Optimal Decision Trees for Categorical Data via Integer Programming

Oktay Günlük · Jayant Kalagnanam · Minhan Li · Matt Menickelly · Katya Scheinberg

Abstract Decision trees have been a very popular class of predictive models for decades due to their interpretability and good performance on categorical features. However, they are not always robust and tend to overfit the data. Additionally, if allowed to grow large, they lose interpretability. In this paper, we present a mixed integer programming formulation to construct optimal decision trees of a prespecified size. We take the special structure of categorical features into account and allow combinatorial decisions (based on subsets of values of features) at each node. Our approach can also handle numerical features via thresholding. We show that very good accuracy can be achieved with small trees using moderately-sized training sets. The optimization problems we solve are tractable with modern solvers.

Keywords Decision Trees · Integer Programming · Machine Learning · Binary Classification

1 Introduction

Interpretability has become a well-recognized goal for machine learning models as they push further into domains such as medicine, criminal justice, and business. In many of these applications machine learning models complement domain experts and for human decision-makers to trust these models, interpretability is crucial. Decision trees have been a very popular class of predictive models for decades due to their interpretability and good performance on categorical features. Decision trees (DTs, for short) are similar to flow-charts as they apply a sequence of binary tests or *decisions* to predict the output label of the input data. As they can be easily interpreted and applied by non-experts, DTs are considered as one of the most widely used tools of machine learning and

· Oktay Günlük, IBM Research, E-mail: gunluk@us.ibm.com

· Jayant Kalagnanam, IBM Research, E-mail: jayant@us.ibm.com

· Minhan Li, Lehigh University, E-mail: mil417@lehigh.edu

· Matt Menickelly, Argonne National Laboratory, E-mail: mmenickelly@anl.gov

· Katya Scheinberg, Cornell University, E-mail: katyas@cornell.edu

data analysis (see the recent survey [11] and references therein). Another advantage of DTs is that they often naturally result in feature selection, as only a part of the input is typically used in the decision-making process. Furthermore, DTs can work with both numerical and categorical data directly, which is not the case for *numerical classifiers* such as linear classifiers or neural networks, as these methods require the data to be real-valued (and ordinal). For example, if a categorical feature can take three values such as (*i*) red, (*ii*) blue, or, (*iii*) yellow, it is often represented by a group of three binary features such that one of these features takes the value 1 while the other two are 0. A numerical classifier would treat this group of three features independently where any combination of 0/1 values are possible - ignoring the valuable information that only three values for the triplet are possible. Numerical classifiers typically recover this lost information by observing enough data and fitting the model accordingly. However, this is not a trivial task, and may require a more complex model than what is really necessary. In comparison, DTs can explicitly deal with categorical features.

There are also known disadvantages to DT predictors. For example, they are not always robust, as they might result in poor prediction on out-of-sample data when the tree is grown too large. Hence, small trees are often desirable to avoid overfitting and also for the sake of interpretability. Assuming that for a given data distribution there exists a small DT that can achieve good accuracy, the small DTs that are computed by a typical recursive DT algorithm (such as CART [5,16]) may not achieve such accuracy, due to the heuristic nature of the algorithm. Moreover, it is usually impossible to establish a bound on the difference between the expected accuracy of the DT produced by a heuristic algorithm and the best possible DT.

Currently, popular algorithms used for constructing DTs (such as CART or C4.5) are sequential heuristics that first construct a tree and then trim (prune) it to reduce its size, see [11]. When building the tree, these heuristics use various criteria to choose a feature and a condition on that feature to branch on. As the tree is built gradually, the resulting DT is not necessarily "the best" for any particular global criterion. One recent example of this fact is the winning entry [7] in the FICO interpretable machine learning competition [8]. The authors of [7] construct a simple classifier in conjunctive normal form which in fact can also be seen as a small depth decision tree. The authors show that their classifier is both simpler and more accurate (on test data) than the trees constructed by CART.

In this paper, we aim to find *optimal* small DTs for binary classification problems that produce *interpretable* and *accurate* classifiers for the data for which such classifiers exist. We call a DT optimal if it has the best possible classification accuracy on a given training dataset. We allow complex branching rules using subsets of values of categorical features. For example, if a categorical feature represents a person's marital status and can take the values "single", "married", "divorced", "widowed", or "has domestic partner", a simple branching rule, which looks at numerical representation of the features, will make decisions based on a feature being "single" or not, while a more appropriate decision may be "either married or has a domestic partner" or not. Such combinatorial branching rules are considered desirable and in the case of binary classification using CART, branching on the best subset values of a categorical feature can be done again according to a sequential local heuristic. On the other hand, combinatorial branching may lead to overfitting when a

categorical variable can take a large number of values. If the categorical variable can take ℓ values, then, there are $2^\ell - 2$ possible subsets of values of this feature that can be used for branching. To avoid overfitting, our model allows bounding the size of the subset used for branching.

While finding an optimal DT (even without the combinatorial decisions) is known to be an NP-hard problem [10], we show that with careful modeling, the resulting integer programs can be solved to optimality in a reasonable amount of time using commercial solvers such as Cplex. Moreover, since we directly optimize the empirical loss of a DT in our model, even suboptimal feasible solutions tend to yield classifiers that outperform those learned by other DT algorithms. In particular, we consider a binary classification problem, which means that the output nodes (leaves) of our DTs generate binary output. Our problem formulation takes particular advantage of this fact. Also, while our formulation can be generalized to real-valued data, it is designed for the case when the input data is binary. Hence, we will consider input data as being a binary vector with the property that features are grouped so that only one feature can take the value 1 in each group for each data sample. Our formulation explicitly takes this structure into account as we allow branching on any subset of the values of that feature. To our knowledge such generalized rules have not been addressed by any algorithm aiming at constructing optimal trees, such as a recent method proposed in [3], which we will discuss in the next section.

In this paper, we focus on constructing small DTs with up to four levels of decisions, which makes the resulting model clearly interpretable and easily usable by humans. Our formulation, in principle, can work for binary trees of any topology; however, as we will show in our computational results, trees of more complex topologies are much more time consuming to train and require larger training sets to avoid overfitting. The purpose of this paper is to show that if an accurate small (interpretable) tree exists for a given data set, it can be obtained in a reasonable time by our proposed model, while popular heuristic methods such as C4.5 [16] and random forests [6] tend to produce less accurate and less interpretable trees. We note that even though we mostly focus on categorical features, our approach can easily handle numerical features via tresholding. We discuss how to do this later and also present numerical experiments with data sets with both categorical and numerical features.

The key approach we pursue is to formulate the DT training problem as a mixed-integer optimization problem that is specially designed to handle categorical variables. We then propose several modifications that are intended to aid a branch-and-bound solver, e.g. symmetry breaking. We also consider an extension to a formulation that directly constrains either training sensitivity or training specificity and then maximizes the other measure.

The rest of the paper is organized as follows: First, in Section 2, we discuss related work in using integer formulations for practical machine learning. Then, in Section 3, we describe the main ideas of our approach and the structure of the data for which the model is developed. In Section 4 we describe an initial IP model and several techniques for strengthening this formulation. We present some computational results and comparisons in Section 5.

2 Related Work

The idea of solving decision trees to optimality given a fixed topology is hardly new. In [5] from 1984, the authors discuss the "one-step optimality" of inductive (greedy) tree algorithms, and how one would ideally prefer an "overall optimal" method wherein the tree is learned in one step (such as the one we explore in this paper). The authors remark that this is analogous to a "best subset selection" procedure of linear regression, and continue to say that "At the current stage of computer technology, an overall optimal tree growing procedure does not appear feasible for any reasonably sized dataset". In [14], the authors detail what they call the "look-ahead pathology" of greedy tree learning algorithms, lending further evidence of possible failures of greedy one-step methods.

In the 1990s several papers considered optimization formulations for optimal decision tree learning, but deliberately relaxed the inherently integer nature of the problem. In particular, in [1], a large-scale linear optimization problem, which can be viewed as a relaxation, is solved to global optimality via a specialized tabu search method over the extreme points of the linear polytope. In [2], a similar formulation is used, but this time combined with the use of support-vector machine techniques such as generalized kernels for multivariate decisions, yielding a convex nonlinear optimization problem which admits a favorable dual structure. More recent work [15] has employed a stochastic gradient method to minimize a continuous upper bound on misclassification error made by a deep decision tree. None of these methods, though, guarantee optimal decision trees, since they do not consider the exact (integer) formulations, such as the one discussed in this paper.

Recently, in [3], an integer model for optimal decision trees has been proposed. The key difference with the model in this paper is that [3] does not target categorical variables and, hence, does not exploit the resulting combinatorial structure. Moreover, all features are treated as real-valued ones, hence a categorical feature is replaced by several binary features, and two possible models are proposed. The first uses arbitrary linear combinations of features, and, in principal, is more general than what we propose here, but results in a loss of interpretability. The second uses the value of one feature in each branching decision, and hence is less general than the model in this paper. Additionally, we focus on binary classification problems whereas [3] presents a formulation for multi-class classification. Rather than fixing a tree topology, as we do, they propose tuning a regularization parameter in the objective; as the parameter magnitude increases, more leaf nodes may have no samples routed to them, effectively yielding shallower trees. We note that this does not simplify the underlying optimization problem, and moreover requires tuning parameters in a setting where the training of models is computationally non-negligible, and the effect of the choice of regularization parameter on the tree topology cannot be known a priori. In fact, in the computational results of [3], the depth is often fixed. Finally, unlike the work in [3], we not only propose a basic model that specifically exploits the categorical nature of the features, but we also propose several modifications of the model that produce stronger formulations and improve the efficiency of the branch-and-bound solver.

We would now like to remark on other relevant uses of integer optimization in classification settings. In particular, [18] considered the problem of learning optimal "or's of and's", which fits into the problem of learning optimal disjunctive normal forms (DNFs), where optimality is measured
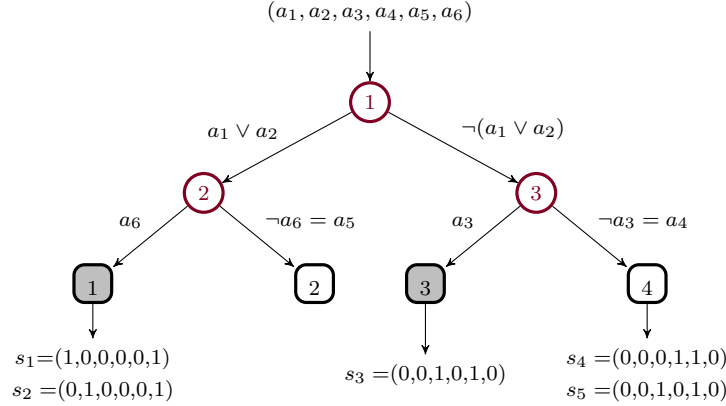
by a trade-off between the misclassification rate and the number of literals that appear in the "or of and's". The work in [18] remarks on the relationship between this problem and learning optimal decision trees. In [18], for the sake of computational efficiency, the authors ultimately resort to optimally selecting from a subset of candidate suboptimal DNFs learned by heuristic means rather than solving their proposed mixed-integer optimization problem. Similarly, [13] proposes learning DNF-like rules via integer optimization, and propose a formulation that can be viewed as boolean compressed sensing, lending theoretical credibility to solving a linear programming relaxation of their integer problem. Another integer model that minimizes misclassification error by choosing general partitions in feature space was proposed in [4], but when solving the model, global optimality certificates were not easily obtained on moderately-sized classification datasets, and the learned partition classifiers rarely outperformed CART, according to the overlapping author in [3]. Finally, a column generation based mixed-integer programming approach to construct optimal DNFs was recently proposed in [7]. This approach seems to work quite well on several binary classification datasets including the FICO challenge data [8].

3 Setting

In this paper we consider datasets of the form $\{(g_1^i, \ldots, g_t^i, y^i) : i \in 1, 2, \ldots, N\}$ where $g_j^i \in G_j$ for some finite set G_j for $j = 1, \ldots, t$, and $y^i \in \{-1, +1\}$ is the class label associated with a negative or positive class, respectively. For example, if the data is associated with a manufacturing process with t steps, then each G_j may correspond to a collection of different tools that can perform the jth step of the production process and the label may denote whether the resulting product meets certain quality standards or not. The classification problem associated with such an example is to estimate the label of a new item based on the particular different step-tool choices used in its manufacturing. Alternatively, the classification problem can involve estimating whether a student will succeed in graduating from high school based on features involving gender, race, parents marital status, zip-code and similar information.

Any (categorical) data of this form can alternatively be represented by a binary vector so that $g_j^i \in G_j$ is replaced by a unit vector of size $|G_j|$ where the only non-zero entry in this vector indicates the particular member of G_j that the data item contains. In addition, a real-valued (numerical) feature can be, when appropriate, made into a categorical one by "binning" - that is breaking up the range of the feature into segments and considering segment membership as a categorical feature. This is commonly done with features such as income or age of an individual. For example, for advertising purposes websites typically represent users by age groups such as "teens", "young adults", "middle aged", and "seniors" instead of actual age.

The non-leaf nodes in a decision tree are called the *decision nodes* where a binary test is applied to data items. Depending on the results of these tests, the data item is routed to one of the *leaf* nodes. Each leaf node is given a binary label that determines the label assigned to the data by the DT. The binary tests we consider are of the form "does the jth feature of the data item belong to set $\bar{G}_j$?", where $\bar{G}_j \subseteq G_j$. If the categorical data is represented by a binary vector, then the test

Fig. 1 A decision tree example

$$(a_1, a_2, a_3, a_4, a_5, a_6)$$

1

$a_1 \vee a_2$ $\neg(a_1 \vee a_2)$

2   3

a_6 $\neg a_6 = a_5$ a_3 $\neg a_3 = a_4$

1   2   3   4

$s_1 = (1,0,0,0,0,1)$ $s_4 = (0,0,0,1,1,0)$

$s_2 = (0,1,0,0,0,1)$ $s_3 = (0,0,1,0,1,0)$ $s_5 = (0,0,1,0,1,0)$

becomes checking if at least one of the indices from a given collection contains a 1 or not. We do not consider more general tests that might check different conditions on multiple features.

As a concrete example, consider the tree in Figure 3 applied to binary vectors $a \in \{0, 1\}^6$ whose elements are divided into two groups: $\{a_1, a_2, a_3, a_4\}$ and $\{a_5, a_6\}$ corresponding to two categorical features in the original data representation. The branching decision at node 1 (the root), is based on whether one of a_1 or a_2 is equal to 1. If true, a given data sample is routed to the left, otherwise (that is, if both a_1 and a_2 are 0), the sample is routed to the right. The branching at nodes 2 and 3 (the two children of node 1) are analogous and are shown in the picture. We can now see that data samples $s^1 = (1, 0, 0, 0, 0, 1)$ and $s^2 = (0, 1, 0, 0, 0, 1)$ are routed to leaf node 1, sample $s^3 = (0, 0, 1, 0, 1, 0)$ is routed to leaf node 3, and samples $s^4 = (0, 0, 0, 1, 1, 0)$ and $s^5 = (0, 0, 1, 0, 1, 0)$ are routed to leaf node 4. The labels of the leaf nodes are denoted by the colors white and gray in Figure 3.

Formally, a DT is defined by (*i*) the topology of the tree, (*ii*) binary tests applied at each decision node, and, (*iii*) labels assigned to each leaf node. Throughout the paper we consider tree topologies where a decision node either has two leaf nodes or else has two other decision nodes as children. Note that decision trees defined this way are inherently symmetric objects, in the sense that the same DT can be produced by different numberings of the decision and leaf nodes as well as different labeling of the leaf nodes and the binary tests applied at the decision nodes. For example, reversing the binary test from (a_6) to $(\neg a_6)$ in decision node 2, and at the same time flipping the labels of the leaf nodes 1 and 2, results in an identical DT. More generally, it is possible to reverse the binary test at any decision node and "flip" the subtrees rooted at that node to obtain the same tree.

The optimization problem we consider in the next section starts with a given tree topology and finds the best binary tests (and labels for the leaf nodes) to classify the test data at hand with minimum error. Due to the symmetry discussed above, we can fix the labeling of the leaf nodes at the beginning of the process and the problem reduces to finding the best binary tests, or equivalently, choosing a categorical feature and a subset of its realizations at each decision node. Therefore, the optimization problem consists of assigning a binary test to each decision node so

as to maximize the number of correctly classified samples in the *training set*. We say that the classification of the ith sample is *correct* provided the path the ith sample takes through the tree starting from the root node ends at a leaf corresponding to the correct label. The ultimate goal of the process, however, is to obtain a DT that will classify new data well, i.e., we are actually concerned with the generalization ability of the resulting DT.

Notice that given two tree topologies such that one is a minor of the other (i.e. it can be obtained from the other by deleting nodes and contracting edges), the larger tree would always be able to classify at least as many samples correctly as the smaller one on the training data. Consequently, for optimization purposes, larger trees always perform better than any of its minors. However, larger trees generally result in more computationally challenging optimization problems. In addition, smaller trees are often more desirable for classification purposes as they are more robust and are easier to interpret.

4 Integer Programming Formulation

In this section, we first present the basic integer programming formulation and then describe some enhancements to improve its computational efficiency. We initially assume that the topology of the binary tree is given (see Figure 2) and therefore the number of decision and leaf nodes as well as how these nodes are connected is known. We will then describe how to pick a good topology. The formulation below models how the partitioning of the samples is done at the decision nodes, and which leaf node each sample is routed to as a result.

We begin by introducing the notation. Let the set of all samples be indexed by $I = \{1, 2, \ldots, |I|\}$, let $I_+ \subset I$ denote the indices of samples with positive labels and let $I_- = I \setminus I_+$ denote the indices of the samples with negative labels. Henceforth, we assume that for each sample the input data is transformed into a binary vector where each categorical feature is represented by a unit vector that indicates the realization of the categorical feature. With some abuse of terminology, we will now refer to the entries of this binary vector as "features", and the collection of these 0/1 features that are associated with the same categorical feature as "groups". Let the set of groups be indexed by $G = \{1, 2, \ldots, |G|\}$ and the set of the 0/1 features be indexed by $J = \{1, 2, \ldots, |J|\}$. In addition, let $J(g)$ denote the set of features that are contained in group g. In the example associated with Figure 3 above, we have $G = \{1, 2\}$, $J = \{1, 2, 3, 4, 5, 6\}$, and $J(1) = \{1, 2, 3, 4\}$, $J(2) = \{5, 6\}$. For sample i, we denote the value of its jth feature by a_j^i.

Let the set of decision nodes be indexed by $K = \{1, 2, \ldots, |K|\}$ and the set of leaf nodes be indexed by $B = \{1, 2, \ldots, |B|\}$. We denote the indices of leaf nodes with positive labels by $B_+ \subset B$ and the indices of leaf nodes with negative labels by $B_- = B \setminus B_+$. For convenience, we let B_+ contain even indices, and B_- contain the odd ones.

4.1 The basic formulation

We now describe our key decision variables and the constraints on these variables. We use binary variables $v_g^k \in \{0, 1\}$ for $g \in G$ and $k \in K$ to denote if group g is selected for branching at decision node k. As discussed in Section 3, exactly one group has to be selected for branching at a decision node; consequently, we have the following set of constraints:

$$\sum_{g \in G} v_g^k = 1 \quad \forall k \in K. \tag{1}$$

The second set of binary variables $z_j^k \in \{0, 1\}$ for $j \in J$ and $k \in K$ are used to denote if feature j is one of the selected features for branching at a decision node k. Clearly, feature $j \in J$ can be selected only if the group containing it is selected at that node. Therefore, we have the following set of constraints:

$$z_j^k \le v_g^k \quad \forall k \in K, \ \forall g \in G, \ \forall j \in J(g) \tag{2}$$

in the formulation. Without loss of generality, we use the convention that if a sample has one of the selected features at a given node, it follows the left branch at that node; otherwise it follows the right branch.

Let

$$S = \left\{ (v, z) \in \{0, 1\}^{|K| \times |G|} \times \{0, 1\}^{|K| \times |J|} : \quad (v, z) \text{ satisfies inequalities (1) and (2)} \right\},$$

and note that for any $(v, z) \in S$ one can construct a corresponding decision tree in a unique way and vice versa. In other words, for any given $(v, z) \in S$ one can easily decide which leaf node each sample is routed to. We next describe how to relate these variables (and therefore the corresponding decision tree) to the samples.
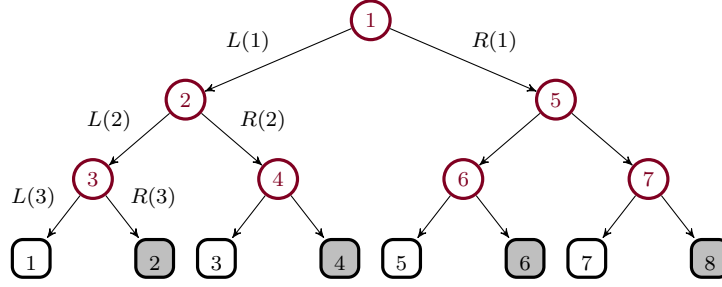
We use binary variables $c_b^i \in \{0, 1\}$ for $b \in B$ and $i \in I$ to denote if sample i is routed to leaf node b. This means that variable c_b^i should take the value 1 only when sample i exactly follows the unique path in the decision tree that leads to leaf node b. With this in mind, we define the expression

$$L(i, k) = \sum_{j \in J} a_j^i z_j^k \quad \forall k \in K, \ \forall i \in I, \tag{3}$$

and make the following observation:

Proposition 1 Let $(z, v) \in S$. Then, for all $i \in I$ and $k \in K$ we have $L(i, k) \in \{0, 1\}$. Furthermore, $L(i, k) = 1$ if and only if there exists some $j \in J$ such that $a_j^i = 1$ and $z_j^k = 1$.

Proof For any $(z, v) \in S$ and $k \in K$, exactly one of the v_g^k variables, say $v_{g'}^k$, takes value 1 and $v_g^k = 0$ for all $g \ne g'$. Therefore, $z_j^k = 0$ for all $j \notin J(g)$. Consequently, the first part of the claim follows for all $i \in I$ as $L(i, k) = \sum_{j \in J} a_j^i z_j^k = \sum_{j \in J(g')} a_j^i z_j^k = z_{j_i}^k \in \{0, 1\}$ where $j_i \in J(g')$ is the index of the unique feature for which $a_{j_i}^i = 1$. In addition, $L(i, k) = 1$ if and only if $z_{j_i}^k = 1$ which proves the second part of the claim.   ∎

Fig. 2 A balanced depth-3 tree

Consequently, the expression $L(i,k)$ indicates if sample $i \in I$ branches left at node $k \in K$. Similarly, we define the expression

$$R(i,k) = 1 - L(i,k) \quad \forall k \in K, \; \forall i \in I, \tag{4}$$

to indicate if sample i branches right at node k.

To complete the model, we relate the expressions $L(i,k)$ and $R(i,k)$ to the c_b^i variables. Given that the topology of the tree is fixed, there is a unique path leading to each leaf node $b \in B$ from the root of the tree. This path visits a subset of the nodes $K(b) \subset K$ and for each $k \in K(b)$ either the left branch or the right branch is followed. Let $K^L(b) \subseteq K(b)$ denote the decision nodes where the left branch is followed to reach leaf node b and let $K^R(b) = K(b) \setminus K^L(b)$ denote the decision nodes where the right branch is followed. Sample i is routed to b only if it satisfies all the conditions at the nodes leading to that leaf node. Consequently, we define the constraints

$$c_b^i \leq L(i,k) \quad \forall b \in B, \; \forall i \in I, \; \forall k \in K^L(b), \tag{5}$$

$$c_b^i \leq R(i,k) \quad \forall b \in B, \; \forall i \in I, \; \forall k \in K^R(b), \tag{6}$$

for all $i \in I$ and $b \in B$. Combining these with the equations

$$\sum_{b \in B} c_b^i = 1 \quad \forall i \in I \tag{7}$$

gives a complete formulation. Let

$$Q(z,v) = \left\{ c \in \{0,1\}^{N \times |B|} : \text{ such that (5)-(7) hold} \right\}.$$

We next formally show that combining the constraints in S and $Q(z,v)$ gives a correct formulation.

Proposition 2 *Let $(z,v) \in S$, and let $c \in Q(z,v)$. Then, $c_b^i \in \{0,1\}$ for all $i \in I$ and $b \in B$. Furthermore, if $c_b^i = 1$ for some $i \in I$ and $b \in B$, then sample i is routed to leaf node b.*

Proof Given $(z,v) \in S$ and $i \in I$, assume that the correct leaf node sample i should be routed to in the decision tree defined by (z,v) is the leaf node b'. For all other leaf nodes $b \in B \setminus \{b'\}$, sample i either has $L(i,k) = 0$ for some $k \in K^L(b)$ or $R(i,k) = 0$ for some $k \in K^R(b)$. Consequently, $c_b^i = 0$ for all $b \neq b'$. Equation (7) then implies that $c_{b'}^i = 1$ and therefore $c_b^i \in \{0,1\}$ for all $b \in B$. Conversely, if $c_{b'}^i = 1$ for some $b' \in B$, then $L(i,k) = 1$ for all $k \in K^L(b)$ and $R(i,k) = 1$ for all $k \in K^R(b)$. ∎

We therefore have the following integer programming (IP) formulation:

$$\max \quad \sum_{i \in I_+} \sum_{b \in B_+} c_b^i + C \sum_{i \in I_-} \sum_{b \in B_-} c_b^i \tag{8a}$$

$$\text{s. t.} \quad (z, v) \in S \tag{8b}$$

$$c \in Q(z, v) \tag{8c}$$

where C in the objective (8a) is a constant weight chosen in case of class imbalance. For instance, if a training set has twice as many good examples as bad examples, it may be worth considering setting $C = 2$, so that every correct classification of a bad data point is equal to two correct classifications of good data points.

Notice that formulation (8) allows solutions where all samples follow the same branch. For example, it is possible to have a solution where a branching variable $v_g^k = 1$ for some $k \in K$ and $g \in G$, and at the same time $z_j^k = 0$ for all $j \in J(g)$. In this case $L(i, k) = 0$ for all $i \in I$ and all samples follow the right branch. It is possible to exclude such solutions using the following pair of constraints:
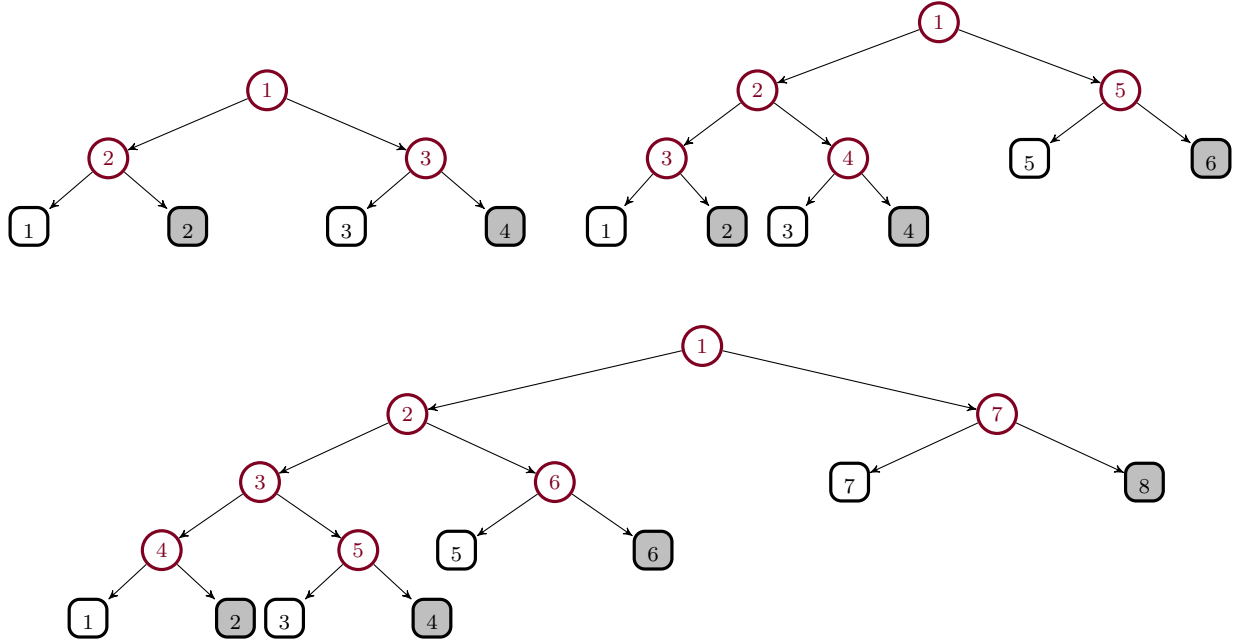
$$(|J(g)| - 1)v_g^k \ \geq \ \sum_{j \in J(g)} z_j^k \ \geq \ v_g^k, \tag{9}$$

for all $k \in K$ and $g \in G$. These constraints enforce that if a group is selected for branching, then at least one, but not all, of its features should be selected. We should note that in our experiments we have not seen any benefit from using these inequalities and decided not to include them in the formulation.

4.2 Choosing the tree topology

The IP model (8) finds the optimal decision tree for a given tree topology which is an input to the model. It is possible to build a more complicated IP model that can also build the tree topology (within some restricted class) but for computational efficiency, we decided against it. Instead, for a given dataset, we use several fixed candidate topologies and build a different DTs for each one of them. We then pick the most promising one using cross-validation. The four tree topologies we use are the balanced depth-3 tree shown in Figure 2 and the additional trees shown in Figure 3.

Note that the first two trees presented in Figure 3 can be obtained as a minor of the balanced depth-3 tree shown in Figure 2 and therefore, the optimal value of the model using the balanced depth-3 tree will be at least as good as that of the smaller trees. Similarly, these two trees can also be obtained as a subtree of the last tree in Figure 3. However, due to possible overfitting, the larger trees might perform worse than the smaller ones on new data (in testing). As we will show via computational experiments, training smaller trees take fraction of the time compared to training larger trees, hence training a collections of trees of increasing topologies is comparable to training one large tree.

Fig. 3 Possible tree topologies

4.3 Computational tractability

While (8) is a correct formulation, it can be improved to enhance computational performance. We next discuss some ideas that help reduce the size of the problem, break symmetry and strengthen the linear programming relaxation. We first observe that the LP relaxation of (8), presented explicitly below, is rather weak.

$$
\max \quad \sum_{i \in I_+} \sum_{b \in B_+} c_b^i + C \sum_{i \in I_-} \sum_{b \in B_-} c_b^i
$$

$$
\begin{aligned}
\text{s. t.} \quad \sum_{g \in G} v_g^k &= 1 \quad \forall k \in K, \\
z_j^k &\le v_g^k \quad \forall k \in K, \ \forall g \in G, \ \forall j \in J(g), \\
c_b^i &\le L(i,k) \quad \forall b \in B, \ \forall i \in I, \ \forall k \in K^L(b), \\
c_b^i &\le R(i,k) \quad \forall b \in B, \ \forall i \in I, \ \forall k \in K^R(b), \\
\sum_{b \in B} c_b^i &= 1 \quad \forall i \in I \\
c, \ v, \ z &\ge 0.
\end{aligned}
$$

Note that we do not need an explicit upper bound of 1 on the variables as it is implied by other constraints. Also note that as $\sum_{b \in B} c_b^i \le 1$, for all $i \in I$, the optimal value of the LP relaxation is at most $|I_+| + C|I_-|$. Assuming that the decision tree has at least two levels, we will next construct a solution to the LP that attains this bound. Moreover, this solution would also satisfy $v_g^k \in \{0, 1\}$ for all $k \in K$ and $g \in G$.

As the decision tree has at least two levels, both the left and right branches of the root node contain a leaf node in B_+ as well as a leaf node in B_-. Let $b_-^L, b_-^R \in B_-$ and $b_+^L, b_+^R \in B_+$ where b_-^L and b_+^L belong to the left branch and b_-^R and b_+^R belong to the right branch. For an arbitrary $\bar{g} \in G$, we construct the solution (z, v, c) as follows: First we set $v_{\bar{g}}^k = 1$ for all $k \in K$ and $z_j^k = 1/2$ for all $k \in K$ and $j \in J(\bar{g})$. We then set $c_b^i = 1/2$ for $b \in \{b_+^L, b_+^R\}$ for all $i \in I_+$ and set $c_b^i = 1/2$ for $b \in \{b_-^L, b_-^R\}$ for all $i \in I_-$. We set all the remaining variables to zero. Notice that $\sum_{b \in B_-} c_b^i = 1$ for $i \in I_-$ and $\sum_{b \in B_+} c_b^i = 1$ for $i \in I_+$ and therefore the value of this solution is indeed $|I_+| + C|I_-|$. To see that the this solution is feasible for the LP relaxation of (8), first note that $\sum_{g \in G} v_g^k = 1$ for all $k \in K$ and $z_j^k \le v_g^k$ for all $j \in J(g)$, $g \in G$, and $k \in K$. Also notice that $L(i,k) = R(i,k) = 1/2$ for all $i \in I$ and $k \in K$, which implies that (11) and (12) are also satisfied for all $i \in I$ and $k \in K$.

4.3.1 Relaxing some binary variables

The computational difficulty of a MILP typically increases with the number of integer variables in the formulation and therefore it is desirable to impose integrality on as few variables as possible. We next show that all of the v variables and most of the z variables take value $\{0, 1\}$ in an optimal solution even when they are not explicitly constrained to be integral.

Proposition 3 *Every extreme point solution to (8) is integral even if (i) variables v_g^k are not declared integral for all $g \in G$ and decision nodes $k \in K$, and, (ii) variables z_j^k are not declared integral for $j \in J$ and decision nodes $k \in K$ that are adjacent to a leaf node.*

Proof Assume the claim does not hold and let $\bar{p} = (\bar{v}, \bar{z}, \bar{c})$ be an extreme point solution that is fractional. Let $K^L \subset K$ denote the decision nodes that are adjacent to leaf nodes and consider node $a \notin K^L$. First note that if $\bar{v}_b^a$ is fractional, that is, if $1 > \bar{v}_b^a > 0$ for some feature group $b \in G$, then $1 > \bar{v}_g^a$ for all groups $g \in G$ as $\sum_{g \in G} \bar{v}_g^a = 1$. Consequently, for this decision node we have all $\bar{z}_j^a = 0$ as $\bar{z}_j^a \in \{0, 1\}$ for $j \in J$. This also implies that $L(i, a) = 0$ for all $i \in I$. In this case, for any $g \in G$, the point $\bar{p}$ can be perturbed by setting the v_g^a variable to 1 and setting the remaining v_*^a variables to 0 to obtain a point that satisfies the remaining constraints. A convex combination of these perturbed points (with weights equal to $\bar{v}_g^a$) gives the point $\bar{p}$, a contradiction. Therefore all $\bar{v}_g^k$ are integral for $g \in G$ and $k \in K \setminus K^L$.

Therefore, if $\bar{p}$ is fractional, then at least one of the following must hold: either (i) $1 > \bar{v}_g^k > 0$ for some $k \in K^L$ and $g \in G$, or, (ii) $1 > \bar{z}_j^k > 0$ for some $k \in K^L$ and $j \in J$, or, (iii) $1 > c_b^i > 0$ for some $b \in B$ and $i \in I$. As all these variables are associated with some decision node $k \in K^L$, we conclude that there exists a decision node $a \in K^L$ for which either $1 > \bar{v}_g^a > 0$ for some $g \in G$, or, $1 > \bar{z}_j^a > 0$ for some $j \in J$, or, $1 > c_b^i > 0$ for some $i \in I$ and $b \in \{b_+, b_-\}$ where $b_+ \in B_+$ and $b_- \in B_-$ are the two leaf nodes attached to decision node a on the left branch and on the right branch, respectively.

Let I_a^+ denote the set of samples in I^+ such that $\bar{c}_{b_+}^i > 0$ and similarly, let I_a^- denote the set of samples in I^- such that $\bar{c}_{b_-}^i > 0$. If $\bar{c}_{b_+}^i \ne L(i, a)$, for some $i \in I_a^+$, then point $\bar{p}$ can be perturbed by increasing and decreasing $\bar{c}_{b_+}^i$ to obtain two new points that contain $\bar{p}$ in their convex hull, a contradiction. Note that $L(i, k) \in \{0, 1\}$ for all $i \in I$ and $k \in K \setminus K^L$ and therefore these two

points indeed satisfy all the constraints. Consequently, we conclude that $\bar{c}_{b_+}^i = L(i,a)$ for all $i \in I_a^+$. Similarly, $\bar{c}_{b_-}^i = 1 - L(i,a)$ for all $i \in I_a^-$. Notice that this observation also implies that, if $\bar{c}_{b_+}^i$ is fractional for some $i \in I_a^+$ or $\bar{c}_{b_-}^i$ is fractional for some $i \in I_a^-$, then $L(i,a)$ is also fractional, which in turn implies that for some feature $h \in J$ we must have $\bar{z}_h^a > 0$ fractional as well.

Now assume there exists a feature $h \in J(g)$ such that $v_g^a > \bar{z}_h^a > 0$. In this case increasing and decreasing $\bar{z}_h^a$ by a small amount and simultaneously updating the values of $\bar{c}_{b_+}^i$ for $i \in I_a^+$ and $\bar{c}_{b_-}^i$ for $i \in I_a^-$ to satisfy $\bar{c}_{b_+}^i = L(i,a)$ and $\bar{c}_{b_-}^i = 1 - L(i,a)$ after the update, leads to two new points that contain $\bar{p}$ in their convex hull. Therefore, we conclude that $\bar{z}_h^a$ is either zero, or $\bar{z}_h^a = \bar{v}_g^a$.

So far, we have established that if $\bar{c}_b^i$ is fractional for some $i \in I_a^- \cup I_a^+$ and $b \in \{b_+, b_-\}$, then there is a fractional $\bar{z}_j^a$ variable for some feature $j \in J$. In addition, we observed that if there is a fractional $\bar{z}_j^a$ for some $j \in J$ then there is a fractional $\bar{v}_g^a$ for some $g \in G$. Therefore, if $\bar{p}$ is not integral, there exists a feature group $d \in G$ such that $1 > \bar{v}_d^a > 0$. As $\sum_{g \in G} \bar{v}_g^a = 1$, this implies that there also exists a different group $e \in G \setminus \{d\}$ such that $1 > \bar{v}_e^a > 0$.

We can now construct two new points that contain $\bar{p}$ in their convex hull as follows: For the first point we increase $\bar{v}_d^a$ and decrease $\bar{v}_e^a$ by a small amount and for the second point we do the opposite perturbation. In addition, for both points we first update the values of $\bar{z}_j^a$ for all $j \in J(d) \cup J(e)$ and $\bar{z}_j^a > 0$ so that $\bar{z}_j^a = \bar{v}_d^a$ for all $j \in J(d)$ and $\bar{z}_j^a = \bar{v}_e^a$ for all $j \in J(e)$. Finally, we perturb the associated $\bar{c}_b^i$ variables for $i \in I_a^- \cup I_a^+$ and $b \in \{b_+, b_-\}$ so that $\bar{c}_{b_+}^i = L(i,a)$, for $i \in I_a^+$, and $\bar{c}_{b_-}^i = 1 - L(i,a)$ for all $i \in I_a^-$. Both points are feasible and therefore we can conclude that $\bar{p}$ is not an extreme point, which is a contradiction. Hence $\bar{p}$ cannot be fractional. ∎

We have therefore established that the v variables do not need to be declared integral and the only z variables that need to be declared integral in the formulation (8) are the feature selection variables z_j^k for all features $j \in J$ and decision nodes $k \in K$ that are not adjacent to a leaf node.

4.3.2 Deleting unnecessary variables

Notice that the objective function (8a) uses variables c_b^i only if it corresponds to a correct classification of the sample (i.e., $i \in I_+$ and $b \in B_+$, or $i \in I_-$ and $b \in B_-$). Consequently, the remaining c_b^i variables can be projected out of the formulation without changing the value of the optimal solution. We therefore only define c_b^i variables for

$$\left\{ (i,b) : i \in I_+, b \in B_+, \text{ or, } i \in I_-, b \in B_- \right\} \tag{10}$$

and write constraints (5) and (6) for these variables only. In addition, This reduces the number of c variables and the associated constraints in the formulation by a factor of one half. In this projected formulation equation (7) becomes

$$\sum_{b \in B^+} c_b^i \leq 1 \text{ for } i \in I_+ \text{ and } \sum_{b \in B^-} c_b^i \leq 1 \text{ for } i \in I_-$$

4.3.3 Relaxing more binary variables

Also note that the objective function (8a) is maximizing a (weighted) sum of c_b^i variables and the only constraints that restrict the values of these variables are inequalities (5), (6) and (7) which all have a right hand side of 0 or 1. Consequently, replacing the integrality constraints $c_b^i \in \{0,1\}$ with simple bound constraints $1 \geq c_b^i \geq 0$, still yields optimal solutions that satisfy $c_b^i \in \{0,1\}$. Hence, we do not require c_b^i to be integral in the formulation and therefore significantly reduce the number of integer variables. Thus, we have a formulation for training optimal decision trees, where the number of integer variables is *independent* of the number of samples.

4.3.4 Strengthening the model

We next present valid inequalities for (8) that can be used to strengthen its LP relaxation. Consider inequalities (5)

$$c_b^i \leq L(i,k)$$

for $i \in I$, $b \in B$ and $k \in K^L(b)$ where $K^L(b)$ denotes the decision nodes where the left branch is followed to reach the leaf node b. Also remember that $\sum_{b \in B} c_b^i = 1$ for $i \in I$ due to equation (7).

Now consider a fixed $i \in I$ and $k \in K$. If $L(i,k) = 0$, then $c_b^i = 0$ for all b such that $k \in K^L(b)$. On the other hand, if $L(i,k) = 1$ then at most one $c_b^i = 1$ for b such that $k \in K^L(b)$. Therefore,

$$\sum_{b \in B: K^L(b) \ni k} c_b^i \leq L(i,k) \tag{11}$$

is a valid inequality for all $i \in I$ and $k \in K$. While this inequality is satisfied by all integral solutions to the set $Q(z,v)$, it is violated by some of the solutions to its continuous relaxation. We replace the inequalities (5) in the formulation with (11) to obtain a tighter formulation. We also replace inequalities (6) in the formulation with the following valid inequality:

$$\sum_{b \in B: K^R(b) \ni k} c_b^i \leq R(i,k) \tag{12}$$

for all $i \in I$ and $k \in K$. Note that, by definition, $L(i,k) + R(i,k) = 1$ for all $i \in I$ and $k \in K$, and consequently, adding inequalities (11) and (12) for the root node of the decision tree implies that $\sum_{b \in B} c_b^i \leq 1$ for all $i \in I$.

When using inequalities (11) and (12) in the projected formulation described in Section 4.3.2, we write these inequalities with the c_b^i variables whose indices are contained in the set described in (10). Moreover, in this case adding the inequalities associated with the root node of the decision tree yields $\sum_{b \in B^+} c_b^i \leq 1$ for $i \in I_+$ and $\sum_{b \in B^-} c_b^i \leq 1$ for $i \in I_-$. Therefore, the projected version of (7) described in Section 4.3.2 becomes redundant.

4.3.5 Breaking symmetry: Anchor features

If the variables of an integer program can be permuted without changing the structure of the problem, the integer program is called *symmetric*. This poses a problem for MILP solvers (such as

Cplex) since the search space increases exponentially, see Margot (2009). The formulation (8) falls into this category as there may be multiple alternative solutions that represent the same decision tree. In particular, as we have discussed earlier in the paper, we consider a decision node that is not adjacent to leaf nodes and assume that the subtrees associated with the left and right branches of this node are symmetric (i.e. they have the same topology). In this case, if the branching condition is reversed at this decision node (in the sense that the values of the v variables associated with the chosen group are flipped), and, at the same time, the subtrees associated with the left and right branches of this node are switched, one obtains an alternative solution to the formulation corresponding to the same decision tree. To avoid this, we designate one particular feature $j(g) \in J(g)$ of each group $g \in G$ to be the *anchor feature* of that group and enforce that if a group is selected for branching at such a node, samples with the anchor feature follow the left branch. More precisely, we turn one of the inequalities in (2) to an equation and add the following to the formulation:

$$z^k_{j(g)} = v^k_g \tag{13}$$

for all $g \in G$, and all $k \in K$ that is not adjacent to a leaf node and has symmetric subtrees hanging on the right and left branches. While equations (13) lead to better computational performance, they do not exclude any decision trees from the feasible set of solutions.

4.4 Controlling overfitting due to combinatorial branching

As mentioned earlier, combinatorial branching may lead to overfitting when $|J(g)|$ is large for a categorical feature $g \in G$ as there are $2^{|J(g)|}$ possible ways to branch using this feature. To avoid overfitting, we require the size of the subset used for branching to be either at most $max.card$ or at least $(|J(g)| - max.card)$ for some input parameter $max.card$. To this end, for each node $k \in K$ and for each group $g \in G$ that corresponds to a categorical feature with $|J(g)| > max.card$, we create an additional variable x^k_g and include the following constraints in the formulation,

$$\sum_{j \in J(g)} z^k_j \leq max.card + (|J(g)| - max.card)(1 - x^k_g)$$

$$\sum_{j \in J(g)} z^k_j \geq (|J(g)| - max.card) - (|J(g)| - max.card)x^k_g$$

$$x^k_g \in \{0, 1\}.$$

We note that these new variables can also be used to break symmetry in the problem. Instead of using anchor features, one can simply set all x^k_g variables to 1 for $g \in G$ whenever $k \in K$ is not adjacent to a leaf node and has symmetric subtrees hanging on the right and left branches. Similar to using anchor features, this restriction would exclude one of the solutions obtained by reversing the branching condition at a decision node (i.e. flipping the values of the v variables associated with the chosen group), and, switching the subtrees associated with the left and right branches of this node.

4.5 Handling numerical features

To handle numerical features, we simply turn them into categorical features by binning them into intervals using deciles as thresholds. Consequently, each numerical feature becomes a categorical feature with (up to) 10 possible values, depending on the decile it belongs to. Therefore, one can use the model described above without any further changes. However, this might lead to decision trees that branch on, for example, whether or not a numerical feature belongs to the second or seventh quantiles, which of course is not a very interpretable condition. It is therefore desirable to branch on these features in a way that captures their ordinal nature. To this end, we add additional constraints for these features to ensure that the branching decisions correspond to "less than or equal to" or "greater than or equal to" conditions.

More precisely, for each node $k \in K$ and for each group $g \in G$ that corresponds to a numerical feature, we create an additional variable w_g^k to denote if the branching condition is of "greater than or equal to" or "less than or equal to" form. We then require the associated z_j^k variables for $j \in J(g)$ to take either increasing (when $w_g^k = 1$) or decreasing values (when $w_g^k = 0$). The additional constraints are,

$$
\begin{aligned}
z_j^k &\geq z_{j+1}^k - w_g^k & \forall j, j+1 \in J(g) \\
z_j^k &\geq z_{j-1}^k - (1 - w_g^k) & \forall j, j-1 \in J(g) \\
w_g^k &\in \{0, 1\}.
\end{aligned}
$$

We note that it is possible to enforce "less than or equal to" or "greater than or equal to" form without using the additional variables w, by binarizing numerical features differently, see [7, 19]. However in this case the LP formulation becomes more dense and overall solution times are significantly slower.

We also note that an alternative way to break symmetry in this case is to set all w_g^k variables to 1 for $g \in G$ (without loss of generality) whenever $k \in K$ is not adjacent to a leaf node and has symmetric subtrees hanging on the right and left branches. For balanced trees this property is satisfied for all non-leaf nodes. Fixing w variables this way enforces that the left branch of a decision node will check if the greater than or equal to condition holds for the associated numerical feature. Clearly, if this symmetry breaking rule is used, one should not use anchor features described in Section 4.3.5.

4.6 Maximizing sensitivity/specificity

In many practical applications, especially those involving imbalanced datasets, the user's goal is to maximize sensitivity (the true positive rate, or TPR), while guaranteeing a certain level of specificity (the true negative rate, or TNR), or vice versa, instead of optimizing the total accuracy. While such problems cannot be addressed with heuristics such as CART (except by a trial-and-error approach to reweighting samples), our model (8) readily lends itself to such a modified task. For example, if we intend to train a classifier with a guaranteed specificity (on the training set) of 0.95, then we

simply add the following constraint to (8)

$$\sum_{i \in I_-} \sum_{b \in B_-} c_b^i \geq \lceil (1 - 0.95)|I_-| \rceil \tag{14}$$

and change the objective function (8a) to

$$\sum_{i \in I_+} \sum_{b \in B_+} c_b^i. \tag{15}$$

Likewise, we can produce a model that maximizes specificity while guaranteeing a certain level of sensitivity by switching the expressions in the constraint (14) and objective (15).

5 Computational Results

We now turn to computational experiments for which we used a collection of 10 binary (two-class) classification datasets. We obtained two of these datasets (*a1a* and *breast-cancer-wisconsin*) from LIBSVM [9], one from FICO Explainable Machine Learning Challenge [8] and the remaining 7 from the UCI Machine Learning repository [12]. These datasets were selected because they fit into our framework as the majority of their variables are either binary or categorical. Each dataset was preprocessed to have the binary form assumed by the formulation, with identified groups of binary variables. A summary description of the problems is given in Table 1.

Table 1 Summary description of the datasets

dataset	# Samples	% Positive	# Features	# Groups
a1a	1605	25%	122	14
breast-cancer-wisconsin (bc)	695	65%	90	9
chess-endgame (krkp)	3196	52%	73	36
mushrooms (mush)	8124	52%	111	20
tic-tac-toe-endgame (ttt)	958	65%	27	9
monks-problems-1 (monks-1)	432	50%	17	6
congressional-voting-records (votes)	435	61%	48	16
spect-heart (heart)	267	79%	44	22
student-alcohol-consumption (student)	395	67%	109	31
FICO Explainable ML Challenge (heloc)	9871	48%	253	23

Each dataset/tree topology pair results in a MILP instance, which we implemented in Python 2.7 and then solved with Cplex version 12.6.1 on a computational cluster, giving each instance access to 8 cores of an AMD Opteron 2.0 GHz processor. Throughout this section, we will refer to our method as ODT (Optimal Decision Trees).

5.1 Tuning the IP model

We begin with some computational tests to illustrate the benefit of various improvements to the IP model that were discussed in §4.3. We only show results for five of the datasets: *a1a, bc, krkp,*

mush and *ttt*, since for the other datasets, aside from *heloc*, the IP is solved quickly and the effect of improvements is less notable, while for *heloc* the time limit was reached in all cases.

We note that the deletion of unnecessary variables discussed in §4.3.2 seems to be performed automatically by Cplex in preprocessing, and so we do not report results relevant to this modeling choice. However, we experiment with anchoring adding (13) (§4.3.5), relaxing appropriate z variables and c variables (§4.3.1 and §4.3.1), and strengthening the model using additional constraints (11) and (12) (§4.3.4). In particular, we compare the model where none of the above techniques are applied and using the formulation (8) (Nothing), only relaxation and strengthening (11) and (12) are applied (No Anchor), only anchoring (13) and strengthening (11) and (12) are applied (No Relax), only anchoring (13) and relaxation are applied (No Strength) and finally when all of the techniques are applied (All).

In Table 2 we show the results for symmetric DTs of depths 3, while using reduced datasets of 200 randomly subsampled data instances. In each column we list the total time in seconds it took Cplex to close the optimality gap to below the default tolerance and the total number of LPs solved in the process. In the case when Cplex exceeded 3 hours, the solve is terminated and a "*" is reported instead of the time.

Table 2 IP Strengthening for depth-3 with 200 samples - each table entry represents # seconds/number of LPs solved

Dataset	Nothing	No Anchor	No Relax	No Strength	All
a1a	*/2443792	*/2422165	*/5660954	2670/598733	3098/1157891
bc	2193/50075	405/118193	139/52375	188/18121	44/18660
krkp	5377 /2766623	392/95623	3726/2702709	1434/291221	320/131274
mush	31/26	22/20	12/65	22/26	23/49
ttt	1837/1914999	346/169235	71/63109	175/28588	31/10737
monks-1	32/6904	8/1596	7/2997	14/1165	5/988
votes	293/53430	99/37350	92/29934	199/26077	96/22971
heart	423/71498	199/42365	404/253792	898/62794	329/56847
student	*/666388	*/785314	*/1290360	*/406928	*/426357
heloc	*/347971	*/77376	*/187537	*/99320	*/281425

As we see from Table 2, the data set with 200 data points make the IP difficult to solve for some data sets, such as *a1a*, *student* and *heloc* but is easy to some others, such as *bc* and *mush*. Hence in Table 3 we show results for various sizes of data, selected so that the corresponding IP is not trivial but is still solvable within three hours.

We can conclude from Tables 2 and 3 that our proposed strategies provide significant improvement in terms of computational time. In some cases, turning off an option may outperform using all options; for example, turning off variable strengthening improves computational time for *a1a* and *mush* compared to the *All* option in Table 2 and for *krkp* and *student* in Table 3 However, the *All* option consistently dominates other options in the majority of the cases, hence we conclude that using all proposed improvements is the best overall strategy.

Next we show the dependence of computational time on the tree topology and the size of the data set. In Table 4 we report these results for the *krkp*, *a1a*, and *bc* data set each averaged over five runs with random sample selection. Here, by depth-2.5 we refer to the topology shown in the upper right

Table 3 IP Strengthening for depth-3 with varying samples - each table entry represents # seconds/number of LPs solved

Dataset	Samples	Nothing	No Anchor	No Relax.	No Strength	All
a1a	100	7262/2555737	2541 / 1584533	503/426853	1352 /840813	170/104504
bc	300	7766/1013135	5445/981711	223/64411	386/32262	349/53194
krkp	400	*/559764	6984/847235	7533/1289615	2936 /97214	3693/719622
mush	500	151/37	41/0	55/1109	182/215	38/7
ttt	300	1394/404553	946/226864	424/88755	253/29869	35/12154
monks-1	600	397/32176	21/6248	15/2639	44/3198	18/2616
votes	600	9176/347632	959/109632	1373/181187	877/40894	283/47520
heart	600	1204/104583	558/66056	231/44498	1002/38486	806/101075
student	50	1861/389174	2079/733080	814/135282	734/211029	1774/484257
heloc	50	187/9995	216/21293	110/11791	170/18185	25/3195

corner of Figure 3, and by imbalanced, we refer to the topology shown in the bottom of Figure 3. In these experiments we terminated each Cplex run after 3 hours and when this happens on all five runs we report "*" in the tables instead of the time. In the case when some runs terminated in less than two hours and some did not, we averaged the times of the finished runs and reported the time in the able, followed by "(*)".

Table 4 Solution times (in seconds) for **krkp, bc and a1a**.

Topology	Data set	100	200	300	400	500	600
depth2	krkp	2.7	6.0	11.1	14.1	17.4	22.0
depth-2.5	krkp	13.4	34.0	76.5	97.3	796.6	321.4
depth-3	krkp	238.9	1851.3	1556.7	2226.4	4320.7(*)	6238.8(*)
imbalanced	krkp	568.5	4367.2(*)	5950.1(*)	6660.2(*)	*	*
depth2	bc	1.8	3.6	6.8	8.7	12.5	14.1
depth2.5	bc	9.7	35.4	55.2	106.3	175.4	199.6
depth3	bc	9.3	252.6	531.6	2100.5	2917.7	6753.8(*)
imbalanced	bc	19.8	2238.4	2843.52(*)	4706.9	6861.5(*)	*
depth2	a1a	2.9	7.1	11.5	18.3	23.0	31.1
depth2.5	a1a	72.7	470.8	754.6	935.3	961.1	3032.2
depth3	a1a	364.4	1975.6	5928.7(*)	6626.0 (*)	*	*
imbalanced	a1a	2163.5	6282.1(*)	*	*	*	*

As one would expect, Table 4 shows that solving the IP to optimality becomes increasingly more difficult when the sample size increases and when the tree topology becomes more complicated. However, the increase in solution time as sample size increases differs significantly among different datasets for the same tree topology depending on the number of features and groups of the dataset as well as how well the data can be classified using a decision tree. Note that even though the imbalanced trees and depth-3 trees have the same number of nodes, solving the IP for imbalanced trees is more challenging. We believe that this is at least partly due to the fact that symmetry breaking using anchor features has to be disabled at the root node of imbalanced trees, as the tree is not symmetric. To confirm this we switched off symmetry breaking for depth-3 trees and the solutions time general increased dramatically. For example for *a1a* the corresponding row of the table became [3469.1(∗), 6430.8(∗), ∗, ∗, ∗, ∗] which means that some of the instances with

100 and 200 samples did not solve to optimality and for solved instances the average time for 100 samples increased from 364.4 to 3469.1 seconds and time for 200 samples increased from 1975.6 to 6430 seconds. Moreover, none of the larger instances finished solving. For *bc* the corresponding row of the table became [16.5, 2538.7, 4477.4(∗), 6721.6(∗), 6729.9(∗), ∗].

Restricting the number of features in the data can significantly reduce computational time. To demonstrate this, we run the following experiments: we first repeatedly apply the CART algorithm to each data set, using 90% of the data and default setting and thus not applying any particular restriction of the size of the tree. We then select groups that have been used for branching decision at least once in the CART tree. We then remove all other feature groups from the IP formulation (by setting the corresponding v variables to 0) and apply our ODT model to the reduced problem. On average this procedure reduced the number of original features (groups) in *a1a* from 14 to 7.2, in *bc* from 9 to 2.4, and in *krkp* from 36 to 8. The effect of this reduction on the solution time is illustrated in Table 5. We can see that in many cases significant improvement in terms of time is achieved over results reported in Table 4. We will discuss the effect of the feature selection on the prediction accuracy later in Section 5.4.

Table 5 Solution times (in seconds) for **krkp, bc and a1a** using feature selection

Topology	Data set	100	200	300	400	500	600
depth2	krkp	0.2	0.3	0.8	1.4	2.4	2.8
depth-2.5	krkp	1.0	2.0	4.7	8.0	12.3	14.8
depth-3	krkp	1.8	4.5	12.9	19.0	31.0	37.2
imbalanced	krkp	4.3	10.7	36.9	60.1	90.3	108.8
depth2	bc	0.2	0.23	0.3	0.4	0.5	0.8
depth2.5	bc	0.7	1.6	2.7	4.4	6.8	10.6
depth3	bc	0.8	2.4	4.1	6.2	9.0	12.1
imbalanced	bc	2.4	5.8	10.7	18.4	28.9	41.9
depth2	a1a	1.0	2.4	3.8	5.6	8.4	10.2
depth2.5	a1a	8.8	22.5	36.9	72.3	105.8	145.3
depth3	a1a	47.6	288.4	610.8	1636.3	1963.7	1987.2
imbalanced	a1a	167.8	767.0	2020.4	4069.7(∗)	5786.1(∗)	6334.0(∗)

5.2 Effect of combinatorial branching

We next make a comparison to see the effect of the constraint on combinatorial branching for categorical data which is discussed in Section 4.4. When using this constraint with $max.card = 1$ we recover "simple" branching rules where branching is performed using only one possible value of the feature, as is done in [3]. We compare simple branching denoted as *simple*, constrained branching using $max.card = 2$, denoted by *comb-con* and unconstrained branching, denoted as *comb-unc*. We have also tried $max.card = 3$ and $max.card = 4$, but $max.card = 2$ consistently gave better testing accuracy than the other values. We only show the results for two data sets, *a1a* and *mush* because for the other data sets combinatorial branching did not produce different results as most of the
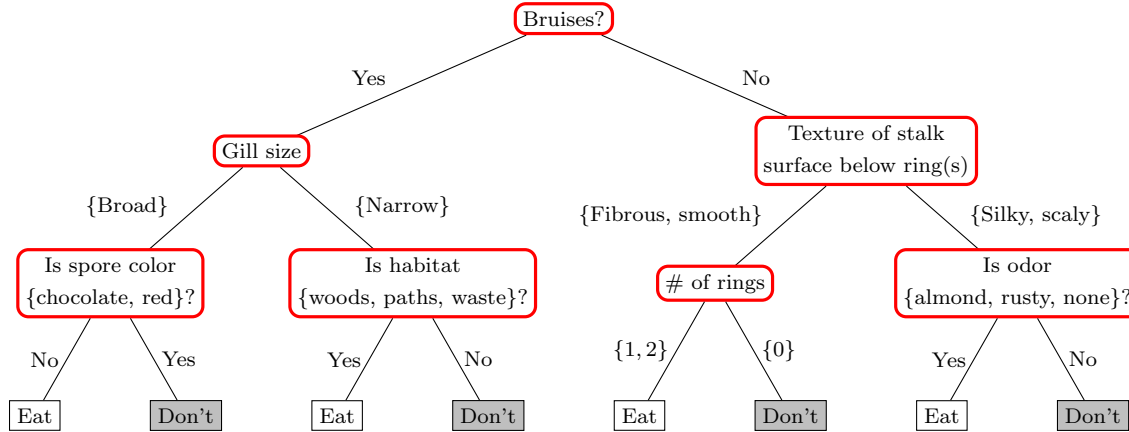
categorical features had only 2 or 3 possible values. We compare decision trees of depths 2 and 3 trained using data sets of size 600. Results averaged over five runs are shown in Table 6.

Table 6 The average training (testing) accuracy for combinatorial vs. simple branching using depth-2 and depth-3 trees

Dataset	depth-2			depth-3		
	simple	comb-con	comb-unc	simple	comb-con	comb-unc
a1a	82.2 (80.8)	82.9 (81.0)	83.3 (79.9)	84.0 (80.8)	84.8 (80.8)	85.7 (80.1)
mush	95.8 (95.7)	99.6 (99.4)	99.6 (99.4)	98.4 (97.7)	99.9 (99.4)	99.9 (99.3)

We see that for *mush* using combinatorial branching makes a significant improvement. In particular, for depth-3 trees and even without max cardinality constraint, it achieves a 99.3% out-of-sample accuracy compared to 97.7% for simple branching. We show the optimal depth-3 tree for *mush* dataset in Figure 4. However, for *a1a* - even though unconstrained combinatorial branching achieves good training accuracy they do not generalize as well as simple branching rules. In particular, the *a1a* dataset contains one group (occupation) with many different possible values. Branching on this group results in combinatorially many possible decisions which leads to overfitting. Adding a constraint with $max.card = 2$ remedies the situation, while still providing a small improvement over simple branching.

Fig. 4 Optimal depth-3 decision tree for the Mushroom dataset with %99.3 out of sample accuracy.

5.3 Effect of constraints for numerical features.

Here we compare the effect of special constraints introduced for the numerical features in Section 4.5. The results of this comparison are shown in Table 7. When the constraint is imposed, the feature group is treated as numerical, and this formulation is label with "n", for numerical. When the constraint is not imposed, then the group is treated as if the original feature is categorical, and the formulation is labeled with "c", for categorical. We compare both accuracy and time averaged from 5 runs with 30 mins limit.

Table 7 The average training (testing) accuracy/solution time with or without constraints for numerical features.

Dataset	n/c	depth-2	depth-2.5	depth-3	Imbalanced
a1a	n	82.9 (81.0)/22	84.7 (80.5)/748	84.8 (80.8)/1800	84.7 (80.7)/1800
	c	82.9 (81.0)/24	84.5 (81.0)/1191	84.7 (80.3)/1800	84.8 (80.1)/1800
bc	n	96.7 (96.6)/6	97.5 (95.4)/70	97.8 (96.2)/608	97.8 (95.6)/1749
	c	96.7 (96.0)/6	97.8 (94.9)/272	98.4 (94.7)/1800	98.5 (95.5)/1800
heloc	n	72.0 (71.2)/7	73.3 (70.6)/119	73.8 (70.0)/515	74.1 (69.8)/1788
	c	72.9 (70.4)/13.6	74.9 (68.0)/1711	75.9 (68.1)/1800	75.6 (68.6)/1800
student	n	92.1 (90.5)/1	92.6 (91.0)/8	92.6 (90.5)/25	93.1 (91.5)/127
	c	92.1 (90.5)/1	92.6 (91.0)/10	92.6 (90.5)/62	93.1 (91.0)/113

We observe that overall adding the special constraint to impose the numerical nature of the group improves the testing accuracy and saves computational time.

5.4 Comparison with CART depth-3 trees

We next focus on comparing the accuracy of ODTs with CART. We consider 4 different tree topologies for ODTs: depth-2, depth-2.5, depth-3 and imbalanced. We use CART as implemented in the package rpart for R [17]. We compare the performance of ODT to CART by restricting the maximum depth of the learned CART trees to 3, thus allowing at most 8 leaf nodes, which is the maximum that our trees can have. We note that this does not mean the learned CARTs have the same topology as our ODTs. In fact, we found that due to various pruning heuristics, the topologies of the trees learned by CART vary erratically and in most cases the tree has much fewer that 8 leaves, as is shown in Table 8. On the other hand, in a later section we show that when CART is not restricted to maximum depth-3 the resulting trees are much larger.

We also investigate the effect of *feature selection* by running CART first and considering only the features used by CART in constructing ODTs. For each dataset, we generate five random training/testing splits of the dataset by sampling without replacement and report the averages. We use 90% of the data for training CART and we use min{90%, 600} data points for training ODTs.

In Tables 8 and 9 we show the results for ODTs trained for up to 30 minutes with and without feature selection, respectively, and compare with CART trees of depth-3. In both tables we list the average training and testing accuracy, in percentages, over the five runs. We highlight in bold the best testing accuracy achieved by the ODTs if it is more than 1% larger than that achieved by CART, and reversely, highlight accuracy of CART when it is more than 1% larger than best accuracy of ODT. The standard deviation in all cases is fairly small, typically around $0.2 - 0.3\%$.

In Table 8 we see that testing accuracy achieved by ODTs after 30 minutes of training is significant better than that of depth-3 CART. Comparing Tables 8 and 9, we see that on average the feature selection typically degrades training accuracy but results in better testing accuracy. This can be explained by the fact that reducing the number of features prevents the ODTs from overfitting. This observation suggests that using feature selection, especially for larger trees could be beneficial not only for computational speedup but for better accuracy.

Table 8 The average training (testing) accuracy with 30 mins limit without feature selection.

Dataset	depth-2	depth-2.5	depth-3	Imbalanced	CART-D3	# of leaves
a1a	82.9 (80.9)	84.7 (80.5)	84.7 (**81.0**)	85.2 (80.0)	82.0 (79.3)	3.6
bc	96.7 (**96.6**)	97.5 (95.6)	97.8 (94.9)	97.9 (96.4)	96.0 (94.6)	3.8
heloc	72.4 (69.8)	73.2 (70.0)	73.7 (69.6)	73.0 (68.8)	70.8 (71.0)	2
krkp	86.7 (87.0)	93.2 (93.9)	93.3 (93.9)	94.1 (**94.1**)	90.4 (90.3)	4
mush	99.6 (99.4)	100.0 (99.5)	100.0 (99.7)	100.0 (99.6)	99.4 (99.3)	4
ttt	71.8 (67.7)	77.0 (72.7)	79.3 (74.2)	81.9 (**79.5**)	75.3 (73.1)	6.6
monks-1	78.2 (74.1)	84.1 (76.8)	89.6 (82.3)	100.0 (**100.0**)	76.6 (76.8)	2.4
votes	96.2 (95.5)	96.9 (93.6)	97.4 (94.1)	98.0 (95.0)	95.7 (95.9)	2.4
heart	85.5 (88.1)	88.6 (89.6)	88.7 (89.6)	90.7 (85.2)	88.5 (**91.1**)	4
student	92.8 (**91.0**)	93.1 (91.0)	93.3 (89.0)	93.4 (89.5)	89.5 (86.0)	4.4

Table 9 The average training (testing) accuracy with 30 mins limit with feature selection

Dataset	depth-2	depth-2.5	depth-3	Imbalanced	CART-D3
a1a	82.9 (**81.0**)	84.7 (80.5)	84.8 (80.8)	84.7 (80.7)	82.0 (79.3)
bc	96.7 (**96.6**)	97.5 (95.4)	97.8 (96.2)	97.8 (95.6)	96.0 (94.6)
heloc	72.0 (71.2)	73.3 (70.6)	73.8 (70.0)	74.1 (69.8)	70.8 (71.0)
krkp	86.7 (87.0)	93.2 (93.9)	93.2 (**93.9**)	94.6 (93.8)	90.4 (90.3)
mush	99.6 (99.4)	99.9 (99.4)	99.9 (99.4)	100.0 (99.6)	99.4 (99.3)
ttt	71.8 (67.7)	77.0 (72.7)	79.3 (74.2)	81.9 (**79.5**)	75.3 (73.1)
monks-1	78.2 (74.1)	84.1 (76.8)	89.6 (82.3)	100.0 (**100.0**)	76.6 (76.8)
votes	95.9 (95.5)	96.3 (95.0)	96.7 (95.0)	97.3 (96.8)	95.7 (95.9)
heart	85.5 (88.1)	88.6 (90.4)	88.6 (90.4)	90.2 (88.9)	88.5 (91.1)
student	92.1 (90.5)	92.6 (91.0)	92.6 (90.5)	93.1 (**91.5**)	89.5 (86.0)

We next repeat the same experiments from Tables 8 and 9 with a 5 minute time limit on Cplex and report the results in Tables 10 and 11. Note that the time for feature selection is negligible.

Table 10 The average training (testing) accuracy with 5 mins limit without feature selection.

Dataset	depth-2	depth-2.5	depth-3	Imbalanced	CART-D3
a1a	82.9 (**80.9**)	84.5 (80.6)	84.4 (80.9)	83.5 (80.4)	82.0 (79.3)
bc	96.7 (**96.6**)	97.5 (95.6)	97.7 (96.4)	97.6 (96.2)	96.0 (94.6)
heloc	72.4 (69.8)	72.0 (69.1)	66.2 (65.0)	58.2 (57.6)	70.8 (71.0)
krkp	86.7 (87.0)	93.2 (**93.9**)	92.1 (92.1)	92.9 (92.9)	90.4 (90.3)
mush	99.6 (99.4)	100.0 (99.5)	100.0 (99.7)	100.0 (99.7)	99.4 (99.3)
ttt	71.8 (67.7)	77.0 (72.7)	78.7 (74.0)	77.5 (**75.0**)	75.3 (73.1)
monks-1	78.2 (74.1)	84.1 (76.8)	89.6 (82.3)	100.0 (**100.0**)	76.6 (76.8)
votes	96.2 (95.5)	96.9 (93.6)	97.3 (94.5)	97.5 (92.7)	95.7 (95.9)
heart	85.5 (88.1)	88.6 (89.6)	88.7 (89.6)	90.4 (88.1)	88.5 (**91.1**)
student	92.8 (91.0)	93.0 (**91.5**)	93.0 (90.0)	87.7 (86.5)	89.5 (86.0)

Comparing Tables 8 and 10, we do not see a significant difference in accuracy for depth-2 and depth-2.5 ODTs due to the reduction of the time limit from 30 minutes to 5 minutes. For depth-3 ODTs, and the imbalanced trees however, both training and testing performance gets noticeably worse due to the reduction of the time limit. Comparing Tables 10 and 11, we see that in most cases feature selection helps in terms of both training and testing accuracy.

Table 11 The average training (testing) accuracy with 5 mins limit with feature selection

Dataset	depth-2	depth-2.5	depth-3	Imbalanced	CART-D3
a1a	82.9 (**81.0**)	84.6 (80.4)	84.4 (80.2)	84.6 (80.7)	82.0 (79.3)
bc	96.7 (**96.6**)	97.5 (95.4)	97.7 (95.8)	97.7 (95.2)	96.0 (94.6)
heloc	72.0 (71.2)	73.3 (70.6)	73.7 (69.7)	73.5 (70.9)	70.8 (71.0)
krkp	86.7 (87.0)	93.2 (93.9)	93.2 (**93.9**)	94.6 (93.8)	90.4 (90.3)
mush	99.6 (99.4)	99.9 (99.4)	99.9 (99.4)	99.9 (99.4)	99.4 (99.3)
ttt	71.8 (67.7)	77.0 (72.7)	78.7 (74.0)	77.5 (**75.0**)	75.3 (73.1)
monks-1	78.2 (74.1)	84.1 (76.8)	89.6 (82.3)	100.0 (**100.0**)	76.6 (76.8)
votes	95.9 (95.5)	96.3 (95.0)	96.7 (95.0)	97.3 (96.8)	95.7 (95.9)
heart	85.5 (88.1)	88.6 (90.4)	88.6 (90.4)	90.2 (88.9)	88.5 (91.1)
student	92.1 (90.5)	92.6 (91.0)	92.6 (90.5)	93.1 (**91.5**)	89.5 (86.0)

Overall the testing accuracy degrades between Tables 8 and 11, but not very significantly, thus we conclude that feature selection helps for larger trees independent of the time limit. Moreover, average testing accuracy of ODTs obtained only after 5 minutes of computation using feature selection seems to be similar to testing accuracy with 30 minute time limit (with or without feature selection) and thus still outperforms CART. We should also note that when the IPs are terminated earlier, the optimality gap is usually larger but it often happens that an optimal or a near optimal integral solution is already obtained by Cplex.

5.5 Effect of training set size

To demonstrate the effect of the training set size on the resulting testing accuracy we present the appropriate comparison in Table 12. In these experiments we run Cplex with a 30 minute time.

Table 12 Comparison of training (testing) accuracy across training data sizes with 30 minutes limit and feature selection

Dataset	Topology	600	1200	1800	2400
a1a	2	82.9 (81.0)	82.4 (79.3)	82.0 (79.6)	82.0 (79.6)
krkp	2	86.7 (87.0)	86.8 (87.0)	86.8 (87.1)	86.8 (87.2)
mush	2	99.6 (99.4)	99.5 (99.4)	99.4 (99.4)	99.4 (99.4)
heloc	2	72.0 (71.2)	72.2 (70.8)	71.9 (71.2)	71.7 (71.2)
a1a	2.5	84.7 (80.5)	83.7 (80.0)	83.4 (79.6)	83.4 (79.6)
krkp	2.5	93.2 (93.9)	93.8 (93.8)	93.6 (94.0)	93.7 (94.1)
mush	2.5	99.6 (99.4)	99.8 (99.5)	99.7 (99.6)	99.7 (99.6)
heloc	2.5	73.3 (70.6)	73.1 (70.9)	72.6 (70.6)	72.3 (71.5)
a1a	3	84.7 (80.7)	83.6 (79.6)	83.3 (80.2)	83.3 (80.2)
krkp	3	94.6 (93.8)	93.8 (93.8)	93.6 (94.0)	93.7 (94.1)
mush	3	100.0 (99.6)	99.9 (99.6)	99.9 (99.7)	99.8 (99.8)
heloc	3	73.8 (70.0)	73.5 (70.9)	72.9 (71.3)	72.5 (71.4)
a1a	IB	84.8 (80.8)	83.6 (79.2)	82.5 (79.6)	82.2 (79.0)
krkp	IB	93.2 (93.9)	94.5 (93.7)	94.2 (93.9)	94.1 (94.1)
mush	IB	99.9 (99.4)	100.0 (99.8)	100.0 (100.0)	100.0 (100.0)
heloc	IB	74.1 (69.8)	73.2 (71.0)	72.1 (71.4)	72.0 (71.4)

We observe that in most cases increasing the size of the training data narrows the gap between training and testing accuracy. This can happen for two reasons - because optimization progress slows down and training accuracy drops and/or because there is less overfitting. For example, for *a1a* it appears to be harder to find the better tree and so both the training and the testing accuracy drops, while for *mush* testing accuracy gets better, as the gap between training and testing accuracy closes. We also see, for example in the case of *mush* and *krkp*, the effect of the increase of the data set tends to diminish as the gap between training and testing accuracy. This is a common behavior for machine learning models, as larger training data tends to be more representative with respect to the entire data set. However, in our case, we utilize the larger data set to perform prior feature selection and as a result relatively small training sets are often sufficient for training of the ODTs. Hence, the computational burden of solving IPs to train the ODTs is balanced by the lack of need to use large training sets.

5.6 Choosing the tree topology

In this section we discuss how to chose the best tree topology via cross-validation and compare the accuracy obtained by the chosen topology to the accuracy of trees obtained by CART with cross-validation.

For each dataset we randomly selected 90% of the data points to use for training and validation, leaving the remaining data for final testing. For the smaller data sets, we select the best topology using standard 5-fold cross validation. For large data sets such as *a1a, bc, krkp, mush* and *ttt*, we instead repeat the following experiment 5 times: we randomly select 600 data points as the training set and train a tree of each topology on this set. The remaining data is used as a validation set and we compute the accuracy of each trained tree on this set. After 5 experiments, we select the topology that has the best average validation accuracy. We then retrain the tree with this topology and report the testing accuracy using the hold-out 10%. We train CART with 90% of the data points, allowing it to choose the tree depth using its default setting and then report the testing accuracy using the hold-out set. We summarize the results in Table 13 where for each method we list the average testing accuracy and the average number of leaves in the tree chosen via cross-validation. We set ODT time limit to 30 mins and used feature selection from CART trained on 90% of each dataset.

We can summarize the results in Table 13 as follows: in most cases, either ODTs outperform CARTs in terms of accuracy or else they tend to have a significantly simpler structure than the CART trees. In particular, for data sets *a1a*, *student* and *bc* that contain interpretable human-relatable data, ODTs perform better in terms of accuracy and better or comparably in interpretability, undoubtedly because there exist simple shallow trees that make good predictors for such data sets, and the exact optimization method such as ours can find such trees, while a heuristic, such as CART may not. On the other hand, on the dataset *ttt* (which describes various positions in a combinatorial game), simple two or three levels of decision are simply not enough to predict the game outcome. In this case, we see that CART can achieve better accuracy, but at the cost of using much deeper trees. A similar situation holds for *krkp*, but to a lesser extent. Finally, *monks-1*

Table 13 Comparison of testing accuracy and size of cross validated trees vs. CART

Dataset	ODT	ave. # of leaves	CART	ave. # of leaves
a1a	**80.9**	**6.8**	79.6	9.6
bc	**96.0**	4.8	94.9	**4.2**
heloc	**71.4**	4.8	71.0	**3.6**
krkp	93.6	**6.8**	**96.6**	9
mush	**99.8**	7.6	99.3	**3**
ttt	81.0	**8.0**	**93.1**	20.2
monks-1	**100.0**	**8.0**	82.3	8.6
votes	**95.7**	7.2	95.5	**2.4**
heart	**89.6**	7.2	88.9	**5**
student	**90.5**	**4.8**	86.0	6.2

data set is an artificial data set, classifying robots using simple features describing parts of each robot. Classification in *monks-1* is based on simple rules that can be modeled using shallow trees and ODT performance is much better on that data set than that of CART. In conclusion, our results clearly demonstrate that when classification can be achieved by a small interpretable tree, ODT outperforms CART in accuracy and interpretability.

5.7 Training depth-2 tree on full *heloc* data.

We performed a more detailed study of the *heloc* data set which was introduced in the FICO interpretable machine learning competition [8]. The authors of the winning approach [7] produced a model for this data set which can be represented as a depth-2 decision tree achieving 71.7 testing accuracy. Here we show how we are able to obtain comparable results with our approach. First we applied feature selection using CART, making sure that at least 4 features are selected. Then we trained a depth-2 tree using our ODT model and 90% of the data points (8884 points). The optimal solution was obtained within 405 seconds and the resulting testing accuracy is 71.6. The corresponding CART model gives 71.0 testing accuracy.
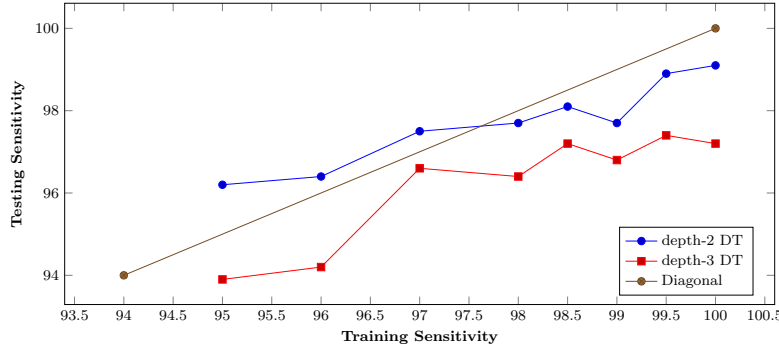
5.8 Results of maximizing sensitivity/specificity

We now present computational results related to the maximization of sensitivity or specificity, as discussed in Section 4.6. We will focus on the *bc* dataset, which contains various measurements of breast tumors. The positive examples in this data sets are the individuals with malignant tumors in the breast. Clearly, it is vitally important to correctly identify all (or almost all) positive examples, since missing a positive example may result in sending a individual who may need cancer treatment home without recommending further tests or treatment. On the other hand, placing a healthy individual into the malignant group, while undesirable, is less damaging, since further tests will simply correct the error. Hence, the goal should be maximizing specificity, while constraining sensitivity. Of course, the constraint on the sensitivity is only guaranteed on the training set. In Table 14 we present the results of solving such model using $\min(\lceil .9n \rceil, 600)$ samples and the resulting testing sensitivity (TPR) and specificity (TNR). We report average and variance over 30 runs.

Table 14 TPR vs. TNR, breast cancer data, depth-2 and depth-3 trees

	depth-2				depth-3			
	Training		Testing		Training		Testing	
TPR	TNR	TPR	TNR	TPR	TNR	TPR	TNR
100	79.6	99.1	76.8	100	91.6	97.2	83.6
99.5	85.4	98.9	82.4	99.5	94.6	97.4	89.7
99	89.5	97.7	89.4	99	97.2	96.8	90.0
98.5	92	98.1	90.9	98.5	97.2	97.2	90.9
98	92.7	97.7	91.0	98	98.7	96.4	94.6
97	95.8	97.5	94.7	97	99.4	96.6	96.1
96	97.3	96.4	93.9	96	99.9	94.2	94.7
95	98.4	96.2	98.0	95	100.0	93.9	93.0

We observe that, while depth-2 trees deliver worse specificity in training than depth-3 trees, they have better generalization and hence closely maintain the desired true positive rate. This is also illustrated in Figure 5.

Fig. 5 Breast Cancer Data, Training v.s. Testing Sensitivity

6 Concluding remarks

We have proposed an integer programming formulation for constructing optimal binary classification trees for data consisting of categorical features. This integer programming formulation takes problem structure into account and, as a result, the number of integer variables in the formulation is independent of the size of the training set. We show that the resulting MILP can be solved to optimality in the case of small decision trees; in the case of larger topologies, a good solution can be obtained within a set time limit. We show that our decision trees tend to outperform those produced by CART, in accuracy and/or interpretability. Moreover, our formulation can be extended to optimize specificity or sensitivity instead of accuracy, which CART cannot do.

Our formulation is more specialized than that proposed recently in [3] and is hence is easier to solve by an MILP solver. However, our model allows flexible branching rules for categorical variables,

as those allowed by CART. In addition the formulations proposed in [3] are not particularly aimed at interpretability.

Several extensions and improvements should be considered in future work. For example, while the number of integer variables does not depend on the size of the training set, the number of continuous variables and the problem difficulty increases with the training set size. Hence, we plan to consider various improvements to the solution technique which may considerably reduce this dependence.

References

1. K.P. Bennett and J. Blue. Optimal decision trees. Technical Report 214, Rensselaer Polytechnic Institute Math Report, 1996.
2. K.P. Bennett and J.A. Blue. A support vector machine approach to decision trees. In *Neural Networks Proceedings of the IEEE World Congress on Computational Intelligence*, volume 3, pages 2396–2401, 1998.
3. D. Bertsimas and J. Dunn. Optimal classification trees. *Machine Learning*, 106.7:1039–1082, 2017.
4. D. Bertsimas and R. Shioda. Classification and regression via integer optimization. *Operations Research*, 55(2):252–271, 2017.
5. L. Breiman, J. H. Friedman, R. A. Olshen, and C. J. Stone. Classification and Regression Trees. Chapman & Hall, New York, 1984.
6. L. Breiman. Random forests. *Machine Learning*, 45(1):5–32, 2001.
7. S. Dash, O. Günlük, D. Wei. Boolean decision rules via column generation. *Advances in Neural Information Processing Systems,* Montreal, Canada, December 2018.
8. FICO Explainable Machine Learning Challenge https://community.fico.com/s/explainable-machine-learning-challenge
9. C. C. Chang and C.J. Lin. LIBSVM: A library for support vector machines. *ACM Transactions on Intelligent Systems and Technology*, 2:27:1–27:27, 2011.
10. L. Hyafil and R.L. Rivest. Constructing optimal binary decision trees is np-complete. *Information Processing Letters*, 5(1):15–17, 1976.
11. S. B. Kotsiantis. Decision trees: a recent overview. *Artificial Intelligence Review*, 39(4):261–283, 2013.
12. M. Lichman. UCI machine learning repository, 2013.
13. D.M. Malioutov and K.R. Varshney. Exact rule learning via boolean compressed sensing. In *Proceedings of the 30th International Conference on Machine Learning*, volume 3, pages 765–773, 2013.
14. S. Murthy and S. Salzberg. Lookahead and pathology in decision tree induction. In *Proceedings of the 14th International Joint Conference on Artificial Intelligence*, volume 2, pages 1025–1031, San Francisco, CA, USA, 1995. Morgan Kaufmann Publishers Inc.
15. M. Norouzi, M. Collins, M.A. Johnson, D.J. Fleet, and P. Kohli. Efficient non-greedy optimization of decision trees. In *Advances in Neural Information Processing Systems*, pages 1720–1728, 2015.
16. J. Ross Quinlan. *C4.5: Programs for Machine Learning*. Morgan Kaufmann Publishers Inc., San Francisco, CA, USA, 1993.
17. T. Therneau, B. Atkinson, and B. Ripley. rpart: Recursive partitioning and regression trees. Technical report, 2017. R package version 4.1-11.
18. T. Wang and C. Rudin. Learning optimized or's of and's. Technical report, 2015. arxiv:1511.02210.
19. T. Wang, C. Rudin, F. Doshi-Velez, Y. Liu, E. Klampfl, and P. MacNeille A Bayesian framework for learning rule sets for interpretable classification. *Journal of Machine Learning Research*, 18(70):137, 2017.