less-than" types of split conditions, respectively. In this manner, the algorithm implements the subsumption principle as described in Rijnbeek and Kors (2010) to ensure that only potentially valuable split conditions are generated, and ones that are not in the efficient frontier are left out. Other approaches for binarizing numeric features, such as using the empirical quantiles as cut points—see, for example, Malioutov and Varshney (2013)—are also worth implementing in future work. For a categorical variable of L unique levels, the binarization process creates L binary dummy variables when L is below a threshold (default 30). When L is greater than the threshold, value grouping is applied before creating dummy variables. Finally, binary features generated by other decision tree packages can be imported to `bsnsing` for optimal selection as well.

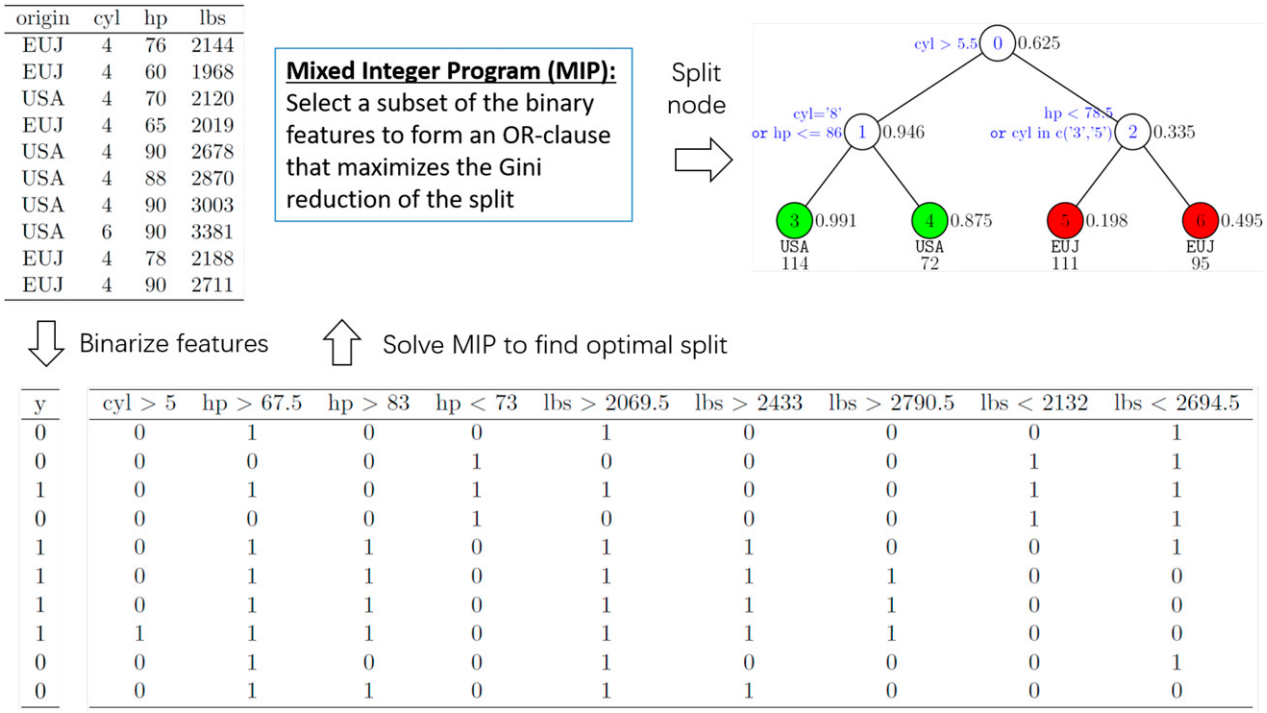
3.3. Mixed-Integer Program to Maximize Gini Reduction

The Gini index, developed by Italian statistician Corrado Gini in 1912, is a measure of variability for categorical data. It can be used to measure the impurity of a decision tree node. In general, the Gini index for a set of objects (e.g., cases contained in a tree node) that come from J possible classes is given by $1 - \sum_{j=1}^{J} p_j^2$, where p_j is the relative frequency of the target class j, $j \in \{1, \ldots, J\}$, in the node. A split that produces a large reduction in Gini (i.e., ΔGini) is preferred. The ΔGini splitting criterion is first proposed in Breiman et al. (1984). It is available in many decision tree codes such as `rpart` and `tree` packages in R and the decision tree method in SAS Enterprise Miner. We formulate the ΔGini criterion as a mixed-integer linear program of the splitting decision. To our knowledge, no prior work has done so.

Let P and N denote the numbers of positive and negative cases, respectively, at the current (parent) node.

Published in INFORMS Journal on Computing on July 27, 2022 as DOI: 10.1287/ijoc.2022.1225.
This article has not been copyedited or formatted. The final version may differ from this version.

Figure 1. (Color online) Process Flow of the bsnsing Method



y	cyl > 5	hp > 67.5	hp > 83	hp < 73	lbs > 2069.5	lbs > 2433	lbs > 2790.5	lbs < 2132	lbs < 2694.5
0	0	1	0	0	1	0	0	0	1
0	0	0	0	1	0	0	0	1	1
1	0	1	0	1	1	0	0	1	1
0	0	0	0	1	0	0	0	1	1
1	0	1	1	0	1	1	0	0	1
1	0	1	1	0	1	1	1	0	0
1	0	1	1	0	1	1	1	0	0
1	1	1	1	0	1	1	1	0	0
0	0	1	0	0	1	0	0	0	1
0	0	1	1	0	1	1	0	0	0

Notes. First, a binary feature matrix B is created based on the original input variables. Each feature represents a general question that demands a yes/no answer. Next, an MIP is solved to select the set of questions to form a boolean OR clause that would maximize impurity reduction. Finally, the tree node is split by the selected rule(s). Detailed annotation of the bsnsing tree plot is given in the appendix.

The Gini index of this node, denoted by $G(\text{parent})$, is

$$G(\text{parent}) = 1 - \left(\frac{P}{P+N}\right)^2 - \left(\frac{N}{P+N}\right)^2 = \frac{2P \cdot N}{(P+N)^2}. \quad (1)$$

Suppose the node is split into two child nodes by a split rule, that is, cases that are ruled to be positive fall in the left node and cases that are ruled to be negative fall in the right node. Let TP and FP denote the numbers of positive and negative cases, respectively, that fall in the left node, and TN and FN denote the numbers of negative and positive cases, respectively, that fall in the right node. Then, the Gini indexes of the left and right nodes are

$$G(\text{left}) = \frac{2 \cdot TP \cdot FP}{(TP+FP)^2} \quad \text{and} \quad G(\text{right}) = \frac{2 \cdot TN \cdot FN}{(TN+FN)^2}.$$

The ΔGini of the split is defined to be the Gini index of the parent node minus the weighted sum of the Gini indexes of the child nodes, whereas the weights are the squared proportions of cases that fall in each child node,

$$\Delta G = G(\text{parent}) - \left(\frac{TP+FP}{P+N}\right)^2 \cdot G(\text{left}) - \left(\frac{TN+FN}{P+N}\right)^2 \cdot G(\text{right}). \quad (2)$$

Note that the squared proportions are used here, instead of the proportions as originally proposed in Breiman et al. (1984), for ease of mathematical modeling. Note that the value of ΔG is not affected by the classification labels assigned to the child nodes. For example, if we were to classify the left node as negative and the right node as positive, it would only cause a swap between TP and FN and a swap between FP and TN in the equations, which would not affect the value of ΔG. Equation (2) can be simplified to $\Delta G = (2P \cdot N - 2(TP \cdot FP + TN \cdot FN))/(P+N)^2$. Because P and N are known values for the parent node irrespective of the split, maximizing ΔG is equivalent to minimizing $TP \cdot FP + TN \cdot FN$, which is, in turn, equivalent to minimizing

$$P \cdot FP + N \cdot FN - 2 \cdot FN \cdot FP. \quad (3)$$

Here, FP and FN are variables whose values depend on the split rule.

Mathematical symbols used in the MIP formulation are defined in Table 2. Let the binary variable w_k indicate whether question k is selected (= 1) or not (= 0); then, the product $B_{ik}w_k$ equals one if both question k is selected and case i answers yes for the question. When the selected questions form an OR clause classification rule, the case i is classified as positive ($z_i = 1$) if there exists a $k \in \mathcal{K}$ such that $B_{ik}w_k = 1$. Conversely,

Published in INFORMS Journal on Computing on July 27, 2022 as DOI: 10.1287/ijoc.2022.1225.
This article has not been copyedited or formatted. The final version may differ from this version.

Table 2. Notation Definition for the Mathematical Program

Symbol	Meaning
$\mathcal{I}$	$= \{1, \ldots, n\}$, index set of cases
$\mathcal{P}, \mathcal{N}$	index sets of positive and negative cases, respectively
$\mathcal{K}$	$= \{1, \ldots, m\}$, index set of questions
B_{ik}	= 1 if case i answers yes for question k; zero otherwise
y_i	= 1 if case i is positive; zero otherwise
w_k	= 1 if question k is selected into the split rule; zero otherwise
z_i	= 1 if case i is classified as positive; zero negative
θ_{ij}	A binary variable for each pair of cases i and j

case i is classified as negative ($z_i = 0$) if $B_{ik}w_k = 0$ for all $k \in \mathcal{K}$. This classification rule is expressed by the following linear constraints:

$$B_{ik}w_k \le z_i, \qquad \forall i \in \mathcal{I},\ \forall k \in \mathcal{K}, \tag{4}$$

$$\sum_{k\in\mathcal{K}} B_{ik}w_k \ge z_i, \qquad \forall i \in \mathcal{I}. \tag{5}$$

We only need to enforce $0 \le z_i \le 1$ for $i \in \mathcal{I}$ in the formulation because the integrality of variable z_i is implied by Constraints (4) and (5), the fact that each w_k is binary, and the sense (i.e., minimization) of the optimization objective.

Using variable z_i, we can express the terms in (3) as follows:

$$P \cdot FP = P \cdot \sum_{j\in\mathcal{I}} (1 - y_j)z_j = \sum_{i\in\mathcal{P}} \sum_{j\in\mathcal{N}} z_j$$

$$N \cdot FN = N \cdot \sum_{i\in\mathcal{I}} y_i(1 - z_i) = \sum_{j\in\mathcal{N}} \sum_{i\in\mathcal{P}} (1 - z_i)$$

$$\begin{aligned} FN \cdot FP &= \left(\sum_{i\in\mathcal{I}} y_i(1 - z_i)\right)\left(\sum_{j\in\mathcal{I}} (1 - y_j)z_j\right) \\ &= \sum_{i,j\in\mathcal{I}} y_i(1 - z_i)(1 - y_j)z_j \\ &= \sum_{i\in\mathcal{P}} \sum_{j\in\mathcal{N}} (1 - z_i)z_j. \end{aligned}$$

Minimizing (3) rewards a greater value of $FN \cdot FP$; therefore, we can replace each term $(1 - z_i)z_j$ in the last equation by a free variable θ_{ij}, defined for $i \in \mathcal{P}$ and $j \in \mathcal{N}$, and impose the following constraints:

$$\theta_{ij} \le 1 - z_i, \qquad \forall i \in \mathcal{P},\ \forall j \in \mathcal{N}, \tag{6}$$

$$\theta_{ij} \le z_j, \qquad \forall i \in \mathcal{P},\ \forall j \in \mathcal{N}. \tag{7}$$

Putting everything together, the problem that optimizes Gini reduction is formulated as a mixed-integer program as follows. Let us call it OPT-G.

(OPT-G):

$$\text{Minimize} \quad \sum_{i\in\mathcal{P}} \sum_{j\in\mathcal{N}} (1 - z_i + z_j - 2\theta_{ij}) \tag{8}$$

$$\text{s. t.} \quad (4),\ (5),\ (6) \text{ and } (7),$$

$$w_k \in \{0, 1\}, \quad \forall k \in \mathcal{K}, \tag{9}$$

$$0 \le z_i \le 1, \quad \forall i \in \mathcal{I}, \tag{10}$$

$$\theta_{ij} \text{ free} \quad \forall i \in \mathcal{P},\ j \in \mathcal{N}. \tag{11}$$

For each pair of cases $i \in \mathcal{P}$ and $j \in \mathcal{N}$, θ_{ij} equals one only when both cases are classified incorrectly. When this happens, the corresponding objective term $(1 - z_i + z_j - 2\theta_{ij})$ is the same value (zero) as when both cases are classified correctly. This corroborates the theory that the ΔG split criterion is agnostic of the polarity of the labels assigned to the child nodes. Equation (10) can be further reduced to $z_i \le 1$ for $i \in \mathcal{P}$ and $z_i \ge 0$ for $i \in \mathcal{N}$. Most MIP models in the OCT literature attempt to minimize misclassification (plus other terms to suppress overfitting). The OPT-G model can be reduced to minimizing misclassification simply by removing the objective terms and constraints that involve θ. Specifically, the following model (OPT-E) explicitly minimizes the number of misclassified cases from the split.

(OPT-E):

$$\text{Minimize} \quad \sum_{i\in\mathcal{P}} \sum_{j\in\mathcal{N}} (1 - z_i + z_j)$$

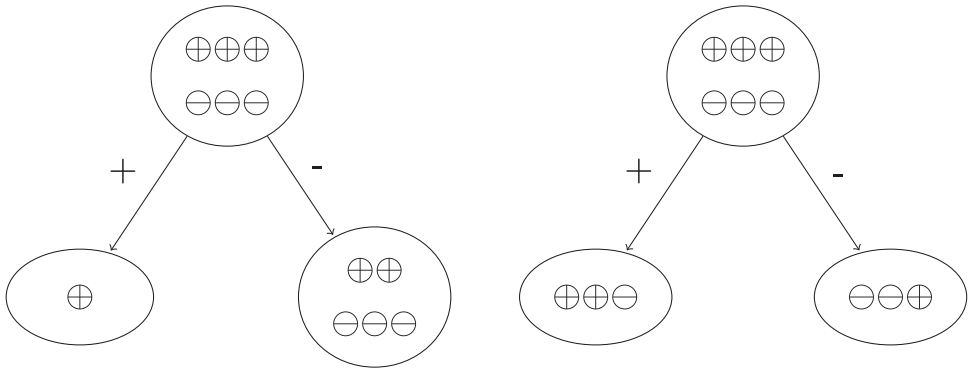
$$\text{s. t.} \quad (4),\ (5),\ (9) \text{ and } (10). \tag{12}$$

The Gini-based splitting criterion is designed to purify the class composition in the child nodes. Compared with error-based criterion, it is conducive to more balanced child nodes. This is demonstrated in the simple example in Figure 2. Therefore, in this paper, we focus on analyzing the OPT-G model and its solution strategy. Splits based on the OPT-E model are available in the `bsnsing` package through the option `opt.model = "error"`. Users can choose Gurobi, CPLEX, and lpSolve to solve the MIP model by setting the `opt.solver` option in `bsnsing` provided that the chosen solver and its R API package are installed.

In the tree-building process, the instance size of the OPT-G model is largest at the root node and decreases exponentially as the process moves down the tree branches. For large training data sets, the root node

Published in INFORMS Journal on Computing on July 27, 2022 as DOI: 10.1287/ijoc.2022.1225.
This article has not been copyedited or formatted. The final version may differ from this version.

Figure 2. Examples of Splitting a Tree Node



Notes. In contrast, OPT-G favors the one leading to more balanced child nodes. The two ways of splitting the parent node give the same number of misclassifications; Hence, they are indistinguishable under OPT-E.

model can become computationally prohibitive for even the commercial solvers, such as CPLEX and Gurobi. In the next section, we propose an implicit enumeration approach for solving OPT-G that scales better than commercial solver codes in practical settings.

3.4. Implicit Enumeration (ENUM) Algorithm for Solving OPT-G

Each candidate solution to OPT-G can be denoted by its corresponding index set $\mathcal{S}$ of the selected questions, that is, $\mathcal{S} := \{k \in \mathcal{K} | w_k = 1\}$. At a solution $\mathcal{S}$, let us denote the false positive and negative counts at the solution by $FP_{\mathcal{S}}$ and $FN_{\mathcal{S}}$, respectively, and define $v(\mathcal{S})$ to be the objective value at the solution, that is,

$$v(\mathcal{S}) := P \cdot FP_{\mathcal{S}} + N \cdot FN_{\mathcal{S}} - 2 \cdot FP_{\mathcal{S}} \cdot FN_{\mathcal{S}}. \tag{13}$$

We know that starting at any $\mathcal{S}$, selecting more questions into the solution (i.e., enlarging $\mathcal{S}$) only encourages extra cases to fall in the left node (i.e., answer "yes" to the OR clause) and, hence, causes FP to either stay at the same value or increase and causes FN to stay at the same value or decrease. In other words, for any $\mathcal{S}_+ \supset \mathcal{S}$, we have $FP_{\mathcal{S}_+} \geq FP_{\mathcal{S}}$ and $FN_{\mathcal{S}_+} \leq FN_{\mathcal{S}}$. Leveraging this property, we can derive a lower bound on the objective value for any possible solution that "branches out" from a given solution $\mathcal{S}$.

Proposition 1. *For any superset of $\mathcal{S}$, denoted by $\mathcal{S}_+$, the following inequalities hold:*

$$v(\mathcal{S}_+) \geq \begin{cases} v(\mathcal{S}), & \text{if } FN_{\mathcal{S}} < P/2 \text{ and } FP_{\mathcal{S}} > N/2 \\ P \cdot FP_{\mathcal{S}}, & \text{if } FN_{\mathcal{S}} < P/2 \text{ and } FP_{\mathcal{S}} \leq N/2 \\ N \cdot (P - FN_{\mathcal{S}}), & \text{if } FN_{\mathcal{S}} \geq P/2 \text{ and } FP_{\mathcal{S}} > N/2 \\ 0, & \text{if } FN_{\mathcal{S}} \geq P/2 \text{ and } FP_{\mathcal{S}} \leq N/2. \end{cases}$$

Proof. In the case $FN_{\mathcal{S}} < P/2$ and $FP_{\mathcal{S}} > N/2$, we have $N - 2 \cdot FP_{\mathcal{S}_+} \leq N - 2 \cdot FP_{\mathcal{S}} < 0$ and $P - 2 \cdot FN_{\mathcal{S}_+} \geq P - 2 \cdot FN_{\mathcal{S}} > 0$; therefore,

$$\begin{aligned} v(\mathcal{S}_+) &= P \cdot FP_{\mathcal{S}_+} + (N - 2 \cdot FP_{\mathcal{S}_+}) \cdot FN_{\mathcal{S}_+} \\ &\geq P \cdot FP_{\mathcal{S}_+} + (N - 2 \cdot FP_{\mathcal{S}_+}) \cdot FN_{\mathcal{S}} \\ &= (P - 2 \cdot FN_{\mathcal{S}_+}) \cdot FP_{\mathcal{S}_+} + N \cdot FN_{\mathcal{S}} \\ &\geq (P - 2 \cdot FN_{\mathcal{S}}) \cdot FP_{\mathcal{S}} + N \cdot FN_{\mathcal{S}} \\ &= v(\mathcal{S}). \end{aligned}$$

In the case $FN_{\mathcal{S}} < P/2$ and $FP_{\mathcal{S}} \leq N/2$, we have $P - 2 \cdot FN_{\mathcal{S}_+} \geq P - 2 \cdot FN_{\mathcal{S}} > 0$ and $N - 2 \cdot FP_{\mathcal{S}} \geq 0$; therefore,

$$\begin{aligned} v(\mathcal{S}_+) &= N \cdot FN_{\mathcal{S}_+} + (P - 2 \cdot FN_{\mathcal{S}_+}) \cdot FP_{\mathcal{S}_+} \\ &\geq N \cdot FN_{\mathcal{S}_+} + (P - 2 \cdot FN_{\mathcal{S}_+}) \cdot FP_{\mathcal{S}} \\ &= (N - 2 \cdot FP_{\mathcal{S}}) \cdot FN_{\mathcal{S}_+} + P \cdot FP_{\mathcal{S}} \\ &\geq P \cdot FP_{\mathcal{S}}. \end{aligned}$$

In the case $FN_{\mathcal{S}} \geq P/2$ and $FP_{\mathcal{S}} > N/2$, we have $N - 2 \cdot FP_{\mathcal{S}_+} \leq N - 2 \cdot FP_{\mathcal{S}} < 0$ and $P - 2 \cdot FN_{\mathcal{S}} < 0$; therefore,

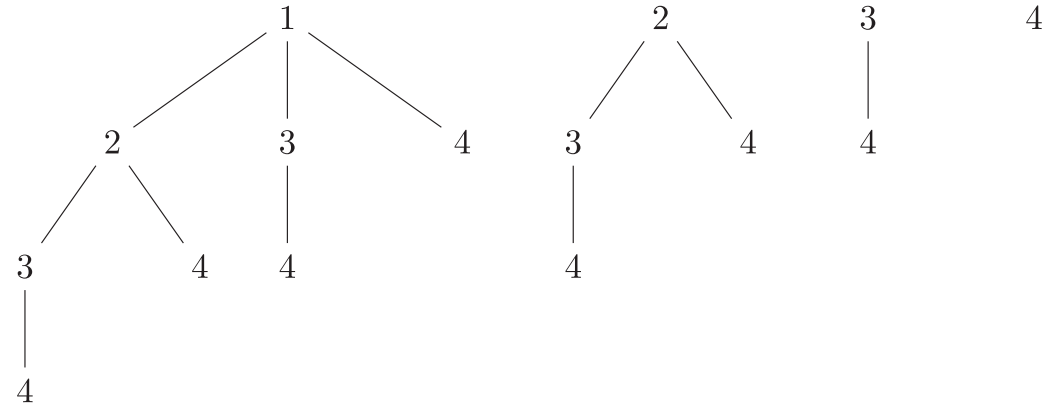
$$\begin{aligned} v(\mathcal{S}_+) &= P \cdot FP_{\mathcal{S}_+} + (N - 2 \cdot FP_{\mathcal{S}_+}) \cdot FN_{\mathcal{S}_+} \\ &\geq P \cdot FP_{\mathcal{S}_+} + (N - 2 \cdot FP_{\mathcal{S}_+}) \cdot FN_{\mathcal{S}} \\ &= (P - 2 \cdot FN_{\mathcal{S}}) \cdot FP_{\mathcal{S}_+} + N \cdot FN_{\mathcal{S}} \\ &\geq (P - 2 \cdot FN_{\mathcal{S}_+}) \cdot N + N \cdot FN_{\mathcal{S}} \\ &= (P - FN_{\mathcal{S}}) \cdot N. \end{aligned}$$

In the last case in which $FN_{\mathcal{S}} \geq P/2$ and $FP_{\mathcal{S}} \leq N/2$, there is insufficient information to derive a nontrivial lower bound for $v(\mathcal{S}_+)$ as $v(\mathcal{S}_+) \geq 0$ always holds. □

Let us denote the lower bound for $v(\mathcal{S}_+)$ by $\tau(\mathcal{S})$ to emphasize its sole dependence on the current solution $\mathcal{S}$ and assign its value according to Proposition 1. In the search for the optimal solution, whenever $\tau(\mathcal{S})$ is greater than the best objective value found so far, any solution that results from enlarging $\mathcal{S}$ can be eliminated. The algorithm is outlined as follows. We start

Published in INFORMS Journal on Computing on July 27, 2022 as DOI: 10.1287/ijoc.2022.1225.
This article has not been copyedited or formatted. The final version may differ from this version.

Figure 3. Enumeration of All Possible Solutions in a Solution Space of Four Candidate Rules Indexed by {1,2,3,4}



Notes. Each node, along with the top-down path leading to it, represents a unique subset of the solution space. Candidate solutions are evaluated in the breadth-first order. Proposition 1 enables opportunities to prune the search tree branches.

with evaluating all single-variable split rules, that is, $\mathcal{S}_k = \{k\}$, for $k \in \mathcal{K}$ by calculating their $v(\mathcal{S}_k)$ and $\tau(\mathcal{S}_k)$. These are the "root nodes" in the search tree.[1] Let v^{best} be the smallest objective values encountered so far. We can eliminate those nodes k having $\tau(\mathcal{S}_k) > v^{\text{best}}$, and we can terminate a node (i.e., making it a leaf in the search tree) when $\tau(\mathcal{S}_k) = v(\mathcal{S}_k)$. For each remaining root node k, we evaluate all two-variable split rules branched from it in such a way that the added variable's index is greater than k (to avoid redundant evaluations) and update the v^{best} and eliminate unpromising branches on the fly. For instance, if $\tau(\{1,2\}) > v^{\text{best}}$, then the candidate solution $\{1,2\} \cup K$ for each $K \subset 2^{\{3,\ldots,m\}}$ can be eliminated. The search proceeds until all possibilities are examined, at which point the v^{best} is the optimal solution to OPT-G. Figure 3 demonstrates the enumeration process via an example. This algorithm is implemented in the `bslearn` function in the R source code under the "enum" solver option, and we call it the ENUM algorithm in the rest of this paper.

The ENUM algorithm, if carried out to completion, can guarantee to return a solution having the smallest objective value. In essence, the sequential evaluation of candidate solutions reduces the search space from $\mathcal{K} \times \mathcal{P} \times \mathcal{N}$ to $2^{\mathcal{K}}$. Because the cost of evaluating a candidate solution ramps up slowly with the size of $\mathcal{I}$ (i.e., $\mathcal{P} \cup \mathcal{N}$) thanks to efficient vector operations, the ENUM can find the optimal solution faster than the typical branch-and-bound method of MIP solvers when $\mathcal{I}$ is large and $\mathcal{K}$ is relatively small.

Compared with using an MIP solver, ENUM boasts the following advantages: (1) it scales betters in terms of memory cost because no branch-and-bound tree is maintained; (2) the search process can be parallelized (though it is a bit tricky to implement in R because R is intrinsically single-threaded); (3) the search is breadth first, meaning that split rules having fewer clauses are evaluated before rules having more clauses get evaluated. Therefore, if any time the search is terminated prematurely (e.g., because of a time limit), a simplest possible best found rule can still be returned; in addition, if the optimal solution is not unique, one having the fewest clauses is returned.

3.5. Complexity Regulation Constraints

When OPT-G is used to split the nodes, the classification tree can be grown until every node is pure (i.e., containing observations of the same class). Such a tree, called a maximal tree, does not generalize well on new data. Additional constraints can be added to OPT-G to regulate the complexity of the tree to curb overfitting. In the `bsnsing` package, we leverage two types of constraints as follows:

$$\sum_{k \in \mathcal{K}} w_k \le \text{MaxRules}, \tag{14}$$

$$\sum_{i \in \mathcal{I}} z_i \ge \text{MinNodeSize}, \tag{15}$$

$$\sum_{i \in \mathcal{I}} (1 - z_i) \ge \text{MinNodeSize}. \tag{16}$$

The right-hand sides of these inequalities are control parameters. Constraint (14) limits the number of questions to enter a split rule. This constraint is quite necessary in practice, especially when the tree is to be used for prediction, because, devoid of this constraint, each split optimization step amounts to a maximal overfitting of the data available in the present node—finding a single composite split rule to maximally reduce the Gini. In addition, this constraint also directly reduces the solution space; thus, its presence expedites the ENUM search. Specifically, the number of objective function evaluations in the worst case (assuming no

Published in INFORMS Journal on Computing on July 27, 2022 as DOI: 10.1287/ijoc.2022.1225.
This article has not been copyedited or formatted. The final version may differ from this version.

pruning and early termination opportunities exist in the search process) reduces to $\sum_{i=1}^{\text{MaxRules}} \binom{m}{i}$ as compared with 2^m when the constraint is not present. For instance, for a pool of $m = 98$ candidate questions, Constraint (14) with MaxRules = 3 reduces the worst case objective function evaluations to $\binom{98}{1}+\binom{98}{2}+\binom{98}{3} = 1,56,947$ from the theoretical worst worst case of $2^{98} = 3.17 \times 10^{29}$. A demonstration is presented in Section 4.

Constraints (15) and (16) require that each child node from a split must contain at least MinNodeSize (a positive number) observations. They are effective at limiting the tree depth as well as generating well-populated leaf nodes. In the `bsnsing` package, the default value for MaxRules is two, and MinNodeSize is, by default, set equal to the square root of n, the number of training examples.

In certain use cases, it is customary to expect the child nodes of a split to bear different majority classes, that is, to require $TP/(TP+FP) \geq 0.5$ and $FN/(FN+TN) \leq 0.5$. This requirement can be translated to the following linear constraint for the OPT-G model:

$$\sum_{i\in\mathcal{P}} z_i - \sum_{j\in\mathcal{N}} z_j \geq \max\{0, 2\cdot P - n\}. \tag{17}$$

To include this constraint in the ENUM algorithm, we can simply discard any candidate solution that violates it in the search process. We comment that this constraint is mainly for the convenience of tree interpretation and leads to an over-regulation (i.e., creating unnecessary bias) to the tree model. Specifically, its inclusion in the model tends to produce simple and shallow trees that lack fidelity in discriminating new cases. Therefore, we choose not to enable it by default in the `bsnsing` package. To enable the constraint, the user can explicitly set the parameter `no.same.gender.children` to true.

Other hyperparameters used in the `bsnsing` function include (1) the bin size (`bin.size`), which specifies the minimum number of observations that must satisfy a candidate binary question for the question to enter the pool; (2) the stop probability (`stop.prob`), a node purity threshold in terms of the proportion of the majority class in the node which, if exceeded, the node is not further split; and (3) the maximum number of segments into which the range of a numeric variable (`nseg.numeric`) is divided by inequalities of the same direction. These parameters do not directly affect the split rule optimization, but they affect the overall efficiency of the tree-building process and the final performance of the tree. Apart from generating candidate binary questions internally, the `bsnsing` function is also able to import split questions generated by other decision tree packages (currently including C50, tree, party, and rpart packages) into its own pool for optimal selection. This option is enabled by the parameter `import.external`. For a complete list of control parameters, users can consult the help document by typing "`?bscontrol`" in R.

4. Evaluation

The models and algorithms developed in this paper are implemented in the open-source R package named `bsnsing`, accessible through the Comprehensive R Archive Network (CRAN). The source code is also hosted at github.com/profyliu/bsnsing.

In this section, we demonstrate the computational efficiency of solving OPT-G using different algorithms, compare the performance of `bsnsing` against several other decision tree packages available in R, and showcase the basic usage of the `bsnsing` library via some code samples. The experiments were performed in RStudio (R version 3.6.2) on a MacBook Pro with Intel Core i9 (8 cores) processor and 16 GB RAM.

4.1. Computational Efficiency of Solving OPT-G

We perform two sets of experiments to demonstrate the superiority of the ENUM algorithm over the Gurobi solver for solving OPT-G. In the first set of experiments, we generate OPT-G instances of different sizes based on the seismic data set from the University of California, Irvine (UCI) machine learning repository (Dua and Graff 2017) and show how each method scales as data size increases; in the second set of experiments, we contrast the solutions by the two methods on more classification data sets adopted from the literature to further solidify our conclusion.

4.1.1. Experiments on the Seismic Data Set.

The seismic data set has 1,690 observations and 18 input variables, among which four are categorical variables and 14 are numeric variables. The output is a categorical variable with two levels, so it is a binary classification problem. We first binarize all inputs to create a binary matrix B (using the `binarize` function in the `bsnsing` package), essentially substituting one or multiple binary features for each of the original input variables. The matrix B has 1,690 rows and 100 columns. We experiment with two scenario factors that are critical to the computational efficiency: the number of training cases n and the maximum number of rules, that is, MaxRules, allowed in the solution. We randomly sample n rows from B and join them with the corresponding response variables to form sets of training data with varying sizes. For each combination of `max.rules` in {1,2,3,4} and n in {200,400,..., 1,600}, we solve the OPT-G

Published in INFORMS Journal on Computing on July 27, 2022 as DOI: 10.1287/ijoc.2022.1225.
This article has not been copyedited or formatted. The final version may differ from this version.

model using both the Gurobi solver and the ENUM algorithm. For example, suppose the selected rows of matrix B are stored in matrix bx and the binary target vector is y; the following R code was used to learn the optimal split rule with MaxRules = 3 using the Gurobi solver:

```
res <- bslearn(bx, y, bscontrol(opt.solver = 'gurobi', solver.timelimit = 7200, max.rules = 3, node.size = 1))
```

Note that the `node.size` option was set to one to relax Constraints (15) and (16), which were irrelevant to this experiment. For the ENUM algorithm, the `opt.solver` option was set to "`enum_c`".[2]

The results are summarized in Table 3. As expected, in each case, both methods are able to find the same optimal objective value, reported in the column Objval. For the same instance, the Objval is nonincreasing with the increase in max.rules, which also matches our expectation.

The next three columns of Table 3 list the computing time in seconds. The Gurobi solver automatically exploited multiple CPU cores available in the computer (which had eight physical cores), so we report both the CPU time (actual computing resource usage) in column Gurobi and the elapsed time (wall time as experienced by the user) in column Gurobi.E. The ENUM algorithm used only one CPU core; thus, only the elapsed time is reported. Clearly, ENUM is the incontestable winner, running at least two orders of magnitude faster than Gurobi in elapsed time.

For the ENUM algorithm, the number of objective function evaluations in the worst case is $\sum_{i=1}^{\text{max.rules}} \binom{100}{i}$. However, the actual number of evaluations is significantly fewer than the theoretical worst case. The actual numbers of objective function evaluations as well as their percentage of the theoretical worst case are listed in columns Obj.Evals and Pct of All Feasible, respectively. For example, when $n = 200$ and `max.rule` = 1, ENUM evaluates 100 candidate solutions to find the optimal one, and this number is 100% of all candidate solutions in the search space. We can see that the percentage drops significantly (hence, substantial savings in computing accrued) with the increase in `max.rules`. The savings are attributed to the bounding and early termination strategy of Proposition 1. Similar experiments on other data sets and other solvers (i.e., CPLEX and lpSolve) reveal the same insight, so we forgo repeated experiments on those scenarios.

Table 3. Computational Costs and Scalability of ENUM

Observations	Max.rules	Objval	Gurobi	Gurobi.E	ENUM	Obj. Evals	Percentage of all feasible
200	1	512	5.6	2.8	0.0	100	100.0
	2	492	14.5	2.9	0.0	2,959	58.6
	3	420	20.4	3.8	0.1	36,998	22.2
	4	378	20.0	3.7	0.5	249,951	6.1
400	1	2,324	14.5	6.8	0.0	100	100.0
	2	2,046	97.7	14.9	0.0	3,163	62.6
	3	1,974	188.7	27.6	0.2	46,270	27.7
	4	1,974	93.3	14.8	2.0	406,135	9.9
600	1	5,044	48.5	19.7	0.0	100	100.0
	2	4,710	261.1	38.6	0.0	3,332	66.0
	3	4,610	421.3	59.2	0.3	49,059	29.4
	4	4,610	359.4	50.5	3.8	445,965	10.9
800	1	8,494	117.6	43.9	0.0	100	100.0
	2	8,153	796.7	109.2	0.1	3,308	65.5
	3	7,933	1,447.3	191.8	0.6	51,389	30.8
	4	7,802	787.1	108.2	5.4	463,565	11.3
1,000	1	12,410	58.5	34.2	0.0	100	100.0
	2	11,946	1,359.8	188.2	0.1	3,241	64.2
	3	11,652	1,727.2	233.7	0.7	50,300	30.2
	4	11,382	2,498.2	327.0	6.6	442,363	10.8
1,200	1	17,027	842.7	433.8	0.0	100	100.0
	2	15,840	2,370.9	316.3	0.1	3,282	65.0
	3	15,457	3,094.5	409.8	0.9	54,261	32.5
	4	15,316	4,143.4	543.3	9.0	535,880	13.1
1,400	1	25,354	625.7	151.4	0.0	100	100.0
	2	23,302	3,299.4	442.6	0.1	3,373	66.8
	3	22,794	4,856.0	639.5	1.0	53,873	32.3
	4	22,578	6,905.6	895.9	10.1	540,219	13.2
1,600	1	34,664	558.9	293.4	0.1	100	100.0
	2	32,460	19,969.1	2,572.5	0.1	3,282	65.0
	3	31,887	21,834.1	2,804.5	1.2	54,642	32.8
	4	31,737	33,050.8	4,212.2	11.2	551,884	13.5

Published in INFORMS Journal on Computing on July 27, 2022 as DOI: 10.1287/ijoc.2022.1225.
This article has not been copyedited or formatted. The final version may differ from this version.

4.1.2. Experiments on Selected DL8.5 Data Sets.

In validating the DL8.5 algorithm and demonstrating its superiority over the BinOCT algorithm (Verwer and Zhang 2019), Aglin et al. (2020) employs 24 binary classification data sets, all consisting of pure binary features. These data sets can directly form instances of OPT-G without the need for feature binarization. We selected 13 from the 24 data sets by the criteria $n \le 1,000$ and $p \le 200$ and compare Gurobi and ENUM solutions to OPT-G for the root node split rule identification on these data sets. The reason for the selection criteria is that Gurobi cannot solve the larger cases in a tolerable amount of time, that is, two hours. The results are presented in Table 4. The problem size is noted by n and m (for these cases, $p = m$). Sharp contrasts in solution time—up to a ratio of 10^6—between the two methods persists throughout all cases. In addition, for the Australian-credit and tic-tac-toe cases, Gurobi fails to terminate within the two-hour time limit, ending with a suboptimal solution in the latter case. In comparison, ENUM is able to find the optimal solutions consistently in less than 0.1 second. In cases in which Gurobi succeeds, both methods return the same optimal rule. These experiments serve to solidify our conclusion that ENUM should be the solver of choice when the `bsnsing` package is employed in practice.

4.2. Comparison with the DL8.5, OSDT, and GOSDT Algorithms

In this section, we compare `bsnsing` against three recently developed ODT methods, namely, DL8.5 (Aglin et al. 2020), OSDT (Hu et al. 2019), and GOSDT (Lin et al. 2020). The software programs were obtained through the Github links provided in the respective papers. Given that the model assumptions, formulation, and parameters are all different for the different tools under comparison, we do not aim to provide a comprehensive evaluation by, for instance, performing problem-specific parameter tuning and model interpretation or making inferences about which method is most suitable for what kind of data and applications. Instead, we exhibit our computational experience from a user's perspective, and let it convey the unique position of `bsnsing` among other recent ODT tools.

We experiment on the 24 data sets (i.e., binary classification problems with binary features) used in the DL8.5 paper (Aglin et al. 2020). The experiments were performed as follows. The comparison experiment on each data set was repeated 20 times. In each repetition, using an arbitrarily sequenced seed for the random number generator (RNG), we randomly split the data set into two parts, 70% for training and 30% for testing. For each method, we recorded the classification accuracy on the test set as well as the time taken

Table 4. Comparison Between Gurobi and ENUM for Root Node Split on DL85 Data Sets with `max.rules = 2`

	n	m	Method	CPU	Elapsed	Objval	Rule	nEval
anneal	812	93	ENUM	0.045	0.083	47,750	V60 | V67	2,204
			Gurobi	9,005.889	1,181.632	47,750	V60 | V67	0
audiology	216	148	ENUM	0.023	0.027	885	V2 | V32	5,768
			Gurobi	63.565	10.994	885	V2 | V32	0
australian-credit	653	125	ENUM	0.057	0.059	24,456	V34 | V76	4,493
			Gurobi	56,229.196	7,204.716	24,456	V38 | V76	0
breast-wisconsin	683	120	ENUM	0.051	0.053	9,039	V21 | V77	3,427
			Gurobi	5,276.419	691.689	9,039	V21 | V77	0
diabetes	768	112	ENUM	0.063	0.063	53,312	V20 | V76	3,180
			Gurobi	55,471.420	7,205.700	53,312	V20 | V76	0
heart-cleveland	296	95	ENUM	0.023	0.025	7,460	V91 | V96	2,483
			Gurobi	3,190.064	418.003	7,460	V91 | V96	0
hepatitis	137	68	ENUM	0.006	0.007	876	V37 | V51	1,183
			Gurobi	109.495	15.128	876	V37 | V51	0
lymph	148	68	ENUM	0.008	0.008	1,655	V39 | V56	1,562
			Gurobi	214.038	28.534	1,655	V39 | V56	0
primary-tumor	336	31	ENUM	0.006	0.006	7,728	V21 | V29	284
			Gurobi	530.928	70.518	7,728	V21 | V29	0
soybean	630	50	ENUM	0.016	0.017	19,964	V24 | V36	997
			Gurobi	2,304.774	306.424	19,964	V24 | V36	0
tic-tac-toe	958	27	ENUM	0.012	0.012	95,336	V15	378
			Gurobi	14,160.407	7,202.697	105,464	V28	0
vote	435	48	ENUM	0.011	0.013	3,547	V12	631
			Gurobi	1,027.852	346.244	3,547	V12	0
zoo-1	101	36	ENUM	0.002	0.002	0	V8	36
			Gurobi	0.277	0.283	0	V8	0

Published in INFORMS Journal on Computing on July 27, 2022 as DOI: 10.1287/ijoc.2022.1225.
This article has not been copyedited or formatted. The final version may differ from this version.

(in seconds) to train the respective models. For `bsnsing`, we adopted the default parameters, specifically, `opt.solver = "enum_c", max.rules = 2`, and set the `node.size` to $n^{1/4}$, where n is the training sample size. For DL8.5, we adopted the default parameters as recommended in the package's companion paper and in its online sample code, that is, "max_depth=3, min_sup=5". For OSDT, we adopted the default values for all optional parameters and set the required parameters to "lamb=0.005, prior_metric='curiosity"' by consulting the sample code provided in the author's GitHub page. For GOSDT, we set the required parameter "regularization" to 0.001,[3] a value found suitable in several experiments in the extended manuscript of the method; see the appendices of Lin et al. (2020). In addition, for OSDT and GOSDT, we set the time limit to five minutes (via setting the parameter "timelimit=300" in OSDT and "time_limit =300" in GOSDT) because we observed (which was also acknowledged by OSDT's developer) that the memory usage of the OSDT and GOSDT training process increases linearly and quickly with execution time.

The comparison results are listed in Table 5. The columns n, p, and MR characterize the data sets, and MR represents the rate of the minority class. The best accuracy (averaged over 20 runs) for each data set is highlighted in boldface. There is not a clear winner (or loser) in terms of prediction accuracy among the four methods; overall, all methods seem comparably capable. However, the differences in training time are significant. First, we can observe that OSDT and GOSDT are less competitive in training time. Another factor that discounts the OSDT algorithm's actual performance is that the OSDT program internally uses scikit-learn's decision tree classifier (which implements the CART algorithm) as its baseline predictor and returns the CART model if the OSDT algorithm cannot do better. In most cases, the OSDT program indeed returns the CART model for prediction (the CART column in Table 5 notes the proportion of runs in which the CART model was returned by the OSDT method), indicating that the OSDT algorithm did not outperform the CART baseline in these cases.

The `bsnsing` package also compares favorably to DL8.5 in training speed, especially for large instances. Let us look at the letter data set, which consists of 14,000 training samples for example: it took `bsnsing`, on average, 39.5 seconds to train a model that worked better than the DL8.5 model, which took 304.4 seconds to train on average. It is shown in prior work (i.e., Aglin et al. 2020), that DL8.5 runs orders of magnitude faster than several other optimal decision tree methods, including the original DL8 algorithm (Nijssen and Fromont 2007), a constrained programming–based method (Verhaeghe et al. 2020), and an MIP-based method BinOCT (Verwer and Zhang 2019). Hence, we can infer that

Table 5. Computational Comparison of Four ODT Packages

				bsnsing			DL8.5		OSDT			GOSDT	
	n	p	MR	Accu	CPU	Dp	Accu	CPU	Accu	CPU	CART	Accu	CPU
anneal	812	93	0.230	0.844	2.6	7	**0.847**	1.1	0.839	305.0	0.8	0.846	304.4
audiology	216	148	0.264	0.925	0.4	3	0.925	0.2	**0.938**	297.9	0.3	0.934	306.0
australian-credit	653	125	0.453	0.823	2.3	6	0.853	4.6	**0.858**	304.1	0.7	0.857	311.4
breast-wisconsin	683	120	0.350	0.953	0.8	4	**0.955**	3.1	0.951	304.6	0.3	0.943	307.8
diabetes	768	112	0.349	0.717	3.8	7	0.737	5.4	**0.750**	305.7	0.8	0.749	314.5
german-credit	1,000	112	0.300	0.676	5.0	7	**0.728**	4.2	0.723	305.2	0.6	0.708	319.9
heart-cleveland	296	95	0.459	0.752	1.2	5	**0.766**	1.9	0.763	303.7	0.9	0.746	306.9
hepatitis	137	68	0.190	0.790	0.4	4	0.802	0.5	0.775	306.0	1.0	**0.829**	303.1
hypothyroid	3,247	88	0.085	0.975	2.3	6	0.979	2.2	**0.979**	304.7	0.4	**0.979**	307.0
ionosphere	351	445	0.359	0.881	4.2	4	0.869	235.0	0.888	302.8	0.7	**0.912**	519.6
kr-vs-kp	3,196	73	0.478	**0.984**	2.0	7	0.936	1.3	0.969	303.9	1.0	0.853	307.9
letter	20,000	224	0.041	**0.990**	39.5	8	0.981	304.4	0.959	3,033.1	1.0	0.959	631.6
lymph	148	68	0.453	**0.810**	0.3	4	0.801	0.2	0.798	305.2	0.9	0.761	303.3
mushroom	8,124	119	0.482	**0.999**	2.7	4	0.999	4.4	0.994	305.1	0.0	0.943	314.4
pendigits	7,494	216	0.104	**0.994**	10.9	5	0.991	94.4	0.989	303.4	0.0	0.970	381.2
primary-tumor	336	31	0.244	0.778	0.7	6	**0.843**	0.1	0.822	304.9	0.2	0.751	169.6
segment	2,310	235	0.143	**0.996**	1.6	3	0.996	12.8	0.995	4.6	0.9	0.996	311.0
soybean	630	50	0.146	0.935	0.7	6	**0.941**	0.2	0.938	306.2	0.3	0.862	304.6
splice-1	3,190	287	0.481	**0.946**	18.2	7	0.926	51.6	0.946	303.0	1.0	0.835	308.4
tic-tac-toe	958	27	0.347	0.875	1.0	7	0.733	0.1	**0.901**	304.1	1.0	0.757	305.3
vehicle	846	252	0.258	0.952	3.1	5	**0.956**	33.6	0.946	303.5	0.8	0.879	354.0
vote	435	48	0.386	0.940	0.3	4	0.943	0.2	**0.956**	304.2	0.1	0.950	304.7
yeast	1,484	89	0.312	0.698	7.1	8	0.692	3.1	0.701	305.8	0.3	**0.701**	308.9
zoo-1	101	36	0.406	0.995	0.0	1	0.995	0.0	**1.000**	0.2	1.0	0.995	0.0

Note. Boldface entries mark the best accuracy achieved on the data set.

Published in INFORMS Journal on Computing on July 27, 2022 as DOI: 10.1287/ijoc.2022.1225.
This article has not been copyedited or formatted. The final version may differ from this version.

Table 6. Performance Comparison Under Depth Constraints

	Depth = 1				Depth = 2				Depth = 3			
	bs(2)	bs(1)	DL8.5	OSDT	bs(2)	bs(1)	DL8.5	OSDT	bs(2)	bs(1)	DL8.5	OSDT
anneal	0.778	0.778	**0.818**	0.818	0.786	0.778	**0.826**	0.824	0.806	0.777	**0.847**	0.836
audiology	**0.929**	0.856	0.856	0.856	0.925	0.924	**0.934**	**0.934**	0.925	0.923	0.925	**0.934**
australian-credit	0.854	**0.866**	**0.866**	**0.866**	0.854	**0.866**	0.851	0.862	0.849	0.852	**0.853**	0.849
breast-wisconsin	**0.933**	0.919	0.923	0.918	0.940	0.921	0.960	**0.960**	**0.956**	0.951	0.955	0.955
diabetes	0.716	0.725	**0.751**	**0.751**	0.754	0.734	0.754	**0.762**	0.747	**0.760**	0.737	0.755
german-credit	0.696	0.699	**0.703**	0.703	0.709	0.700	**0.717**	0.717	0.711	0.703	**0.728**	0.725
heart-cleveland	0.736	0.737	**0.737**	0.737	**0.753**	0.735	0.735	0.729	0.774	**0.811**	0.766	0.798
hepatitis	0.798	0.802	**0.844**	0.840	0.812	0.814	**0.831**	0.817	0.787	0.794	**0.802**	0.800
hypothyroid	0.963	**0.963**	**0.963**	**0.963**	0.973	0.963	**0.979**	**0.979**	0.977	0.969	0.979	**0.979**
ionosphere	**0.912**	0.758	0.820	0.820	0.894	0.837	0.901	**0.904**	0.875	0.851	0.869	**0.891**
kr-vs-kp	**0.772**	0.684	0.678	0.678	**0.934**	0.868	0.868	0.868	**0.949**	0.923	0.936	0.903
letter	**0.959**	**0.959**	**0.959**	**0.959**	0.959	0.959	**0.969**	0.968	**0.982**	0.958	0.981	0.959
lymph	0.743	**0.772**	0.752	**0.772**	**0.780**	0.777	0.770	0.766	**0.811**	0.803	0.801	0.784
mushroom	**0.951**	0.887	0.887	0.887	**0.983**	0.915	0.969	0.969	**0.999**	0.969	0.999	0.994
pendigits	**0.967**	0.895	0.931	0.931	**0.988**	0.978	0.978	0.978	0.991	0.987	**0.991**	0.987
primary-tumor	0.765	**0.775**	0.765	0.769	**0.806**	**0.806**	0.801	0.803	0.803	0.797	**0.843**	0.836
segment	**0.991**	0.926	0.981	0.981	**0.996**	0.995	0.995	0.990	**0.996**	0.995	0.996	0.995
soybean	**0.862**	**0.862**	**0.862**	**0.862**	0.854	0.862	0.906	**0.913**	0.913	0.890	**0.941**	0.920
splice-1	**0.824**	0.816	0.816	0.816	**0.875**	0.835	0.827	0.831	**0.943**	0.909	0.926	0.908
tic-tac-toe	0.665	0.669	**0.703**	**0.703**	**0.693**	0.683	0.673	0.678	0.744	**0.745**	0.733	0.735
vehicle	**0.865**	0.736	0.755	0.755	**0.913**	0.895	0.904	0.904	0.944	0.905	**0.956**	0.907
vote	0.948	**0.957**	**0.957**	**0.957**	0.948	**0.957**	0.950	0.956	0.944	0.944	0.943	**0.948**
yeast	0.687	0.687	**0.702**	**0.702**	**0.705**	0.687	0.690	0.696	0.694	0.694	0.692	**0.702**
zoo-1	0.995	0.995	0.995	**1.000**	**0.995**	**0.995**	**0.995**	**0.995**	**0.995**	**0.995**	**0.995**	**0.995**
Average	**0.846**	0.822	0.834	0.835	**0.868**	0.854	0.866	0.867	0.880	0.871	**0.883**	0.879

Note. Boldface entries mark the best accuracy at the given tree depth.

`bsnsing` must also outrun those other methods by a substantial margin. Overall, it is reasonable to claim that `bsnsing`, bearing comparable prediction accuracy, stands out in training speed among decision tree methods that involve solving mathematical optimization problems in the training process.

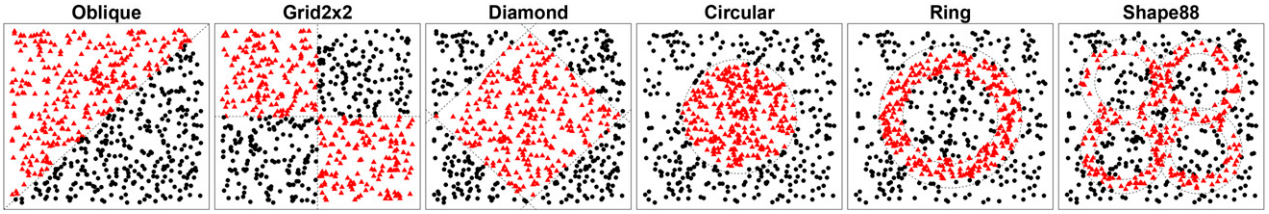
In most decision tree induction methods, depth (i.e., level distance between the root node and the deepest node in a tree) is a hyperparameter for adjusting the classifier's complexity with regard to the bias–variance tradeoff. Heuristic methods, such as CART, typically realize depth control via tree pruning, whereas most ODT methods can explicitly constrain the maximum depth in the optimization models. However, the `bsnsing` method does not endogeneously handle a constraint on the maximum depth. The "Dp" column in Table 5 lists the mode (most frequent value out of the 20 runs) of the depth of the `bsnsing` trees. We can see that, compared with trees built by DL8.5, which have a default "max_depth" of three, the trees built by `bsnsing` generally reach deeper though not all branches extend to the same depth.

Some users might take depth as a proxy for interpretability of a decision tree; shallower trees or trees with fewer leaves are deemed more interpretable than deeper trees. To facilitate performance comparison with depth-constrained ODT trees, we can naively prune a `bsnsing` tree so as to keep the number of leaf nodes below that of a binary tree of a given depth. For instance, a tree of depth one, two, and three would have at most two, four, and eight leaf nodes, respectively. Using this method, we repeat the preceding experiments (over the 24 binary classification data sets, 20 runs for each) under different maximum depth values. The average out-of-sample accuracies are reported in Table 6 with the best value in each depth group highlighted in boldface. Two `max.rules` settings are tested for `bsnsing`: the default setting with `max.rules=2` reported in column bs(2) and the `max.rules=1` setting reported in column bs(1). Because the GOSDT package does not have a parameter to limit the maximum depth or the number of leaves, it is not part of the experimentation. Also, all the OSDT runs with depth = 2 and 3 have hit the five-minute time limit.[4]

We can see that the pruned `bsnsing` trees remain quite competitive, in many cases outperforming the DL8.5 and OSDT trees of the same depth. The multivariate splits (with `max.rules=2`) clearly give `bsnsing` an advantage in these comparisons. Under `max.rules=1`, the pruned `bsnsing` trees become the least accurate in most cases. Though the `max.rules=1` setting, along with the naive pruning, is not recommended for `bsnsing`'s practical use, comparing bs(1) with DL8.5 and OSDT does highlight the benefits of holistic optimization in tree induction as argued in several ODT papers. Another interesting, yet expected, observation is that the constraint on depth, no matter

Published in INFORMS Journal on Computing on July 27, 2022 as DOI: 10.1287/ijoc.2022.1225.
This article has not been copyedited or formatted. The final version may differ from this version.

Figure 4. (Color online) Synthetic Data Sets for Pattern Recognition



Notes. Input variables are the x and y coordinates. Some slanted and nonlinear class boundaries are unamenable to the rectilinear split boundaries produced by tree models.

how it is realized in different packages, does not affect the new-data prediction accuracy in any deterministic direction (increase or decrease). An optimal (unconstrained or unpruned) tree may perform worse than a depth-constrained (or naively pruned as in the `bsnsing` case) tree in some cases. This observation enhances the understanding that, in machine learning algorithms, the notion of optimality only applies to the training problem, not to the inference problem. In other words, there is no single algorithm or parameter setting that is best performing in all cases. Moreover, comparing the average (across all data sets) accuracies of the pruned `bsnsing` trees in column bs(2) and the average accuracy (0.885) of the original `bsnsing` trees in Table 5, we can see that the naive pruning strategy generally hurts performance, upholding the effectiveness of `bsnsing`'s algorithm design and the default parameter setting.

4.3. Comparison with Other Decision Tree Packages in R

In this section, we compare the out-of-the-box performance (i.e., using all default options and no hyperparameter tuning) of the `bsnsing` package against several other decision tree packages, namely, `C50`, `party`, `tree`, and `rpart`, that are available on the Comprehensive R Archive Network. Then, for those cases on which `bsnsing` performs poorly, we demonstrate some simple methods to improve the performance.

Data used in the benchmarking experiments include 57 data sets for binary classification and 18 data sets for multiclass classification. Among these 75 data sets, one (iris) is from the *datasets* package, one (bank) is from a FICO-sponsored explainable machine learning challenge (FICO 2018), two (compas and heloc) are from the ProPublica and Trusted-AI GitHub repositories, two (GlaucomaMVF and dystrophy) are from the *ipred* package, six (BreastCancer, Glass, smiley, spirals, xor, and Sonar) are from/generated by the *mlbench* package, six (obli, grid, diam, circ, ring, and sha88) are synthetic data sets for 2-D pattern recognition (see Figure 4), and the remaining 58 data sets are sourced from the UCI Machine Learning Repository (Dua and Graff 2017). The names, number of observations (n), number of independent variables (p), (for binary-class) rate of the minority class (MR), and (for multiclass) number

Table 7. Binary Classification Data Sets

Name	n	p	MR	Name	n	p	MR	Name	n	p	MR
acute1	120	6	0.492	haberman	306	3	0.265	pima	768	8	0.349
acute2	120	6	0.417	heart	303	13	0.459	Qsar	1,055	41	0.337
Adult	32,561	13	0.241	heloc	10,459	23	0.478	relax	182	12	0.286
auto	392	7	0.375	hepatitis	155	19	0.206	retention	10,000	8	0.338
bank	45,211	16	0.117	HTRU2	17,898	8	0.092	ring	600	2	0.500
banknote	1,372	4	0.445	ILPD	583	10	0.286	seismic	1,690	18	0.031
birthwt	189	9	0.312	Ionos	351	34	0.359	sh88	600	2	0.500
BreastCancer	699	9	0.345	magic04	19,020	10	0.352	Sonar	208	60	0.466
circ	600	2	0.500	mammo	830	5	0.486	spambase	4,601	57	0.394
climate	540	18	0.085	Monks1	556	6	0.500	SPECT	267	22	0.206
compas	7,214	52	0.451	Monks2	601	6	0.343	spirals	600	2	0.500
connect	208	60	0.466	Monks3	554	6	0.480	statlog.a	690	14	0.445
credit	690	15	0.445	Mushroom	8,124	21	0.482	thoraric	470	16	0.149
diam	600	2	0.500	norm3p10	600	10	0.500	tictactoe	958	9	0.347
dystrophy	209	9	0.359	norm3p5	600	5	0.500	titanic	2,201	3	0.323
Echocard	61	11	0.279	obli	600	2	0.500	trans	748	4	0.238
Fertility	100	9	0.120	ozone1	2,536	72	0.029	votes	435	16	0.386
GlaucomaMVF	170	66	0.500	ozone8	2,534	72	0.063	wdbc	569	30	0.373
grid	600	2	0.475	parkins	195	22	0.246	wpbc	198	33	0.237

Published in INFORMS Journal on Computing on July 27, 2022 as DOI: 10.1287/ijoc.2022.1225.
This article has not been copyedited or formatted. The final version may differ from this version.

Table 8. Multiclass Classification Data Sets

Name	n	p	J	Name	n	p	J	Name	n	p	J
derm	366	34	6	imgsegm	210	19	7	thyroid	3,772	21	3
iris	150	4	3	Hayes	132	4	3	wine	178	13	3
smiley	500	2	4	contra	1,473	9	3	WineQuality	4,898	11	7
xor3	600	3	4	balance	625	4	3	nursery	12,960	8	5
Glass	214	9	6	soybean.l	266	35	15				
optdigits	5,620	64	10	soybean.s	47	35	4				
Seeds	210	7	3	tae	151	5	3				

of target classes (J) are listed in Tables 7 and 8 for binary and multiclass classification data sets, respectively. This collection covers most of the commonly used data sets for methodology benchmarking in the classification tree literature; hence, the data collection itself can be useful for future research. These data sets are accessible by name in the R environment once the `bsnsing` library is loaded. The R scripts for conducting the subsequent experiments are not part of the library, but will be published in a different repository.

4.3.1. Out-of-the-Box Performance Comparison. The experiments are conducted as follows. For each data set, we randomly split all observations into two parts, 70% for training and 30% for testing. The training set is fed into different decision tree functions to build the respective tree models, and then, the models are fed into the `predict` functions of the respective packages to make predictions on the test set. Accuracy and the area under the receiver operating characteristic (ROC) curve (AUC) values are calculated for each method based on the prediction results. To calculate the accuracy, class label predictions are requested from the predict functions, and to calculate the AUC, score (or probability) predictions are requested from the predict functions. The whole process (i.e., random 70/30 split, training, and testing) is repeated 20 times with documented RNG seeds for each data set, and the corresponding mean accuracy and mean AUC (for binary classification) are reported in Tables 9 and 10. The average computing time in seconds of the bsnsing method is also reported under the CPU column in the tables. The computing times of other methods are consistently below one second for all test cases; thus, they are omitted from the report.

There are a few points to note: (1) in each run, all five methods were fed with the same training and test sets, so the comparison was apple-to-apple; (2) all the original p independent variables contained in each data set were used—no prior variable selection was done; and (3) the tree-building functions from all five packages were called in the simplest form, that is, only the "regression formula" and the training data set were supplied as arguments in the function calls, to produce results that represent the out-of-the-box performance.

In Tables 9 and 10, the best accuracy and AUC values in each data set are highlighted in italic font. In addition, the `bsnsing` results that are above average among the five methods are printed in boldface. We can see that, for binary classification tasks, the `bsnsing` package is in the leading position under the AUC category; it won 28 cases out of 57, whereas tree, C50, ctree, and rpart won 14, 10, 8, and 3 cases, respectively. The `bsnsing` package also scored above average in 48 data sets, that is, in 84% of all cases. Therefore, as far as the AUC performance is concerned, `bsnsing` should be the package of choice for binary classification tasks. In terms of the accuracy metric, C50 is clearly the leading one, winning 27 cases. For practitioners, we comment that AUC represents a model's ability to correctly rank order new data points according to their likelihood of belonging to the target class. The specific score threshold for making classifications is usually application-dependent, for example, depending on the comparative costs of making an FP claim versus making an FN claim about a given new case. In contrast, the classification accuracy measures the overall proportion of false claims, that is, by treating FP and FN claims with equal weight, at a chosen score threshold. Therefore, we remark that AUC is a more well-rounded performance metric than classification accuracy for binary classifiers.

Whereas the OPT-G model is only applicable for binary classification, the `bsnsing` function can handle multiclass classification tasks as well. When more than two unique levels of the target variable are present in the training data set, a binary classification tree is built for each level (as the positive class) versus all the other levels (as the negative class). In the prediction stage, a score (i.e., probability prediction) is produced from each tree, and the target level having the greatest score is assigned as the class label for the new case. From Table 10, we can see that `bsnsing`'s multiclass performance is second only to C50. A caveat is that the current way `bsnsing` handles multiclass classification tasks is more ensemble learning rather than decision tree learning, and the model's interpretability is not preserved.

Published in INFORMS Journal on Computing on July 27, 2022 as DOI: 10.1287/ijoc.2022.1225.
This article has not been copyedited or formatted. The final version may differ from this version.

Table 9. Comparison on Binary Classification Cases

	Mean accuracy					Mean AUC					
	C5.0	ctree	rpart	tree	bsnsing	C5.0	ctree	rpart	tree	bsnsing	CPU
acute1	*1.000*	0.903	0.912	0.994	0.896	*1.000*	0.957	0.932	0.996	0.970	0.1
acute2	*1.000*	0.954	0.958	*1.000*	***1.000***	*1.000*	0.989	0.965	*1.000*	***1.000***	0.0
Adult	*0.864*	0.847	0.832	0.832	0.825	0.887	*0.894*	0.818	0.852	**0.870**	82.0
auto	*0.897*	0.871	0.882	0.879	0.864	*0.953*	0.901	0.938	0.923	**0.941**	0.3
bank	0.902	*0.904*	0.901	0.890	**0.901**	0.880	*0.915*	0.761	0.881	**0.900**	71.0
banknote	*0.981*	0.967	0.966	0.978	0.963	*0.985*	0.977	0.976	0.984	***0.985***	0.5
birthwt	0.649	*0.674*	0.661	0.626	0.629	0.541	0.486	0.581	0.584	***0.587***	0.3
BreastCancer	0.942	0.946	0.940	*0.950*	**0.945**	0.968	0.974	0.950	0.969	***0.978***	0.3
circ	*0.951*	0.580	0.936	0.940	**0.906**	0.961	0.592	0.942	*0.966*	**0.956**	0.2
climate	0.922	0.920	*0.929*	0.921	**0.925**	0.810	0.809	0.809	0.757	***0.813***	0.6
compas	0.890	0.889	*0.891*	*0.891*	0.890	0.918	0.934	0.884	0.919	***0.934***	4.0
connect	0.702	0.694	*0.727*	0.719	**0.721**	0.755	0.749	0.778	0.770	***0.788***	2.0
credit	0.852	*0.852*	0.851	0.840	**0.851**	0.897	0.909	0.902	0.890	***0.913***	1.0
diam	*0.923*	0.477	0.890	0.906	**0.906**	0.947	0.503	0.919	0.944	***0.953***	0.3
dystrophy	*0.841*	0.821	0.822	0.837	0.813	0.846	0.853	0.823	*0.857*	0.838	0.2
Echocard	0.958	0.936	*0.966*	*0.966*	0.947	0.954	0.953	0.963	*0.967*	0.934	0.0
Fertility	0.882	*0.893*	0.865	0.850	0.860	0.545	0.535	0.578	0.635	***0.664***	0.1
GlaucomaMVF	0.890	0.836	0.895	0.879	***0.900***	0.933	0.894	0.946	*0.948*	***0.948***	1.0
grid	0.520	0.520	*0.976*	0.610	**0.972**	0.505	0.505	*0.990*	0.615	***0.990***	0.1
haberman	*0.736*	0.717	0.726	0.714	**0.729**	0.544	0.619	0.640	0.649	***0.672***	0.3
heart	0.777	0.752	*0.792*	0.766	0.768	0.811	0.805	*0.822*	0.808	***0.822***	0.4
heloc	*0.706*	0.696	0.700	0.697	0.698	0.749	0.757	0.706	0.736	***0.758***	37.0
hepatitis	0.795	0.791	0.789	*0.808*	**0.798**	0.716	0.706	0.677	0.716	***0.764***	0.2
HTRU2	*0.979*	0.979	0.978	0.977	0.977	0.948	*0.974*	0.909	0.969	**0.966**	23.0
ILPD	0.678	*0.704*	0.681	0.673	0.682	0.675	0.665	0.657	*0.681*	**0.677**	0.8
Ionos	0.892	*0.905*	0.870	0.874	0.859	*0.920*	0.901	0.901	0.901	0.894	1.0
magic04	*0.850*	0.844	0.819	0.814	**0.840**	0.885	0.892	0.811	0.842	***0.893***	106.0
mammo	0.828	0.807	*0.830*	0.823	**0.824**	0.869	0.855	0.868	0.878	***0.888***	0.7
Monks1	*0.898*	0.743	0.840	0.743	**0.871**	0.899	0.739	0.916	0.739	***0.956***	0.3
Monks2	*0.925*	0.650	0.750	0.656	0.607	*0.973*	0.492	0.800	0.540	0.598	0.7
Monks3	*0.989*	0.961	0.977	0.986	0.960	0.987	0.983	0.979	*0.989*	**0.986**	0.2
Mushroom	*1.000*	0.999	0.994	0.999	0.987	1.000	*1.000*	0.994	0.999	**0.999**	4.0
norm3p10	*0.768*	0.749	0.758	0.766	0.756	0.823	0.814	0.801	0.815	***0.831***	1.0
norm3p5	*0.838*	0.829	0.831	0.836	**0.835**	0.878	0.877	0.865	*0.896*	**0.892**	0.5
obli	*0.945*	0.922	0.914	0.935	0.902	*0.962*	0.955	0.942	0.957	**0.961**	0.2
ozone1	0.969	*0.972*	0.964	0.956	0.956	0.614	*0.784*	0.674	0.632	**0.773**	6.0
ozone8	0.930	*0.932*	0.930	0.923	0.922	*0.777*	*0.805*	0.750	0.712	**0.795**	17.0
parkins	0.854	0.847	0.861	*0.868*	**0.858**	0.837	0.819	0.854	*0.868*	0.842	0.4
pima	0.741	*0.746*	0.740	0.741	0.731	0.772	0.780	0.779	*0.787*	**0.780**	0.9
Qsar	*0.839*	0.807	0.822	0.818	0.819	0.862	0.850	0.841	0.860	***0.872***	5.0
relax	*0.726*	*0.726*	0.594	0.612	0.617	0.492	0.492	0.483	0.503	***0.509***	0.4
retention	*0.994*	0.971	0.941	0.929	0.935	*0.999*	0.989	0.950	0.966	**0.984**	6.0
ring	0.845	0.482	0.861	*0.881*	**0.782**	0.866	0.498	0.896	*0.918*	**0.856**	0.5
seismic	*0.970*	*0.970*	*0.970*	0.959	***0.970***	0.499	*0.625*	0.499	0.568	**0.570**	1.0
sh88	0.737	0.477	0.812	*0.813*	0.674	0.753	0.493	0.847	*0.851*	**0.745**	0.6
Sonar	0.702	0.694	*0.727*	0.719	**0.721**	0.755	0.749	0.778	0.770	***0.788***	2.0
spambase	*0.924*	0.906	0.896	0.902	**0.911**	0.957	0.949	0.899	0.948	***0.960***	27.0
SPECT	0.814	0.792	*0.829*	0.825	**0.821**	0.786	0.721	0.782	0.785	***0.803***	0.3
spirals	*0.948*	0.655	0.924	0.942	0.783	0.960	0.650	0.948	*0.962*	0.847	0.5
statlog.a	0.847	*0.857*	0.855	0.848	0.851	0.903	0.907	0.904	0.906	***0.912***	0.8
thoraric	0.842	*0.850*	0.831	0.791	0.817	0.505	0.517	0.524	*0.560*	**0.539**	0.7
tictactoe	*0.923*	0.825	0.901	0.880	0.803	*0.974*	0.915	0.961	0.953	0.869	1.0
titanic	*0.777*	*0.787*	0.781	0.782	**0.782**	0.705	*0.748*	0.710	0.710	**0.725**	0.2
trans	0.763	0.761	*0.775*	0.770	0.765	0.680	0.690	*0.709*	0.688	0.688	0.6
votes	*0.960*	0.957	0.949	0.954	0.943	*0.986*	0.978	0.962	0.984	**0.980**	0.2
wdbc	0.939	0.933	0.927	0.933	***0.944***	0.964	0.959	0.940	0.956	***0.967***	1.0
wpbc	0.722	0.726	0.692	0.692	***0.730***	0.562	0.592	0.586	0.586	***0.625***	0.9
Average	0.859	0.811	0.853	0.844	0.841	0.827	0.787	0.825	0.825	0.841	7.3

Published in INFORMS Journal on Computing on July 27, 2022 as DOI: 10.1287/ijoc.2022.1225.
This article has not been copyedited or formatted. The final version may differ from this version.

Table 10. Comparison on Multiclass Classification Cases

				Mean accuracy					
	n	p	J	C5.0	ctree	rpart	tree	bsnsing	CPU
balance	625	4	3	0.782	0.776	0.771	0.769	***0.816***	1.4
contra	1,473	9	3	0.514	0.537	*0.547*	0.520	0.525	5.0
derm	366	34	6	*0.949*	0.934	0.928	0.923	0.915	1.6
Glass	214	9	6	*0.664*	0.606	0.661	0.649	**0.649**	1.2
Hayes	132	4	3	*0.825*	0.494	0.625	0.745	**0.698**	0.5
imgsegm	210	19	7	*0.864*	0.784	0.843	0.848	0.828	1.1
iris	150	4	3	0.934	0.941	0.930	0.936	***0.945***	0.2
nursery	12,960	8	5	*0.992*	0.974	0.874	0.858	0.903	21.7
optdigits	5,620	64	10	*0.902*	0.844	0.768	0.774	**0.897**	70.8
Seeds	210	7	3	0.912	0.879	0.900	*0.922*	**0.922**	0.3
smiley	500	2	4	0.990	0.986	*0.992*	0.992	0.985	0.2
soybean.l	266	35	15	*0.886*	0.699	0.655	0.779	0.631	2.3
soybean.s	47	35	4	0.975	0.482	0.579	0.950	***0.986***	0.2
tae	151	5	3	*0.507*	0.379	0.478	0.489	0.455	0.8
thyroid	3,772	21	3	*0.997*	0.993	0.996	0.997	0.962	1.6
wine	178	13	3	0.920	0.897	0.889	*0.921*	**0.911**	0.3
WineQuality	4,898	11	7	*0.574*	0.532	0.532	0.513	0.534	39.3
xor3	600	3	4	0.711	0.217	*0.956*	0.709	**0.931**	1.1
Average				0.828	0.720	0.773	0.794	0.810	8.3

For `bsnsing`, the median time to train a binary classification model is 0.6 seconds, tallied over the 1,140 training instances (i.e., 57 data sets, 20 instances each), and the median time to train a multiclass classification model is 1.2 seconds, tallied over the 360 training instances. The computing time is much more tolerable than most (if not all) MIP-based optimal classification tree methods.

4.4. Usage Notes of the `bsnsing` Package

Alluding to the no free lunch theorem (Wolpert and Macready 1997), no single machine learning algorithm is universally the best performing algorithm for all problems. To be generally useful for classification problems, most decision tree algorithms allow users to control the behavior of the algorithm via choosing values for a number of hyperparameters. Such flexibility can be a double-edged sword to the usability of an algorithm. Seasoned users, most likely developers, can have the convenience of experimenting with the algorithm without changing the code, but ordinary users unconcerned with the internal workings of the underlying algorithm may find too many parameters perplexing. The large, sometimes infinite, value space of hyperparameters also presents practical challenges to automated parameter-tuning processes. For example, a clear-cut valley point of the generalization error curve in the bias–variance trade-off analysis (see chapter 2 of James et al. 2014) may be difficult to identify, especially when the available training samples are relatively few in a high-dimensional feature space.

To ease the usage, we provide some guidance for parameter selection for the `bsnsing` algorithm from an ordinary user's perspective. The most important parameters for `bsnsing` are `max.rules` and `node.size`. For `max.rules`, we recommend using the default value of two for a good balance between training speed and model performance. A higher value increases the solution time of OPT-G particularly at the root node as is observed in Table 3. In the meantime, a higher value does not necessarily translate to a better classification performance because of the heuristic nature of the recursive partitioning process. For `node.size`, we recommend using the default value of zero first, meaning to set the minimum node size dynamically. A larger value of `node.size` leads to a smaller (thus, more interpretable) tree but might underfit the data, whereas a smaller value leads to a bigger tree and might leave some true patterns undistinguished. If it is known that strong, learnable patterns exist in a data case, then manually setting `node.size` to a small value, that is, some positive integer smaller than $\sqrt{n}$, is likely to improve the classification performance over the default setting. Finally, if interpretability is unimportant, an ensemble of several `bsnsing` trees each trained with different hyperparameter values can effectively boost the performance.

Let us look at some concrete examples. We notice from Table 9 that, on three data sets, namely, Monks2, spirals, and tictactoe, `bsnsing` performed especially poorly compared with the best performing method. This suggests that discoverable patterns exist in these data sets and `bsnsing` could be configured more flexible at discovering them. Indeed, if we reduce the `node.size` value, a significant improvement in the out-of-sample performance can be realized for three cases as shown under the "Improved" columns in Table 11. A greater improvement is also achieved by

Published in INFORMS Journal on Computing on July 27, 2022 as DOI: 10.1287/ijoc.2022.1225.
This article has not been copyedited or formatted. The final version may differ from this version.

Table 11. Improve the `bsnsing` Performance via Changing `node.size` and Using Ensemble

	Original		Improved			Ensemble		
	Accu	AUC	Accu	AUC	Parameter	Accu	AUC	CPU
Monks2	0.607	0.598	0.895	0.898	node.size=1	0.736	0.921	8.8
spirals	0.783	0.847	0.927	0.943	node.size=3	0.911	0.974	5.0
tictactoe	0.803	0.869	0.911	0.934	node.size=3	0.920	0.982	12.0

the ensemble approach in which we trained a total of nine trees with parameter combinations of `max.rules` $\in [1,2,3]$ and `node.size` $\in [0,1,10]$. The class membership prediction is the result of majority voting, and the score prediction is the average of the scores predicted by the nine trees in the ensemble. The total time (in seconds) taken to train the nine trees remains quite manageable as shown in the column CPU in Table 11. More detailed usage examples with sample code are provided in the online appendix.

5. Conclusion and Future Work

In this paper, we propose a new method for classification tree induction that combines mathematical optimization with the recursive partitioning framework. The method optimally selects a boolean combination of multiple variables to maximize the Gini reduction in each node split. The split optimization model is the first that is able to maximize the well-justified but nonlinear Gini reduction metric in a mixed-integer linear program. We develop an efficient search algorithm to solve realistically regulated instances faster than commercial optimization solvers, making the overall solution scheme as well as the R package `bsnsing` more accessible to both practitioner and developer communities. Evaluation results suggest that the `bsnsing` package can generate the best AUC performance among other decision tree packages in R at the cost of a median training time of a few seconds.

A central theme in the design of classification tree algorithms is making trade-offs to strike a balance among competing objectives, such as speed, accuracy, and interpretability. We believe that optimization modeling is no substitute for the recursive partitioning framework; however, it can alleviate some structural restrictions via answering key design questions in a new light. One of the benefits of using mathematical optimization in decision tree induction is that it makes the process more tractable and justifiable. As observed in previous works and in this paper, properly regularized optimal trees do not lead to overfitting. Therefore, decision tree optimization is worthy of further development.

Several aspects of the present work can be extended. First, the OPT-G model exhibits a clear sparsity pattern that may be exploited to expedite the solution. For instance, it is possible to adapt the bounding technique in the ENUM algorithm to the branch-and-bound framework via adding user cuts and to develop multiple branch-and-bound trees for parallel computing. However, this requires the use of advanced callback functions, which are not currently supported in R APIs (for both Gurobi and CPLEX). Implementation in other languages could exploit these possibilities. Second, the feature binarization process is unoptimized, and the actual utility of the candidate split rules are unquantified. It is possible that a good proportion of the candidate rules are dominated by others and, hence, need not be generated in the first place. Future research could explore the column-generation paradigm to generate high-value binary features on the fly during the optimal selection process.

Acknowledgments

The author thanks the area editor Dr. Ram Ramesh, the associate editor, and three anonymous reviewers for their critical and constructive review comments that helped improve the quality of this work.

Appendix. Usage Demonstration of the `bsnsing` Package

To install the `bsnsing` package, an R user can run this command: `install.packages("bsnsing")`. The following code snippet demonstrates a stylized use case of building and evaluating a decision tree model.

```
library (bsnsing)
set.seed(2021) # Set seed for RNG in the sample() function
n <- nrow(BreastCancer)
trainset <- sample(1:n, 0.7*n) # randomly sample 70% for training
testset <- setdiff(1:n, trainset) # the remaining is for testing
# Build a tree to predict Class, using all default options
bs <- bsnsing(Class~., data = BreastCancer[trainset,])
summary(bs) # display the tree structure, see Figure A.2
pred <- predict(bs, BreastCancer[testset,], type='class')
actual <- BreastCancer[testset, 'Class']
table(pred, actual) # display the confusion matrix
# Plot the ROC curve and display the AUC
ROC_func(data.frame(predict(bs, BreastCancer[testset,]),
                    BreastCancer[testset,'Class']),
         2, 1, pos.label = 'malignant', plot.ROC=T)
# Plot the tree to PDF file and generate the latex source code
```

Published in INFORMS Journal on Computing on July 27, 2022 as DOI: 10.1287/ijoc.2022.1225.
This article has not been copyedited or formatted. The final version may differ from this version.

Figure A.1. (Color online) Summary Display of the bsnsing Tree for the BreastCancer Data Set

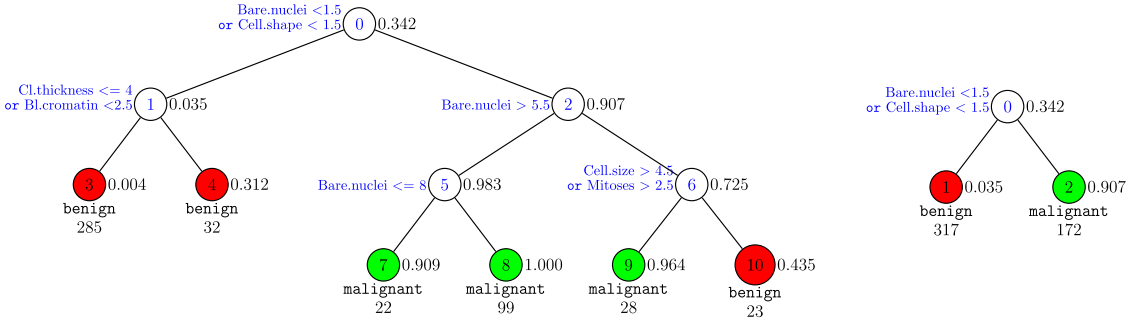
```
> summary(bs)
Call: summary.bsnsing(object = bs)
Response coding: malignant ==> 1   benign ==> 0
Decision Tree:
-------------
Node 0: class = 0 w/ prob 0.6585, prop 1.0000, n1 167, n0 322
Split by (Bare.nuclei <1.5 | Cell.shape < 1.5)
      Node 1: class = 0 w/ prob 0.9653, prop 0.6483, n1 11, n0 306
      Split by (Cl.thickness <= 4 | Bl.cromatin <2.5)
            *Node 3: class = 0 w/ prob 0.9965, prop 0.5828, n1 1, n0 284
            *Node 4: class = 0 w/ prob 0.6875, prop 0.0654, n1 10, n0 22
      Node 2: class = 1 w/ prob 0.9070, prop 0.3517, n1 156, n0 16
      Split by (Bare.nuclei > 5.5)
            Node 5: class = 1 w/ prob 0.9835, prop 0.2474, n1 119, n0 2
            Split by (Bare.nuclei <= 8)
                  *Node 7: class = 1 w/ prob 0.9091, prop 0.0450, n1 20, n0 2
                  *Node 8: class = 1 w/ prob 1.0000, prop 0.2025, n1 99, n0 0
            Node 6: class = 1 w/ prob 0.7255, prop 0.1043, n1 37, n0 14
            Split by (Cell.size > 4.5 | Mitoses > 2.5)
                  *Node 9: class = 1 w/ prob 0.9643, prop 0.0573, n1 27, n0 1
                  *Node 10: class = 0 w/ prob 0.5652, prop 0.0470, n1 10, n0 13
-------------
The tree has 5 splits (0 fractional), 6 leaf nodes.
Confusion matrix:
      actual
fitted   0   1
     0 319  21
     1   3 146
Accuracy: 0.9509
>
```

17 plot(bs, file='./bsnsing_test/fig/BreastCancer.pdf') # see Figure A.3 left

The model summary (generated by line 8) prints out the tree structure as well as node information in plain text, shown in Figure A.1. We can read from the printout, for example, that the root node (node 0) is classified as 0 (benign) with probability 0.6585 and 100% of all training observations fall in this node of which 167 observations are class 1 and 322 observations are class 0. The confusion matrix on the training set is given at the end of the summary print. Detailed information of the `bsnsing` tree object can be accessed by the R command `str(bs)`.

The `bsnsing` package implements the S3 method `plot` for plotting the `bsnsing` object (see line 17). If a file name is provided (as shown in code), the function saves the latex scripts (that utilize the `tikz` package) to a .tex file and attempts to build the .ps and .pdf files by calling `latex`, `dvips`, and `ps2pdf` commands if they are installed. The plot is shown in Figure A.2 left. Each node is represented by a circle with the node number printed inside the circle. The color of a leaf node indicates its predicted class, green for positive (class 1) and red for negative (class 0). The split rule is shown on the left of each internal node in blue, and the class 1 probability of the node is shown on the right of the node. At the bottom of each leaf node, the predicted label (in this case, malignant or benign), along with the number of training observations that fall in the node, is printed. Of course, these features can be easily customized and

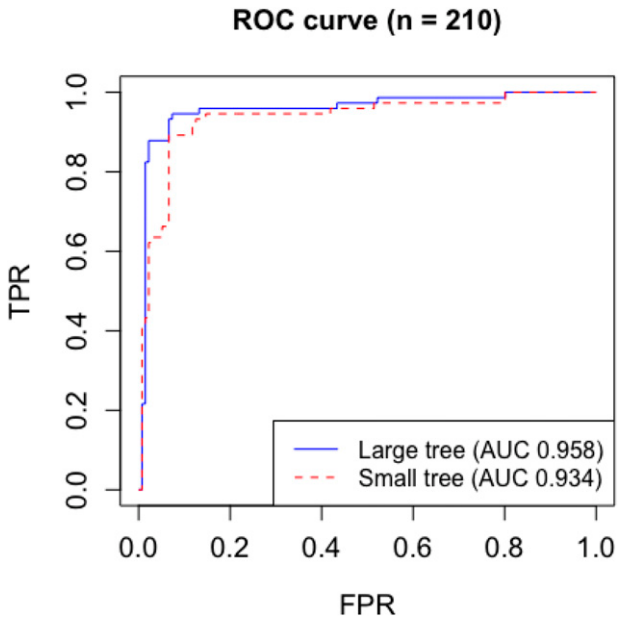
Figure A.2. (Color online) Discriminability vs. Interpretability



Note. The left tree is built with the default options, and the right tree is built with option `no.same.gender.children = true`.

Published in INFORMS Journal on Computing on July 27, 2022 as DOI: 10.1287/ijoc.2022.1225.
This article has not been copyedited or formatted. The final version may differ from this version.

Figure A.3. (Color online) ROC Curves Constructed on 210 Test Cases of the BreastCancer Data Set for Two bsnsing Trees



Note. The large tree was built with the default options, and the small tree with option `no.same.gender.children = true`.

extended by other developers. The right side of Figure A.2 plots a smaller tree generated on the same training set with the option `no.same.gender.children = true` to suppress splits that generate child nodes having the same majority class. Figure A.3 compares the ROC curves of these two trees. In these particular cases, some substantial improvement in interpretability is only accompanied by a slight drop in AUC, so the smaller tree (in the author's opinion) is preferred. Try-and-compare is a common practice in predictive analytics, and the bsnsing library is generally fast enough and flexible enough to support such practice.

Endnotes

[1] The phrase "search tree" is an analogy; in implementation, a queue is used for storing yet-to-explore solutions.

[2] Setting `opt.solver = "enum_c"` invokes the compiled code (written in C) implementing the ENUM algorithm as opposed to using the plain R implementation, which can be invoked by setting `opt.solver = "enum"`.

[3] The regularization parameter in GOSDT is the multiplier on the number-of-leaves term in the two-term objective function to be minimized, whereas the other term is the training accuracy (with multiplier one).

[4] Without the time limit, OSDT would, in many cases, exhaust the computer memory before terminating.

References

Aghaei S, Gomez A, Vayanos P (2020) Learning optimal classification trees: Strong max-flow formulations. Preprint, submitted May 13, https://arxiv.org/abs/2002.09142.

Aglin G, Nijssen S, Schaus P (2020) Learning optimal decision trees using caching branch-and-bound search. *Proc. Conf. AAAI Artificial Intelligence*, vol. 34, 3146–3153.

Agrawal R, Imielinski T, Swami A (1993) Mining association rules between sets of items in large databases. *Proc. 1993 ACM SIGMOD Internat. Conf. Management Data*, 207–216.

Alaradi M, Hilal S (2020) Tree-based methods for loan approval. *2020 Internat. Conf. Data Analytics Bus. Indust. Way Toward Sustainable Econom.*, 1–6.

Angelino E, Larus-Stone N, Alabi D, Seltzer M, Rudin C (2017) Learning certifiably optimal rule lists. *Proc. 23rd ACM SIGKDD Internat. Conf. Knowledge Discovery Data Mining*, (Association for Computing Machinery, New York), 35–44.

Bertsimas D, Dunn J (2017) Optimal classification trees. *Machine Learn.* 106(7):1039–1082.

Bertsimas D, Chang A, Rudin C (2012) An integer optimization approach to associative classification. *Proc. Neural Inform. Processing Systems*, 269–277.

Bertsimas D, Dunn J, Pawlowski C, Zhuo YD (2019) Robust classification. *INFORMS J. Optim.* 1(1):2–34.

Borgelt C (2012) Frequent item set mining. *Wiley Interdisciplinary Rev. Data Mining Knowledge Discovery* 2(6):437–456.

Breiman L, Friedman J, Stone CJ, Olshen RA (1984) *Classification and Regression Trees* (Taylor & Francis).

Dua D, Graff C (2017) UCI machine learning repository. Accessed December 15, 2021, http://archive.ics.uci.edu/ml.

FICO (2018) Explainable machine learning challenge. Accessed December 15, 2021, https://comm unity.fico.com/s/explainable-machine-learning-challenge.

Ghiasi MM, Zendehboudi S, Mohsenipour AA (2020) Decision tree-based diagnosis of coronary artery disease: Cart model. *Comput. Methods Programs Biomedicine* 192:105400.

Goh ST, Rudin C (2014) Box drawings for learning with imbalanced data. *Proc. 20th ACM SIGKDD Internat. Conf. Knowledge Discovery Data Mining*, (Association for Computing Machinery, New York), 333–342.

Holte RC (1993) Very simple classification rules perform well on most commonly used datasets. *Machine Learn.* 11(1):63–90.

Hu X, Rudin C, Seltzer MI (2019) Optimal sparse decision trees. Preprint, submitted April 29, https://arxiv.org/abs/1904.12847.

Hyafil L, Rivest RL (1976) Constructing optimal binary decision trees is NP-complete. *Inform. Processing Lett.* 5(1):15–17.

James G, Witten D, Hastie T, Tibshirani R (2014) *An Introduction to Statistical Learning: With Applications in R* (Springer Publishing Company, Inc.).

Kass GV (1980) An exploratory technique for investigating large quantities of categorical data. *J. Roy. Statist. Soc. Ser. C Appl. Statist.* 29(2):119–127.

Letham B, Rudin C, McCormick TH, Madigan D (2015) Interpretable classifiers using rules and Bayesian analysis: Building a better stroke prediction model. *Ann. Appl. Statist.* 9(3): 1350–1371.

Lin J, Zhong C, Hu D, Rudin C, Seltzer M (2020) Generalized and scalable optimal sparse decision trees. *Internat. Conf. Machine Learn.*, 6150–6160.

Liu B, Hsu W, Ma Y (1998) Integrating classification and association rule mining. *Proc. Fourth Internat. Conf. Knowledge Discovery Data Mining*, 80–86.

Loh WY (2009) Improving the precision of classification trees. *Ann. Appl. Statist.* 3(4):1710–1737.

Malioutov D, Varshney K (2013) Exact rule learning via boolean compressed sensing. Dasgupta S, McAllester D, eds. *Proc. 30th Internat. Conf. Machine Learn.*, vol. 28 (PMLR, Atlanta), 765–773.

Mandala IGNN, Nawangpalupi CB, Praktikto FR (2012) Assessing credit risk: An application of data mining in a rural bank. *Procedia Econom. Finance* 4:406–412.

Nijssen S, Fromont E (2007) Mining optimal decision trees from itemset lattices. *Proc. 13th ACM SIGKDD Internat. Conf. Knowledge*

Discovery Data Mining (Association for Computing Machinery, New York), 530–539.

Nijssen S, Fromont E (2010) Optimal constraint-based decision tree induction from itemset lattices. *Data Mining Knowledge Discovery* 21(1):9–51.

Quinlan JR (1993) *C4.5: Programs for Machine Learning* (Morgan Kaufmann Publishers Inc., San Francisco).

Quinlan JR, Cameron-Jones RM (1995) Oversearching and layered search in empirical learning. *Proc. 14th Internat. Joint Conf. Artificial Intelligence*, vol, 2 (Morgan Kaufmann Publishers Inc., San Francisco), 1019–1024.

Rijnbeek PR, Kors JA (2010) Finding a short and accurate decision rule in disjunctive normal form by exhaustive search. *Machine Learn.* 80(1):33–62.

Sorensen EH, Miller KL, Ooi CK (2000) The decision tree approach to stock selection. *J. Portfolio Management* 27(1):42–52.

Street WN (2005) Oblique multicategory decision trees using nonlinear programming. *INFORMS J. Comput.* 17(1):25–31.

Tan P-N, Steinbach M, Kumar V (2005) *Introduction to Data Mining*, 1st ed. (Pearson).

Tanner L, Schreiber M, Low JGH, Ong A, Tolfvenstam T, Lai YL, Ng LC, et al. (2008) Decision tree algorithms predict the diagnosis and outcome of dengue fever in the early phase of illness. *PLOS Neglected Tropical Diseases* 2(3):1–9.

Verhaeghe H, Nijssen S, Pesant G, Quimper C-G, Schaus P (2020) Learning optimal decision trees using constraint programming. *Constraints* 25:1–25.

Verwer S, Zhang Y (2017) Learning decision trees with flexible constraints and objectives using integer optimization. Salvagnin D, Lombardi M, eds. *Integration of AI and OR Techniques in Constraint Programming* (Springer International Publishing, Cham, Switzerland), 94–103.

Verwer S, Zhang Y (2019) Learning optimal classification trees using a binary linear program formulation. *Proc. 33rd AAAI Conf. Artificial Intelligence* (AAAI Press), 1625–1632.

Wang T, Rudin C, Doshi-Velez F, Liu Y, Klampfl E, MacNeille P (2017) A Bayesian framework for learning rule sets for interpretable classification. *J. Machine Learn. Res.* 18(70):1–37.

Wolpert DH, Macready WG (1997) No free lunch theorems for optimization. *IEEE Trans. Evolutionary Comput.* 1(1):67–82.

Yang H, Rudin C, Seltzer M (2017) Scalable Bayesian rule lists. Precup D, the YW, eds. *Proc. 34th Internat. Conf. Machine Learn.*, vol. 70 (PMLR), 3921–3930.

Zhu H, Murali P, Phan DT, Nguyen LM, Kalagnanam J (2020) A scalable MIP-based method for learning optimal multivariate decision trees. Larochelle H, Ranzato MA, Hadsell R, Balcan MF, Lin HT, eds. *Adv. Neural Inform. Processing Systems 33: Annual Conf. Neural Inform. Processing Systems 2020.*