Optimal Decision Trees for Categorical Data via Integer Programming

Oktay Günlük · Jayant Kalagnanam · Matt Menickelly · Katya Scheinberg

January 2, 2018

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 novel 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. 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

1 Introduction

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 data analysis (see the recent survey [9] 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

The work of Katya Scheinberg was partially supported by NSF Grant CCF-1320137. Part of this work was performed while Katya Scheinberg was on sabbatical leave at IBM Research, Google, and University of Oxford, partially supported by the Leverhulme Trust.

· Oktay Günlük, IBM Research, E-mail: gunluk@us.ibm.com
· Jayant Kalagnanam, IBM Research, E-mail: jayant@us.ibm.com
· Matt Menickelly, Argonne National Laboratory, E-mail: mmenickelly@anl.gov
· Katya Scheinberg, Lehigh University, E-mail: katyas@lehigh.edu

(*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 [5,14]) 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 control its size, see [9]. 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.

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.

While finding an optimal DT (even without the combinatorial decisions) is known to be an NP-hard problem [8], 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 of 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 [14] and random forests [6] tend to produce less accurate and less interpretable trees. Extensions of our approach to larger trees and larger data sets are a subject of future research.

The key approach we pursue is to formulate the DT training problem as a mixed-integer optimization problem that it 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 [12], 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.

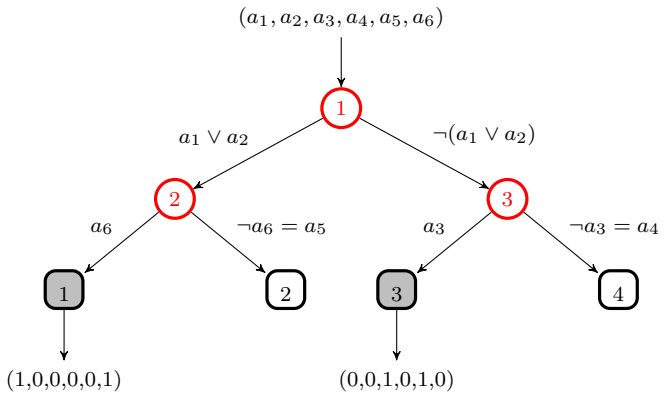
In the 1990s several papers considered optimization formulations for optimal decision tree learning, but deliberately relaxed the inherently integer nature of the problem. In particular, in [1], a large-scale linear optimization problem, which can be viewed as a relaxation, is solved to global optimality 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 opti-

mization problem which admits a favorable dual structure. More recent work [13] 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.

Very 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. Moreover, since the authors of [3] consider the case of real-valued features, this leads to a difficulty in that appropriate big-M constants must be chosen to identify a "less-than/greater-than" split in the data. Such formulations are known to be difficult for MILP solvers and indeed the models in [3] are not reported to have been solved to optimality. While similar big-M modeling choices made in [3] could be applied to our model to yield an extension that can handle continuous features, we do not to consider this issue in this paper, because our primary aim is to produce a strong formulation that can be solved to optimality in a reasonable amount of time. Additionally, for the sake of model simplicity and clarity of presentation in this paper, we have chosen to restrict ourselves to two-class problems, while [3] presents a formulation for arbitrary k-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, [16] 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 [16] remarks on the relationship between this problem and learning optimal decision trees. In [16], 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, [11] 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

Fig. 1 A decision tree example



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].

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 "bucketing" - 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. Clearly, not all numerical features can easily be bucketed and in this case DTs may not be the best choice of classifiers.

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 gives 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

$\bar{G}_j$?" If the categorical data is represented by a binary vector, then the test becomes checking if at least one of the indices from a given collection contains a 1 or not.

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 sample $a^1 = (1,0,0,0,0,1)$ is routed to leaf node 1 and data sample $a^2 = (0,0,1,0,1,0)$ is routed to leaf node 3. 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 child 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 label 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 classification error of the best DT built using the larger tree will always be at least as small as that of the smaller tree. Consequently, for optimization purposes, bigger trees are always at least as good as any of its minors, but they generally result in more computationally challenging optimization problems. However, 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 $I = \{1, 2, \ldots, N\}$, 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 $J = \{1, 2, \ldots, d\}$. In addition, let $g(j) \in G$ denote the group that contains feature $j \in J$ and let $J(g)$ denote the set of features that are contained in group g. 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, namely $g(j)$, is selected at that node. Therefore, for all $k \in K$ we have the following set of constraints:

$$z_j^k \leq v_{g(j)}^k \quad \forall j \in J, \tag{2}$$

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

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

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

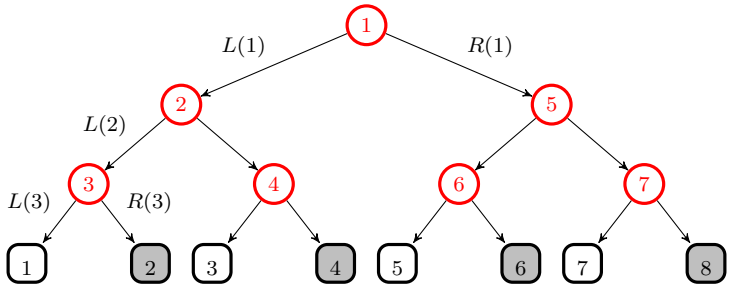
$$L(i,k) = \sum_{j \in J} a_j^i z_j^k \quad \forall k \in K,\ \forall i \in I, \tag{3}$$

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 $g \neq g'$. 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

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

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

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

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

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 constraint:

$$\sum_{j \in J(g)} z_j^k \geq v_g^k, \tag{9}$$

where $J(g) \subset J$ denotes the set of features in group g. This constraint enforces that if a group is selected for branching at a node, then at least one of its features has to be selected as well. Similarly, one can disallow the case when all samples follow the left branch:

$$\sum_{j \in J(g)} z_j^k \leq (|J(g)| - 1) v_g^k. \tag{10}$$

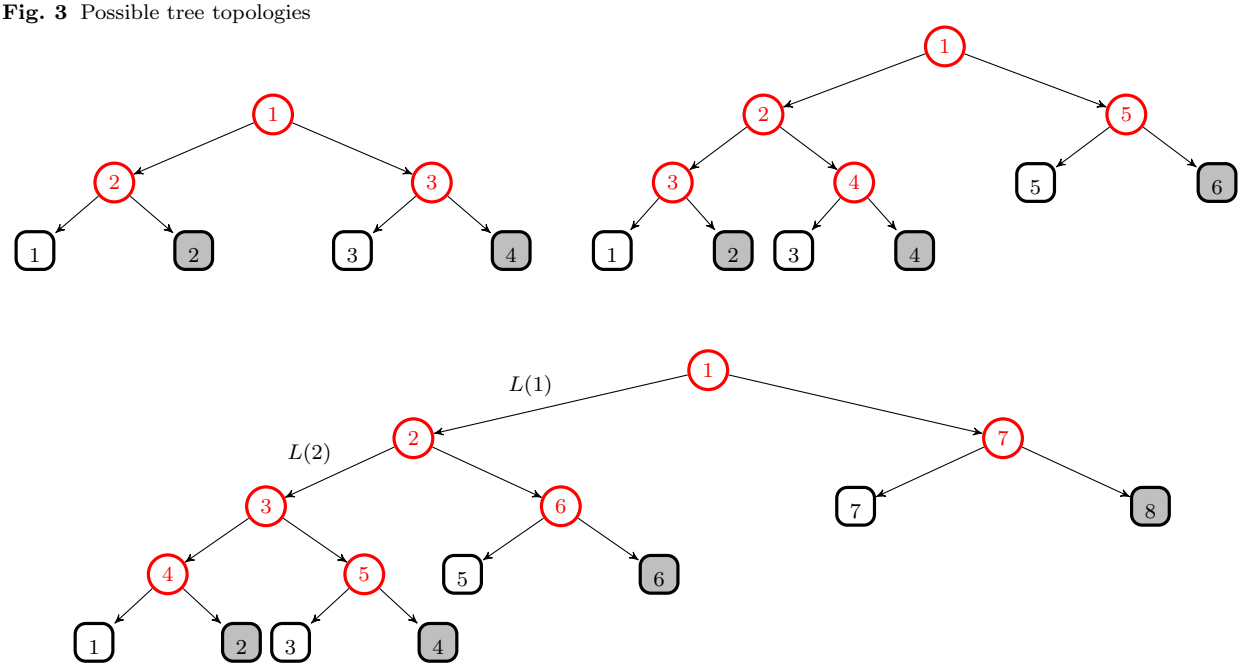
However, we do not use inequalities (9)-(10) in our formulation and allow the decision tree nodes to branch in such a way that all items follow the same branch.

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 4 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 better than that of using either one of these two smaller trees. Similarly, these

Fig. 3 Possible tree topologies



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. 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.

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.

$$
\begin{aligned}
\max \quad & \sum_{i\in I_+}\sum_{b\in B_+} c_b^i + C\sum_{i\in I_-}\sum_{b\in B_-} c_b^i \\
\text{s. t.} \quad & \sum_{g\in G} v_g^k = 1 \quad \forall k\in K, \\
& z_j^k \le v_{g(j)}^k \quad \forall j\in J,\ \forall k\in K \\
& \sum_{b\in B: K^L(b)\ni k} c_b^i \le L(i,k) \quad \forall i\in I,\ k\in K \\
& \sum_{b\in B: K^R(b)\ni k} c_b^i \le R(i,k) \quad \forall i\in I,\ k\in K.
\end{aligned}
$$

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 $|I_+| + C|I_-|$.

Assume the decision tree has at least two levels and let $g^k \in G$ be given for all $k\in K$. We will next construct a solution to the LP that attains this bound. Furthermore, this solution would also satisfy $v_{g^k}^k = 1$ for all $k\in K$.

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. Given any collection of $g^k \in G$ for $k\in K$, we construct the solution (z, v, c) as follows: First we set $v_{g^k}^k = 1$ for all $k\in K$ and $z_j^k = 1/2$ for all $k\in K$ and $j\in J$ such that $g(j) = g^k$. 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, first note that $\sum_{g\in G} v_g^k = 1$ for all $k\in K$ and $z_j^k \le v_{g(j)}^k$ for all $j\in J$ and $k\in K$. Then 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 Strengthening the model

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 $k \in K$.

Let k^0 denote the root node of the decision tree. Note that for any $i \in I$, adding inequalities (11) and (12) for the root node k^0 implies that $\sum_{b\in B} c_b^i \leq 1$ and therefore we can exclude inequality (7) from the formulation.

4.3.2 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 IBM ILOG CPLEX) since the search space increases exponentially, see Margot (2009). The formulation (8) falls into this category as there may be multiple alternate 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 alternate 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 add the following equations to the formulation:

$$z_{j(g)}^k = v_g^k \tag{13}$$

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.

4.3.3 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

$$\left\{(i,b) : i \in I_+, b \in B_+, \text{ or, } i \in I_-, b \in B_-\right\} \tag{14}$$

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

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) and (6) which 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.

4.3.4 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 $\bar{c}_{b_+}^i \neq L(i,a)$, 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 $h \in J$ such that $v_{g(h)}^a > \bar{z}_h^a > 0$. 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 $\bar{z}_h^a = \bar{v}_{g(h)}^a$.

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 $j \in J$ with $g(j) \in \{b, d\}$ and $\bar{z}_j^a > 0$ so that $\bar{z}_h^a = \bar{v}_{g(h)}^a$ still holds. 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 points, which is a contradiction. Hence $\bar{p}$ cannot be fractional. ∎

We have therefore established that the only 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. Thus we have a formulation for training optimal decision trees, where the number of integer variables is *independent* of the number of samples.

4.4 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)

$$\sum_{i \in I_-} \sum_{b \in B_-} c_b^i \geq \lceil (1 - 0.95)|I_-| \rceil \tag{15}$$

and change the objective function (8a) to

$$\sum_{i \in I_+} \sum_{b \in B_+} c_b^i. \tag{16}$$

Likewise, we can produce a model that maximizes specificity while guaranteeing a certain level of sensitivity by switching the expressions in the constraint (15) and objective (16).

5 Computational Results

We now turn to computational experiments for which we used a testset of 11 binary (two-class) classification datasets. We obtained 9 of these datasets are from the UCI Machine Learning repository [10] and the remaining two (a1a and breast-cancer-wisconsin) from LIBSVM [7]. These datasets were selected because they fit into our framework without any adjustment or discretization effort as 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 used

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
monks-problems-2 (monks-2)	432	33%	17	6
monks-problems-3 (monks-3)	432	53%	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

Each dataset/tree topology pair results in a MILP instance, which we implemented in Python 2.7 and then solved with IBM ILOG CPLEX 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, the IP is solved quickly and the effect of improvements is less notable.

We note that the deletion of unnecessary variables discussed in §4.3.3 seems to be performed automatically by CPLEX in preprocessing, and so we do not report results relevant to this modeling choice. However, we will experiment with anchoring (§4.3.2), relaxing appropriate z variables and c variables (§4.3.4), and strengthening the model (§4.3.1). In particular, we compare the model where none of the above techniques are applied (Nothing), only relaxation and strengthening are applied (No Anchor), only anchoring and strengthening are applied (No Relax), only anchoring 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 reach the optimal solution 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	*/22734	*/215216	*/20749	6886/345541	7185/304215
bc	20/125	2/0	6/0	19/162	14/142
krkp	1881/6549	889/38436	420/2906	980/33099	76/4815
mush	3/0	5/0	8/0	5/0	15/3
ttt	*/32868	1053/73643	*/52070	447/19023	728/19067

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* but is too 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.

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	*/14198	4522/395402	*/19649	1656/93501	905/71719
bc	300	4915/15164	455/17212	53/1122	332/4657	289/7488
krkp	400	*/4348	3074/19887	285/7154	*/17518	1979/32983
mush	500	116/27	70/284	146/222	197/429	73/312
ttt	300	*/25941	1571/69058	*/83554	1408/21744	408/18070

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 relaxation improves computational time for *bc* and

krkp compared to the *All* option. However, it gives bad results for *a1a* and *ttt*, 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. Here, by depth 2.5 we refer to the topology shown in the upper right corner of Figure 3, and by imbalanced, we refer to the topology shown in the bottom of Figure 3. In these experiments we terminated the Cplex run after 2 hours and when this happens we report "*" in the tables instead of the time.

Table 4 How solution time (in seconds) changes with topology and sample size for **krkp, bc and a1a**.

Topology	**Data set**	**100**	**200**	**300**	**400**	**500**	**600**
depth2	krkp	2	5	10	16	22	28
depth 2.5	krkp	14	45	93	140	186	270
depth 3	krkp	92	670	787	2048	3546	3708
imbalanced	krkp	508	1572	1732	5819	*	*
depth2	bc	0	2	4	8	10	13
depth2.5	bc	0	38	254	629	1044	1527
depth3	bc	1	16	519	3730	5963	*
imbalanced	bc	1	20	697	*	*	*
depth2	a1a	2	7	12	17	26	36
depth2.5	a1a	110	379	924	1434	1785	2116
depth3	a1a	883	5077	*	*	*	*
imbalanced	a1a	1304	*	*	*	*	*

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 this might be 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. Finally, we should note that even when the IP is not solved to optimality for more complicated tree topologies, the solver still returns a feasible solution (DT) of very high quality. As we discuss in the next section, these solutions can give better training accuracy then optimal solutions for simpler topologies.

5.2 Combinatorial branching vs. simple branching

We next make a comparison to detect whether our combinatorial branching rules to partition the data have advantage over branching on a single feature. Instead of implementing a new model, we utilize the same model developed in this paper as follows: for each binary dataset, we constructed a 2-member group for each feature containing the original bit and its complement to yield a dataset

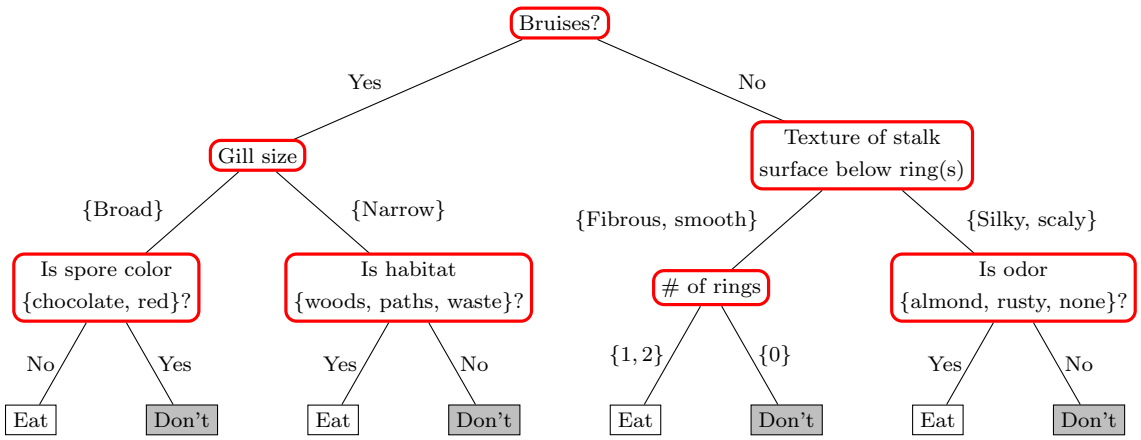
with twice as many features and as many groups as features in the original dataset. It is easy to verify that this is a correct formulation for the optimal decision tree obtainable from branching on one feature at a time. While this may make the solution time slower than necessary (for example when compared to [3] which uses simple branching), the resulting accuracy remains the same. We compare decision trees of depths 2 and 3 trained using data sets of size 600 as in Table 6. Results averaged over 30 runs are shown in Table 5.

Table 5 The average training (testing) accuracy for combinatorial vs. simple branching using depth 2 and depth 3 trees

	Depth 2		**Depth 3**	
Dataset	**simple**	**combin.**	**simple**	**combin.**
a1a	80.8 (78.8)	83.2 (**79.9**)	81.9 (78.9)	85.8 (79.2)
bc	93.6 (93.8)	96.6 (**96.2**)	95.8 (96.0)	98.4 (94.4)
krkp	80.2 (80.1)	86.6 (87.0)	84.9 (84.3)	93.9 (**93.7**)
mush	96.4 (96.3)	99.5 (99.4)	99.3 (99.1)	100 (**99.7**)
ttt	71.2 (68.5)	71.7 (67.9)	78.3 (**73.3**)	79.6 (**73.3**)

We observe that, in general, combinatorial branching yields better accuracy than simple branching. In particular, for depth 3 trees it achieves a 99.7% out-of-sample accuracy for the *mush* dataset compared to 99.1% for simple branching. A similar comparison holds (93.7% v.s. 84.3%) for *krkp* as well. We show the optimal depth 3 tree for *mush* dataset in Figure 4. However, in some cases - for instance, *a1a* and *bc* with depth 3 trees - even though combinatorial branching rules achieve good training accuracy they do not generalize as well as simple branching rules. For both these datasets, depth 2 trees have better accuracy and combinatorial branching again outperforms simple branching. In particular, he *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. For such cases, we would recommend restricting the subset of possible branching rules. Such restrictions can be easily added to our model in form of various constraints and should ideally depend on the context of the data. Note that it is not clear if such restrictions can be easily added to CART. Investigating such extensions is a subject of future work.

Fig. 4 Optimal depth 3 decision tree for the Mushroom dataset with %99.7 out of sample accuracy.



5.3 Testing accuracy vs. tree topology

In this section we focus on comparing the accuracy of our optimal decision trees as a function of the dataset size and the tree topology. In Table 6, we compare the accuracy for different tree topologies - depth 2, depth 2.5, depth 3 and imbalanced - and the accuracy of CART as implemented in the package rpart for R [15]. For each dataset, we generate 30 random training/testing splits of the dataset by sampling a training set of size $\min(\lceil .9n \rceil, 600)$ without replacement, training both models (ODT and CART) on this training set, and then testing the accuracy of the model on the holdout testing set. In these first comparisons to CART, 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 running ODT, we set a time limit of 30 minutes for each run, although, because of our choice of sample size in this comparison, most instances were solved within that time limit. The table lists the average training and testing accuracy, in percentages, over the 30 runs. The standard deviation in all cases is fairly small, typically around $0.2 - 0.3\%$.

Table 6 The average training (testing) accuracy obtained by different topologies.

Dataset	**Depth 2**	**Depth 2.5**	**Imbalanced**	**Depth 3**	**CART-D3**
a1a	83.2 (79.9)	85.2 (80.0)	85.3 (79.4)	85.8 (79.2)	81.4 **(80.8)**
bc	96.6 **(96.2)**	97.9 (94.4)	98.3 (95.5)	98.4 (94.4)	95.7 (93.7)
krkp	86.6 (87.0)	93.8 (93.8)	94.1 **(93.9)**	93.9 (93.7)	90.4 (90.7)
mush	99.5 (99.4)	100 **(99.7)**	100 **(99.7)**	100 **(99.7)**	96.6 (96.4)
ttt	71.7 (67.9)	76.6 (73.4)	79.7 **(79.4)**	79.6 (73.3)	74.8 (73.7)
monks-1	78.2 (74.1)	84.5 (73.2)	97.9 **(95.7)**	89.6 (82.1)	78.3 (78.9)
monks-2	67.1 (67.8)	67.7 (56.8)	68.9 (50.6)	67.8 (53.6)	66.9 **(69.2)**
monks-3	97.2 (97.0)	100 **(100)**	100 **(100)**	100 **(100)**	99.5 (99.8)
votes	96.2 (95.5)	96.8 (95.0)	97.9 **(95.7)**	97.3 (94.3)	95.8 **(95.7)**
heart	82.1 (86.2)	85.7 (85.8)	85.2 (85.2)	88.4 (85.5)	88.3 (**86.9)**
student	89.4 (92.1)	92.0 (**92.9)**	95.1 (92.7)	94.4 (92.1)	92.1 (91.7)

To demonstrate the effect of the training set size on the resulting testing accuracy we present the appropriate comparison in Tables 7 for the depth 3 and imbalanced tree topologies.

Table 7 Comparison of training (testing) accuracy across training data size for depth 3 (D3) and imbalanced (IB) trees.

Dataset	**Topology**	**100**	**200**	**300**	**400**	**500**	**600**
a1a	D3	95.6 (71.9)	90.8 (75.0)	88.7 (76.0)	87.6 (77.7)	86.4 (79.1)	85.7 (79.2)
bc	D3	100 (90.0)	100 (91.4)	99.9 (92.8)	99.2 (93.6)	98.8 (94.3)	98.4 (94.4)
krkp	D3	96.4 (90.0)	94.6 (92.1)	94.5 (93.7)	94.2 (93.6)	94.0 (93.6)	93.9 (93.7)
mush	D3	100 (97.4)	100 (98.4)	100 (99.1)	100 (99.5)	100 (99.6)	100 (99.7)
ttt	D3	87.4 (70.6)	83.4 (74.0)	81.9 (73.4)	80.8 (73.9)	80.2 (73.2)	79.6 (73.3)
a1a	IB	96.3 (72.5)	91.21 (75.6)	88.7 (77.8)	87.6 (77.9)	86.5 (79.0)	85.3 (79.4)
bc	IB	100 (90.6)	100 (91.9)	99.9 (92.7)	99.2 (94.3)	98.6 (94.3)	98.3 (95.5)
krkp	IB	97.8 (91.6)	96.1 (93.5)	95.7 (93.3)	95.0 (93.6)	94.6 (93.8)	94.1 (93.9)
mush	IB	100 (97.0)	100 (98.7)	100 (99.0)	100 (99.6)	100 (99.5)	100 (99.7)
ttt	IB	87.2 (71.2)	84.3 (78.1)	83.5 (79.3)	82.3 (79.0)	80.5 (77.6)	79.7 (79.4)

We observe that in most cases increasing the size of the training data improves the testing accuracy as our trees are less likely to overfit, however, we also see that the effect of this increase tends to diminish as the gap between training and testing accuracy gets smaller. 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 the case of our *simple and interpretable* ODTs we observe that relatively small training sets are sufficient for training. Hence, the computational burden of solving IPs to train the ODTs is balanced by the lack of need to use large training sets.

We also want to note that while, for *a1a* and for *bc* we were unable to train the ODT with more complex topologies to proven optimality using larger training data sizes, the accuracy of the resulting DTs continued to improve. This indicates that, while B&B algorithm did not manage to terminate in the allocated time, it did find a good, possibly optimal, decision tree for each of these cases.

5.4 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 $\min\{90\%, 600\}$ data points to use for training and validation and performed standard 4-fold cross validation to select the best topology for each data set. In particular, after randomly splitting the training data into 4 equal subsets, for each fold we used 3/4 of the set for training and the remaining 1/4 of the set for validation to select the best topology. We report the accuracy after training the tree with the best cross-validated topology on all $\min\{90\%, 600\}$ data points and measure the testing accuracy on the remaining data (which is 10% or $total - 600$). We repeated the same experiment 30 times. For CART we performed 4-fold cross-validation in two settings - in one case, we restricted the CART algorithm to the same training/validation set as we use for ODTs and in the second case, we allowed CART to use 90% of each dataset. We summarize the results in Table 8 and we note two outcomes for CART for data sets *a1a, bc, krkp, mush* and *ttt*, each of which containing more than 600 data points. For each case we list the average number of leaves in the tree chosen via cross-validation.

We can summarize the results in Table 8 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* and *bc* that contain interpretable human-relatable data, ODTs perform better in terms of both accuracy and interpretability, undoubtedly because there exist simple shallow trees that make good predictors for such data sets, and the exact optimization method such as our 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 2 or 3 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. On the *mush* dataset, ODTs and CART are comparable, with

Table 8 Comparison of testing accuracy and size of cross validated trees vs. CART

Dataset	testing accuracy (post CV)	ave.# of leaves	CART min(600,90%)	ave. # of leaves	CART 90%	ave. # of leaves
a1a	**84.6**	6.4	80.8	**5.9**	80.9	7.3
bc	**97.3**	**5.2**	93.3	10.8	94.0	8.6
krkp	93.9	**8.0**	93.3	10.8	**94.0**	8.6
mush	99.9	**5.8**	99.6	7.1	**100**	11.8
ttt	78.6	**7.4**	92.8	40.2	**94.8**	54.5
monks-1	89.5	**8.0**	**92.9**	25.3		
monks-2	67.4	**5.2**	**96.3**	45.8		
monks-3	99.5	7.2	**100**	**5.0**		
votes	**96.7**	5.8	96.1	**2.8**		
heart	86.5	**6.2**	**87.7**	6.3		
student	**93.0**	5.6	91.1	**2.1**		

ODTs once again being much simpler. Finally, *monks* data sets are artificial data sets, classifying robots using simple features describing parts of each robot. Classification in *monks-1* and *monks-3* is based on simple rules that can be modeled using shallow trees, while *monks-2* data set requires a large number of decision nodes for correct classification.

In conclusion, our results clearly demonstrate that when classification can be achieved by a small interpretable tree, ODT outperforms CART in accuracy and interpretability.

5.5 Results of maximizing sensitivity/specificity

We now present computational results related to the maximization of sensitivity or specificity, as discussed in Section 4.4. 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 9 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.

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.

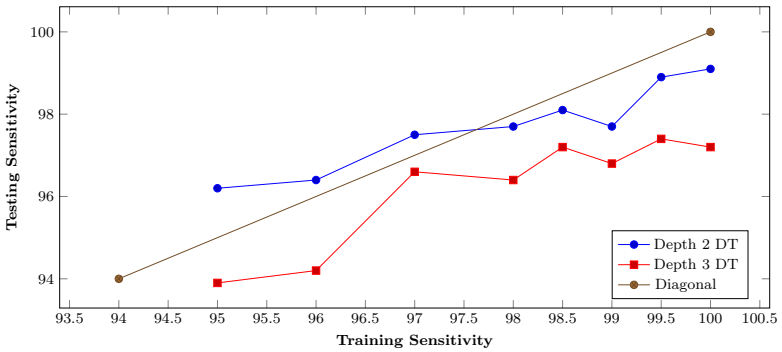
6 Concluding remarks

We have proposed an integer programming formulation for constructing optimal binary classification trees for data consisting of categorical features. This integer programming formulation takes

Table 9 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

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



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. Among the two formulations proposed in [3], one is not aimed at interpretability and the other does not allow flexible branching rules for categorical variables, as those allowed by CART and our method.

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.

In the case of categorical features with many possible values, some additional constraints on the branching rules may be effective in avoiding overfitting. We plan to include such constraints in future experiments.

We also note that the IP model (8) can be extended to deal with real-valued features explicitly. In this case branching decisions are *either* made based on a group of categorical features, *or*, based on the real-valued features. If real-valued features are chosen for branching, then the samples are divided according to a linear classifier on these features optimally chosen by the extended model. Furthermore, it is also possible partition the real-valued features into groups and require that only features that belong to the same group can be used in the classifier. The formulation, however, becomes less tractable with the inclusion of continuous features and specialized techniques might be necessary to implement it efficiently.

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*, 2017. To appear.
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. Leo Breiman. Random forests. *Mach. Learn.*, 45(1):5–32, 2001.
7. Chih-Chung Chang and Chih-Jen Lin. LIBSVM: A library for support vector machines. *ACM Transactions on Intelligent Systems and Technology*, 2:27:1–27:27, 2011. Software available at http://www.csie.ntu.edu.tw/ cjlin/libsvm.
8. L. Hyafil and R.L. Rivest. Constructing optimal binary decision trees is np-complete. *Information Processing Letters*, 5(1):15–17, 1976.
9. S. B. Kotsiantis. Decision trees: a recent overview. *Artificial Intelligence Review*, 39(4):261–283, 2013.
10. M. Lichman. UCI machine learning repository, 2013.
11. 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.
12. Sreerama Murthy and Steven Salzberg. Lookahead and pathology in decision tree induction. In *Proceedings of the 14th International Joint Conference on Artificial Intelligence - Volume 2*, IJCAI'95, pages 1025–1031, San Francisco, CA, USA, 1995. Morgan Kaufmann Publishers Inc.
13. 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.
14. J. Ross Quinlan. *C4.5: Programs for Machine Learning*. Morgan Kaufmann Publishers Inc., San Francisco, CA, USA, 1993.
15. Terry Therneau, Beth Atkinson, and Brian Ripley. rpart: Recursive partitioning and regression trees. Technical report, 2017. R package version 4.1-11.
16. T. Wang and C. Rudin. Learning optimized or's of and's. Technical report, 2015. arxiv:1511.02210.