Learning optimal classification trees using a binary linear program formulation

Conference Paper · November 2018

CITATIONS
0

READS
70

2 authors:

Sicco Verwer
Delft University of Technology
78 PUBLICATIONS **595** CITATIONS
SEE PROFILE

Yingqian Zhang
Technische Universiteit Eindhoven
56 PUBLICATIONS **299** CITATIONS
SEE PROFILE

Learning optimal classification trees using a binary linear program formulation

Sicco Verwer
Delft University of Technology
The Netherlands
s.e.verwer@tudelft.nl

Yingqian Zhang
Eindhoven University of Technology
The Netherlands
yqzhang@tue.nl

Abstract

We provide a new formulation for the problem of learning the optimal classification tree of a given depth as a binary linear program. A limitation of previously proposed Mathematical Optimization formulations is that they create constraints and variables for every row in the training data. As a result, the running time of the existing Integer Linear programming (ILP) formulations increases dramatically with the size of data. In our new binary formulation, we aim to circumvent this problem by making the formulation size largely independent from the training data size. We show experimentally that our formulation achieves better performance than existing formulations on both small and large problem instances within shorter running time.

Introduction

Decision trees (Breiman et al. 1984) have gained increasing popularity these years due to their effectiveness in solving classification and regression problems and their capability to explain prediction results. Learning an optimal decision tree with a predefined depth is NP-hard (Hyafil and Rivest 1976). Hence, greedy based heuristics such as CART (Breiman et al. 1984) and ID3 (Quinlan 1986) have been widely used to construct sub-optimal trees. Recent years have seen an increasing number of work that employ various Mathematical Optimization methods to build better quality decision trees, e.g., (Bennett and Blue 1996; Bessiere, Hebrard, and OSullivan 2009; Verwer and Zhang 2017; Bertsimas and Dunn 2017; Silva 2017; Dash, Günlük, and Wei 2018; Blanquero et al. 2018a; 2018b; Firat et al. 2018).

An advantage of these Mathematical Optimization based approaches is that they are able to employ the powerful optimization solvers to find decision trees. This power has led to interesting new approaches for learning models and rules, see e.g., (Bessiere, Hebrard, and OSullivan 2009; De Raedt, Guns, and Nijssen 2010; Narodytska et al. 2018; Verwer, Zhang, and Ye 2017). In addition, the mathematical optimization models allow flexibility on modeling different learning objectives. For instance, (Verwer and Zhang 2017) incorporate constraints in the proposed Integer Linear Programming (ILP) model to learn discrimination-aware classification trees. In this paper, we focus on the problem of learning optimal classification trees of given depths such that the total classification error is minimized on the training data.

A limitation of the state-of-the-art Mathematical Optimization formulations for this problem is that they create constraints and variables for every row in the training data (Verwer and Zhang 2017; Bertsimas and Dunn 2017). Consequently, the solving time of finding decision trees increases dramatically with the problem size.

We formulate the problem of constructing the optimal decision tree of a given depth as an *binary linear program*. We call our method BinOCT, a Binary encoding for constructing Optimal Classification Trees. Our novel formulation models the selection of decision threshold via a binary search procedure encoded using a type of big-M constraints. This requires a very small number of binary decision variables and is therefore able to find good quality solutions within limited time. Noteworthy is that the number of decision variables is largely independent of the number of training data rows: it only depends logarithmically on the number of unique feature values. Furthermore, our formulation requires fewer constraints than existing approaches. Although this number still depends linearly on the number of data rows. We show using experiments that BinOCT outperforms existing MO based approaches on a variety of data sets in terms of accuracy and computation time.

Related work

Many studies have investigated the interplay of data mining and machine learning with mathematical modeling techniques, see overview in e.g. (Bennett and Parrado-Hernández 2006). In this paper, we are interested in using mathematical optimization methods in order to increase learning performance. For instance, (Bennett and Mangasarian 1993) use linear programming for determining linear combination splits within two-class decision trees. (Chang, Ratinov, and Roth 2012) propose a Constrained Conditional Model (CCM) framework to incorporate domain knowledge into a conditional model for structured learning, in the form of declarative constraints. CCMs solves prediction problems. (Uney and Turkay 2006) build a mixed integer program for multi-class data classification.

(Bennett and Blue 1996) construct binary classification trees with fixed structure and labels. The paths of the tree are

encoded as disjunctive linear inequalities, and non-linear objective functions are introduced to minimize errors. (Norouzi et al. 2015) link the decision tree optimization problem with the problem of structured prediction with latent variables, and use stochastic gradient descent to optimize an upper bound on the empirical loss of the tree's predictions. (Dash, Günlük, and Wei 2018) propose a mathematical optimization model for learning boolean decision rules in disjunctive form or conjunctive normal form. The proposed model takes into account the trade-off between accuracy and the simplicity of the chosen rules and is solved via a column generation method. (Blanquero et al. 2018a; 2018b) use a continuous optimization formulation to learn classification trees, where random decisions are made at internal nodes of the tree. Their approach is essentially a randomized optimal version of CART. (Rhuggenaath et al. 2018) build a mixed integer linear program to learn fuzzy decision trees, where split decisions on the internal nodes of a tree are not crisp. (Firat et al. 2018) apply column generation techniques in constructing univariate binary classification trees. By using threshold sampling on decision nodes of the tree, the proposed approach trades its optimality for speed, i.e., it is capable of handling big data sets with tens of thousands of data rows.

The most relevant work to ours are (Bertsimas and Shioda 2007; Bertsimas and Dunn 2017; Verwer and Zhang 2017). In these papers, the authors formulate the problem of learning classification trees of depth K using Integer Linear programs (ILP). The model proposed in (Bertsimas and Shioda 2007) is quadratic in the data set size and therefore requires a lot of preprocessing in order to reduce the number of generated constraints. The number of decision variables in the ILP model of (Bertsimas and Dunn 2017) is $O(R{\cdot}2^K)$, where R is the number of data rows. In (Verwer and Zhang 2017), a more efficient encoding is proposed for constructing both classification and regression trees, where the number of decision variables is reduced to $O(R \cdot K)$. (Bertsimas and Dunn 2017) show their model is in general better than CART in terms of testing accuracy when the running time was set to two hours for solving each instance. (Verwer and Zhang 2017) also compare their model with CART. Their model outperforms CART with trees up to depth 5 on datasets of size up to 1000.

Learning optimal classification trees as binary linear programs (BinOCT)

We assume readers to be familiar with classification trees, see, e.g., (Flach 2012). The optimization problem that we aim to solve is to find an optimal classification tree of depth K for a given dataset of R rows and F features, such that the total classification error is minimized. The Boolean decisions on each internal nodes of the tree and predictions on the leaves are variables and need to be set such that the classification accuracy is optimized. We solve this problem by formulating it as a Binary Linear Program (BLP), which contains only binary decision variables.

Formulation Intuition

Our formulation for classification trees aims to reduce the dependence of the problem size with the size of the training data. In existing formulations this dependence is present in both the number of constraints and the number of decision variables, see Table 2. The number of (boolean) decision variables that our formulation requires is significantly smaller than that in the previously proposed methods. Moreover, this number is independent of the number of training data rows. Instead, it depends on the maximum amount of unique feature values among all features. Similarly, we also aim to minimize the number of constraints required by our formulation. Below, we first explain the key ideas required to understand our formulation using small examples. We may slightly abuse notations during explanation. At the end of this section, we provide the complete formulation.

Boolean Decision Thresholds. In contrast to earlier formulations that use continuous or integer decision thresholds for each internal node, we represent decision thresholds using only binary variables. When the feature used in the decision is binary, the formulation is intuitive, as explained below. Assume we are learning a tree consisting of a single decision node. Let $l_{r,1}$ and $l_{r,2}$ be binaries indicating that data row r reaches leaf 1 in the left and leaf 2 in the right branch from the root node. A row has to reach a single leaf:

$$l_{r,1} + l_{r,2} = 1.$$

Let t_n be a binary variable for the binary decision threshold for node n, that is, depending on the value of t_n, a data row goes to the left or right branch of node n. Hence, leaf 1 is reached by row r when t_n is 0. Leaf 2 is reached by row r when t_n is 1. This can be encoded by adding the following two constraints

$$l_{r,1} + t_n \leq 1 \text{ and } l_{r,2} - t_n \leq 0.$$

These constraints force $l_{r,1}$ to be 0 when t_n equals 1 and $l_{r,2}$ to 0 when t_n equals 0. The first constraint then guarantees that the leaf not forced to 0 is reached by row r. Note that although the leaf variables l are boolean, they can be modeled as continuous because the above constraints force them to be either 0 or 1. This makes the problem significantly easier because when an optimization solver is used to solve the given BLP model, continuous variables do not have to be considered as branching nodes during its branch-and-bound search, i.e., they can be solved using standard linear programming.

Combining Constraints. A naive formulation based on this intuition would require a large number of constraints, i.e., at least one for every row in the training data. We significantly reduce this number by the observation that we can simply sum the above constraints of threshold checking for all data rows r with feature value V_r^f equal to 1, i.e.,

$$\sum_{r:V_r^f=1} l_{r,1} + M \cdot t_n \leq M \text{ and } \sum_{r:V_r^f=1} l_{r,2} - M \cdot t_n \leq 0,$$

where $M = \sum_{r:V_r^f=1} 1$. Like before, $l_{r,1} = 0$ when $t_n = 1$, and $l_{r,2} = 0$ when $t_n = 0$, only now this is forced for all

rows in the training data. Although it is known that integer programming solvers can experience difficulties with such big-M formulations, in our experience and as shown in the experiments it solves much faster than creating additional constraints for every row in the training data. In the case of larger trees, we can safely add all leaves under the left and right branch to the constraints since their sum is at most 1:

$$\sum_{r:V_r^f=1} \sum_{i \in ll(n)} l_{r,i} + M \cdot t_n \leq M$$
$$\sum_{r:V_r^f=1} \sum_{i \in rl(n)} l_{r,i} - M \cdot t_n \leq 0, \quad (1)$$

where $ll(n)$ and $rl(n)$ are the set of leaves under the left and right branches of node n, respectively, $M = \sum_{r:V_r^f=1} 1$ and t_n is a binary threshold variable for node n. An important observation is that since we force the leaf variables $l_{r,i}$ to be 0, these constraints can be created for every node in the tree without influencing each other. Together they represent row r's path to the leaf i with $l_{r,i}$ equal to 1, i.e., not forced to 0 by any of the node constraints.

Integer Decision Thresholds. The node constraints above are sufficient for modeling all paths of all training data rows when the decision thresholds are boolean. A naive method to transform integer or continuous valued features to binary ones is using a unary representation that uses one variable per decision threshold. Instead, we use a binary representation for the decision thresholds that requires exponentially fewer variables.

Example 1 *For ease of explanation, we assume that we aim to find a single decision node n for the following training set, which contains only one feature with 9 distinct feature values. The rows are ordered using the feature values:*

feature value	*0*	*1*	*2*	*3*	*4*	*5*	*6*	*7*	*8*
target class	+	+	+	-	-	+	-	+	+

Because there is no reason to split groups of examples that have the same label, there are 4 sensible thresholds we can use to split data. In theory it could occur in multivariate datasets such a split is needed to find the optimal tree. We still remove them from consideration to reduce the size of the learning problem, in this case from 8 to 4 possible thresholds. We model these 4 possible thresholds using $\log_2 4 = 2$ binary variables $t_{n,1}$ and $t_{n,2}$ as shown in Figure 1.

Thus, if $t_{n,1}$ is 1, the threshold is one of the first two possible thresholds $th(1)$ or $th(2)$. In our example, such a setting implies that any rows with a feature value greater than 4.5 (the 2nd threshold value) cannot reach any of the leaves under the left branch of node n. Similarly, when $t_{n,1}$ is 0, any rows with a feature value less then 5.5 (the 3rd threshold value) cannot reach any of the leaves under the right branch of node n. This results in the following constraints, updated from Equation (1):

$$\sum_{r:th(2)<V_r^f<th(4)} \sum_{i \in ll(n)} l_{r,i} + M \cdot t_{n,1} \leq M$$

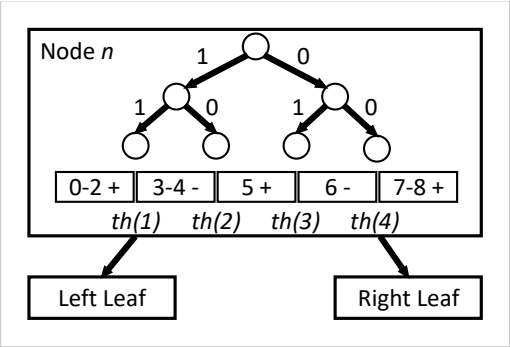

Figure 1: A decision tree with one internal node. Inside of this node, a binary search tree is used to represent one of the 4 possible threshold values $th(1)$-$th(4)$ for Example 1. The edges on the arcs denote the value settings of the binary variables $t_{n,i}$, with i the depth of the search tree. Thus setting $t_{n,1}$ to 1 and $t_{n,2}$ to 0 corresponds to selecting threshold $th(2)$. The boxes show the feature value ranges and class values to the left and right of every decision threshold. A threshold setting of $th(2)$ forces all rows with values from the first two boxes to reach the left leaf, and all rows with values from the last three boxes to reach the right leaf.

$$\sum_{r:th(1)<V_r^f<th(3)} \sum_{i \in rl(n)} l_{r,i} - M' \cdot t_{n,1} \leq 0,$$

where $th(i)$ returns the ith threshold value, and the constants M and M' are $\sum_{r:th(2)<V_r^f<th(4)} 1$ and $\sum_{r:th(1)<V_r^f<th(3)} 1$, respectively. Notice that these constraints force all rows with a feature value of 5 (both greater than the 2nd and smaller than the 3rd threshold) to either the left or right branch, depending on the setting of $t_{n,1}$. Furthermore, notice we do not create these constraints for rows with feature values below the lowest ($th(1)$) or above the highest ($th(4)$) threshold. Rows with values below the minimum threshold can never follow the right branch, no matter which threshold is chosen. The reverse holds for rows with values above the largest threshold and we use separate constraints to model these extreme values, which we discuss later.

The second binary variable $t_{n,2}$ is then used to force the direction for the remaining rows with values between $th(1)$ and $th(2)$, and between $th(3)$ and $th(4)$. Whether $t_{n,2}$ forces this for the first or last range depends on the value of $t_{n,1}$. We first show the first range and include the higher order bit of $t_{n,1}$ using big-M values:

$$\sum_{r:th(1)<V_r^f<th(2)} \sum_{i \in ll(n)} l_{r,i} + M \cdot t_{n,1} + M \cdot t_{n,2} \leq 2 \cdot M$$

$$\sum_{r:th(1)<V_r^f<th(2)} \sum_{i \in rl(n)} l_{r,i} - M \cdot t_{n,2} \leq 0,$$

where $M = \sum_{r:th(1)<V_r^f<th(2)} 1$. Notice that we have to add $M \cdot t_{n,1}$ only for the left leaves. This is due to an effect of threshold value decisions. If $t_{n,2} = 0$, the threshold is either $th(2)$ or $th(4)$. Independent of the value of $t_{n,1}$, the

rows with feature values in range $[th(1), th(2)]$ cannot reach any leaf under the right branch of node n. These rows have feature values that are smaller than either threshold value. In the same way, we make the constraints for the upper range:

$$\sum_{r:th(3)<V_r^f<th(4)} \sum_{i \in ll(n)} l_{r,i} + M \cdot t_{n,2} \leq M$$

$$\sum_{r:th(3)<V_r^f<th(4)} \sum_{i \in rl(n)} l_{r,i} - M \cdot t_{n,1} - M \cdot t_{n,2} \leq 0,$$

with $M = \sum_{r:th(3)<V_r^f<th(4)} 1$.

Example 2 *Let us see the effect of the above* 6 *constraints on our example data. For simplicity, we use the feature value also as row number, i.e.,* $l_{r,i}$ *where* $0 \leq r \leq 8$*, and we assume there is only one single left leaf* 1 *and one single right leaf* 2 *for node* n*. After writing out the above defined constraints, we obtain in order:*

$$l_{5,1} + l_{6,1} + 2 \cdot t_{n,1} \leq 2$$
$$l_{3,2} + l_{4,2} + l_{5,2} - 3 \cdot t_{n,1} \leq 0$$
$$l_{3,1} + l_{4,1} + 2 \cdot t_{n,1} + 2 \cdot t_{n,2} \leq 4$$
$$l_{3,2} + l_{4,2} - 2 \cdot t_{n,2} \leq 0$$
$$l_{6,1} + 1 \cdot t_{n,2} \leq 1$$
$$l_{6,2} - 1 \cdot t_{n,1} - 1 \cdot t_{n,2} \leq 0.$$

The effect of setting $t_{n,1}$*,* $t_{n,2}$ *to* 1,1 *is that* $l_{5,1}$ *and* $l_{6,1}$ *are forced to* 0 *by the first constraint.* $l_{3,1}$ *and* $l_{4,1}$ *are forced to* 0 *by the third constraint.* $l_{6,1}$ *is (again) forced to* 0 *by the fifth constraint. Since the feature values of the last two rows are largest, the last two rows cannot end up in the left leaf, that is,* $l_{7,1}$ *and* $l_{8,1}$ *are always* 0*. Similarly,* $l_{0,1}$*,* $l_{1,1}$*, and* $l_{2,1}$ *are always* 1*. This results in the following table, which we extend with all possible settings of* $t_{n,1}$ *and* $t_{n,2}$*. Note that since we have two leaves, forcing* $l_{i,2}$ *to be* 0 *causes* $l_{i,1}$ *to be set to* 1*:*

$t_{n,1}$	$t_{n,2}$	$l_{0:2,1}$	$l_{3,1}$	$l_{4,1}$	$l_{5,1}$	$l_{6,1}$	$l_{7:8,1}$
1	*1*	*1*	*0*	*0*	*0*	*0*	*0*
1	*0*	*1*	*1*	*1*	*0*	*0*	*0*
0	*1*	*1*	*1*	*1*	*1*	*0*	*0*
0	*0*	*1*	*1*	*1*	*1*	*1*	*0*

This correctly models which rows reach which leaves under the 4 *different threshold settings.*

In our formulation, we extend this idea to arbitrary ranges of threshold values by using a simple recursive routine to generate the corresponding constraints. We make sure that every $t_{n,i}$ bit of the binary encoding divides the possible threshold values in half. The first $t_{n,1}$ is thus always important because it eliminates half of the possible paths of rows (that reach node n) to leaf nodes. Intuitively, this makes it easier for the solver to find solutions since it can eliminate many possible solutions by fixing only a few binary values. Notice also that the used big-M values change depending on the number of influenced rows. In this way, We obtain few constraints with very large M values but few decision variables, and many constraints with small M values but several decision variables.

In the following, we use $bin(f)$ to denote the result of our recursive routine for feature f. For each feature f, we compute the number of possible threshold values T_f as follows. We sort all unique feature values and create thresholds between every two subsequent values, unless all rows with those values belong to the same class (see the small example in Example 1). Pseudocode for the recursive routine for each feature is given in Algorithm 1. Initially, the algorithm is called with $(1, T_f, 1)$. The algorithm returns a set (or iterator) $b \in bin(f)$ over the lower $lr(b)$ and upper $ur(b)$ value ranges, and the required $t_{n,t}$ variables ($tl(b)$) for the left and right leaf nodes.

Algorithm 1 Obtaining the binary encoding value ranges

```
1:  procedure BIN(min, max, depth)
2:      Let b be a binary value range
3:      if max − min <= 1 then
4:          lr(b) = [th(min), th(max)]          lower range of b
5:          ur(b) = [th(min), th(max)]          upper range of b
6:          return [b]
7:      end if
8:      result = [b]
9:      mid = FLOOR((max − min)/2.0)
10:     lr(b) = [th(min), th(min + mid + 1)]    lower range
11:     ur(b) = [th(min + mid), th(max)]        upper range
12:     tl(b) = tr(b) = [depth]                 include current depth t_{n,t}
13:     for b' in BIN(min, min + mid, depth + 1) do
14:         tl(b') = tl(b') + [depth]           include t_{n,t} for ll(n)
15:         result = result + [b']
16:     end for
17:     for b' in BIN(min + mid + 1, max, depth + 1) do
18:         tr(b') = tr(b') + [depth]           include t_{n,t} for rl(n)
19:         result = result + [b']
20:     end for
21:     return result
22: end procedure
```

Features So far, the formulation of node constraints assumed there is only one single feature we can choose from when determining thresholds at internal nodes. For a general case with multiple features, in order to select which feature to use in a node constraint, we add another big-M multiplier, for all features f ($1 \leq f \leq F$) and binary ranges $b \in bin(f)$:

$$M \cdot f_{n,f} + \sum_{r \in lr(b)} \sum_{l \in ll(n)} l_{r,l} + \sum_{t \in tl(b)} M \cdot t_{n,t} \leq M + \sum_{t \in tl(b)} M$$

$$M' \cdot f_{n,f} + \sum_{r \in ur(b)} \sum_{l \in rl(n)} l_{r,l} - \sum_{t \in tl(b)} M' \cdot t_{n,t} \leq M',$$

where $M = \sum_{r \in lr(b)} 1$, $M' = \sum_{r \in ur(b)} 1$, and $ll(n)$ ($rl(n)$) is the set of node n's leaves under the left (right) branch. The additional big-M ensures that these constraints only have effect when $f_{n,f}$ is equal to 1. Of course, we require that exactly one feature is selected for every decision node n:

$$\sum_{1 \leq f \leq F} f_{n,f} = 1.$$

All that remains for modeling the nodes of the decision tree is to add constraints for rows with feature values greater than the largest threshold, or lower than the smallest threshold. These are forced to 0 using only the node feature variable $f_{n,f}$, for all features f:

$$M \cdot f_{n,f} + \sum_{r:\max_{t(f)} < V_r^f} \sum_{l \in ll(n)} l_{r,l} + \sum_{r:V_r^f < \min_{t(f)}} \sum_{l \in rl(n)} l_{r,l} \leq M,$$

where $M = \sum_{r:\max_{t(f)} < V_r^f} 1 + \sum_{r:V_r^f < \min_{t(f)}} 1$, and $\max_{t(f)}$ and $\min_{t(f)}$ are the minimum and maximum decision thresholds for feature f.

Objective Function The node constraints described above model the influence of the decision variables $t_{n,t}$ and $f_{n,f}$ on which leaf node $l_{r,l}$ each of the training data row reaches. We create our objective function using these leaf node (not decision) variables. Similarly to the node constraints, we combine the objective values of as many rows as possible using a big-M value. In this case, we combine all that end in the same leaf with the same target class. For all leaves l and target classes c:

$$\sum_{r:C_r=c} l_{r,l} - M \cdot p_{l,c} \leq e_{l,c},$$

where $M = \sum_{r:C_r=c} 1$, C_r is the class of row r, $p_{l,c}$ is a binary decision variable indicating whether leaf l predicts class c, and $e_{l,c}$ is the number of misclassifications for class c in leaf l. Lastly, every leaf predicts exactly one class value:

$$\sum_{1 \leq l \leq L} p_{l,c} = 1.$$

The BinOCT model

We now present the full formulation for learning optimal classification trees as follows. The objective is to minimize the total classification error. Table 1 lists the used notation.

$$\begin{aligned}
& \min \textstyle\sum_{l,c} e_{l,c} \quad \text{s.t.} \\
\forall_n \quad & \textstyle\sum_f f_{n,f} = 1 \\
\forall_r \quad & \textstyle\sum_l l_{r,l} = 1 \\
\forall_l \quad & \textstyle\sum_c p_{l,c} = 1 \\
\forall_{n,f,b \in bin(f)} \quad & M \cdot f_{n,f} + \textstyle\sum_{r \in lr(b)} \sum_{l \in ll(n)} l_{r,l} + \\
& \textstyle\sum_{t \in tl(b)} M \cdot t_{n,t} - \sum_{t \in tl(b)} M \leq M \\
\forall_{n,f,b \in bin(f)} \quad & M' \cdot f_{n,f} + \textstyle\sum_{r \in rr(b)} \sum_{l \in rl(n)} l_{r,l} - \\
& \textstyle\sum_{t \in tl(b)} M' \cdot t_{n,t} \leq M' \\
\forall_{n,f} \quad & M'' \cdot f_{n,f} + \textstyle\sum_{\max_{t(f)} < f(r)} \sum_{l \in ll(n)} l_{r,l} + \\
& \textstyle\sum_{f(r) < \min_{t(f)}} \sum_{l \in rl(n)} l_{r,l} \leq M'' \\
\forall_{l,c} \quad & \textstyle\sum_{r:C_r=c} l_{r,l} - M''' \cdot p_l \leq e_{l,c}
\end{aligned}$$

with $1 \leq n \leq N, 1 \leq f \leq F, 1 \leq r \leq R, 1 \leq l \leq L, 1 \leq c \leq C$ unless stated otherwise. How to derive the smallest M values for different constraints and the details of equations have been described in the previous subsections. All decision variables $f_{n,f}$, $t_{n,t}$, and $p_{l,c}$ are binary. The number of $f_{n,f}$ variables is bounded by $N \cdot F$. The number of $t_{n,t}$ variables in bounded by $N \cdot \log(T_{\max})$. Note $T_{\max}$ is no more than the number of distinct values for each feature. The number of $p_{l,c}$ variables is bounded by $L \cdot C$. For a depth K tree, the formulation thus requires at most $N \cdot (F + \log(T)) + L \cdot C \leq 2^k(F + C + \log(T))$ decision variables. This depends linearly on the number of features and class values, and only logarithmic on the number of possible decision boundaries. Most importantly, this is independent from the number of data rows as long as new rows do not add new features, possible thresholds, or class values.

The number of constraints required by our formulation is bounded by $N + R + L + 2 \cdot N \cdot F \cdot T + L \cdot C$. This bound simply derived from the formulation given above. To see that the two $\forall_{n,f,bin(f)}$ and $\forall_{n,f}$ results in at most $2 \cdot N \cdot F \cdot T$ constraints one only has to observe that we create a single such constraint for every branch of the binary search tree given in Figure 1, plus one for the minimum and maximum value ranges. This creates 2 constraints for every possible decision threshold (actually -1 because we combine the minimum and maximum ranges). Frequently, the $2 \cdot N \cdot F \cdot T$ will be the largest term. This term is however an over estimation and because the number of constraints is different for every fea-

Symbol	Type	Definition
n	index	internal (non-leaf) node in the tree, $1 \leq n \leq N$
l	index	leaf of the tree, $1 \leq l \leq L$
r	index	row in the training data, $1 \leq r \leq R$
f	index	feature in the training data, $1 \leq f \leq F$
c	index	class in the training data,$1 \leq c \leq C$
$bin(f)$	set	feature f's binary encoding ranges
$lr(b)$	set	rows with values in b's lower range, $b \in bin(f)$
$ur(b)$	set	rows with values in b's upper range
$tl(b)$	set	$t_{n,t}$ variables for b's ranges
$ll(n)$	set	node n's leaves under the left branch
$rl(n)$	set	node n's leaves under the right branch
K	constant	the tree's depth
$N = 2^K - 1$	constant	the number of internal nodes (not leaves)
$L = 2^K$	constant	the number of leaf nodes
F	constant	the number of features
C	constant	the number of classes
R	constant	the number of training data rows
T	constant	the total number of threshold values
T_f	constant	number of threshold values for feature f
$T_{\max}$	constant	maximum of T_f over all features f
V_r^f	constant	feature f's value in training data row r
C_r	constant	class value in training data row r
$\min_{t(f)}$	constant	feature f's minimum threshold value
$\max_{t(f)}$	constant	feature f's maximum threshold value
M	constant	minimized big-M value
$f_{n,f}$	binary	node n's selected feature f
$t_{n,t}$	binary	node n's selected threshold t
$p_{l,c}$	binary	leaf l's selected prediction class c
$e_{l,c}$	continuous	error for rows with class c in leaf l
$l_{r,l}$	continuous	row r reaches leaf l

Table 1: Summary of notation used in the encoding.

ture. For example, if a feature only has two possible values (coming from a one-hot encoding for instance), then there is only a single decision threshold and we require only a single constraint to model the node's behavior for that feature (only the minimum and maximum value ranges). Sometimes R will be the dominating term, i.e., when the training data set is large and the number of possible thresholds is small.

As table 3 shows, our formulation results in very few binary decision variables. Even for problems with thousands of rows, we only require a few hundred binaries.

Method	# decision variables	# constraints
BinOCT	$O(2^K(F+C+log(T_{max})))$	$O(R+2^K(F\cdot T_{all}+C))$
DTIP	$O(R\cdot K)$	$O(R\cdot 2^{K-1})$
OCT	$O(R\cdot 2^K)$	$O(2^{K-1}(R\cdot K+C))$

Table 2: The number of decision variables and constraints used in three methods: our method BinOCT, DTIP in (Verwer and Zhang 2017), and OCT in (Bertsimas and Dunn 2017). R is the number of data rows, F number of features, C number of classes, K depth of tree. T_{max} is the maximum number of thresholds for any feature and T_{all} is the number of all decision thresholds over all features.

	binaries	rows	columns
Balance-scale	153	839	5161
Car	184	1197	12600
Iris	183	1444	1431
Statlog-sat.	741	65284	36309

Table 3: Sizes of depth 4 training problems (containing 50% of the data) reported by Cplex.

Dataset	R	F	C	$T_{\max}$	T_{all}
Balance-scale	625	4	3	4	16
Bank marketing 10%	4521	17	2	799	1690
Banknote-authentification	1372	4	2	624	1855
Car-evaluation	1728	5	4	3	14
Ionosphere	351	34	2	94	2312
Iris	150	4	2	23	56
Monks-problems-1	124	6	2	3	11
Monks-problems-2	169	6	2	3	11
Monks-problems-3	122	6	2	3	11
Pima-Indians-diabetes	768	8	2	309	857
Qsar-biodegradation	1055	41	2	437	4178
Seismic-bumps	2584	18	2	311	1120
Spambase	4601	57	2	1174	8006
Statlog-satellite	4435	36	6	80	2217
Tic-tac-toe-endgame	958	18	2	1	18
Wine	178	13	3	70	710

Table 4: The datasets used in the experiments.

Experiments

To solve classification problems, given a training dataset, We formulate the learning problem using the proposed formulation. The resulting BinOCT model is passed to the optimization solver CPLEX 12.8.0, running on an AMD Ryzen machine with 16GB RAM, which returns the best solution (i.e., a classification tree with highest accuracy) it can find within the given time limit. We provide CPLEX with a priority order such that variables closer to the root of the tree get solved first. We test our method on benchmark datasets from the UCI machine learning repository (Lichman 2013). We compare the performance of BinOCT with OCT (Bertsimas and Dunn 2017) and DTIPs (Verwer and Zhang 2017), two recently proposed ILP-based classification algorithms. We use the accuracy in the cited papers for comparison. We also run CART from sciki-learn with its default parameter setting (i.e., criterion=gini, splitter=best, min samples split=2, min samples leaf=1, max leaf nodes=None), except that the maximum depths of the trees generated by CART are set to the same depths as BinOCT. In addition, as in (Bertsimas and Dunn 2017; Verwer and Zhang 2017), BinOCT is solved by CPLEX with warm starts learned from CART to investigate whether this helps find better solutions. We name it BinOCT*.

In order to compare to OCT, we use the same experiment settings as in (Bertsimas and Dunn 2017). For a given dataset, 50% of the dataset are used for training and 25% for testing. As we do not have hyperparameters to tune in our model, the remaining 25% are not used. The split is down randomly five times. We report the average performance of five experiments for each dataset. We chose datasets (see Table) containing no missing values since it is not clear how the authors of (Bertsimas and Dunn 2017) pre-processed the datasets with missing values.

The time limit for BinOCT is set to 10 minutes for each instance. In comparison, OCT used 30 minutes to 2 hours to solve each instance in (Bertsimas and Dunn 2017), and DTIPs was run at most 30 minutes in (Verwer and Zhang 2017). We learn trees with depths ranging from 2 to 4.

Our implemented models, the code, and the used training and testing data sets are available online at https://github.com/SiccoVerwer/binoct.

Results

We tested our method on 16 datasets. The number of data rows in these datasets range from 124 to 4601, and the number of features from 4 to 57.

In Table 5, we report the accuracy on the training data. As explained earlier, each dataset was split randomly 5 times. The accuracy value of BinOCT and CART in the table is the average accuracy on five training sets. Since DTIPs in (Verwer and Zhang 2017) used the complete dataset for training, we also include in the table the reported CART results from (Verwer and Zhang 2017). The training results of OCT are absent in (Bertsimas and Dunn 2017).

BinOCT and BinOCT* found the optimal trees when the problem size is small. For learning trees of depth 2, they returned the optimal trees on 9 out 16 datasets. On small datasets such as Iris and Wine, the learned trees from BinOCT and BinOCT* are optimal even with depth 4. Our method nearly always outperforms CART, no matter whether it is fed with starting solutions from CART. The depth 3-4 instances of Statlog-sat are too hard to solve within 10 minutes. These problems have many features, classes,

Dataset	BinOCT	BinOCT* k=2	CART/R	DTIPs	BinOCT	BinOCT* k=3	CART/R	DTIPs	BinOCT	BinOCT* k=4	CART/R	DTIPs
Balance-scale	**73.3***	**73.3***	71.7		**79.2**	78.7	76.5		**84.8**	84.1	82.9	
Bank market. 10%	**90.3**	**90.3**	89.9/90.1	90.1	90.4	**90.9**	90.7/90.4	90.6	90.6	**91.8**	91.6/91.2	91.3
Banknote-auth.	**93.4***	**93.4***	91.7		**97.8**	97.7	94.6		99.4	**99.7**	97.4	
Car-evaluation	**76.9***	**76.9***	**76.9***		**80.5**	80.4	79.0		85.3	**85.7**	84.2	
Ionosphere	91.1	**91.2**	91.0		94.3	**94.9**	93.8		96.8	**97.1**	96.0	
Iris	**96.8***	**96.8***	**96.8***/96*	96*	**100***	**100***	98.1/97.3	99.3*	**100***	**100***	**100***/99.3	100*
Monks-probl-1	**83.5***	**83.5***	76.8		**92.6***	**92.6***	81.6		**99.4**	**99.4**	86.1	
Monks-probl-2	**69.8***	**69.8***	65.2		**79.5**	**79.5**	70.0		86.7	**86.9**	79.8	
Monks-probl-3	**93.8***	**93.8***	**93.8***		**95.7***	**95.7***	94.8		97.7	**98.0**	95.7	
PI-diabetes	**79.3**	**79.3**	77.3/77.2	77.7	**81.6**	81.3	78.9/77.6	79.4	83.0	**84.7**	82.9/79.3	82.6
Qsar-biodeg.	80.9	**81.2**	80.5		83.9	**85.8**	85.3		84.6	**89.1**	88.7	
Seismic-bumps	**93.5**	93.4	93.1		**93.7**	**93.7**	93.4		93.7	**94.2**	93.9	
Spambase	85.6	**86.7**	86.0		86.1	**90.2**	89.6		84.8	**91.9**	91.6	
Statlog-sat.	**68.8**	66.7	63.2		72.7	**80.5**	78.7		66.5	**81.6**	81.6	
Tic-tac-toe	**72.1***	**72.1***	**71.2***		**79.2**	77.6	75.4		85.2	**85.3**	84.4	
Wine	**97.3***	**97.3***	95.7		**100***	**100***	99.3		**100***	**100***	**100***	

Table 5: Training accuracy of BinOCT, BinOCT* (BinOCT with CART as starting solutions), CART, R (CART in (Verwer and Zhang 2017), DTIPs. The symbol * next to the values means that the solutions are optimal. The best performing method at same depths is marked in bold.

Dataset	BinOCT	BinOCT* k=2	CART/R	OCT	BinOCT	BinOCT* k=3	CART/R	OCT	BinOCT	BinOCT* k=4	CART/R	OCT
Balance-scale	69.3	**69.3**	67.5/64.5	67.1	**73.4**	71.3	70.6/70.4	68.9	**79.6**	78.9	77.5/73.4	71.6
Bank market. 10%	**90.3**	**90.3**	88.9/		88.4	88.5	**88.8**/70.4		**88.5**	**88.5**	**88.5/**	
Banknote-auth.	**91.7**	**91.7**	90.6/89.0	90.1	96.2	**96.6**	93.6/89.0	89.6	97.7	**98.1**	95.8/89.0	90.7
Car-evaluation	**77.8**	**77.8**	**77.8**/73.7	73.7	79.9	**80.4**	78.9/77.4	77.4	85.2	**86.5**	84.8/78.8	78.8
Ionosphere	88.6	87.7	87.7/87.8	**87.8**	87.0	85.5	86.4/87.8	**87.6**	86.8	88.6	87.5/**87.8**	87.6
Iris	**96.3**	95.8	95.8/92.4	92.4	96.3	**97.9**	95.8/92.4	93.5	96.3	**98.4**	97.9/92.4	93.5
Monks-prob-1	**80.0**	**80.0**	68.4/57.4	67.7	**83.2**	80.0	76.8/65.8	70.3	85.8	**87.1**	74.2/68.4	74.2
Monks-prob-2	58.1	54.4	54.0/**60.9**	60.0	59.5	55.3	56.7/**60.9**	60.0	61.4	**63.3**	**63.3**/62.8	54.0
Monks-prob-3	93.5	93.5	93.5/**94.2**	**94.2**	85.2	89.7	92.3/**94.2**	**94.2**	87.7	84.5	93.5/**94.2**	**94.2**
PI-diabetes	**75.4**	75.3	74.7/71.9	72.9	73.1	**74.4**	73.3/70.6	71.1	72.8	73.0	**73.9**/71.7	72.4
Qsar-biodeg.	**78.6**	78.1	76.8/76.4	76.1	79.6	**81.3**	80.2/78.5	78.6	80.0	81.0	**81.9**/79.6	79.8
Seismic-bumps	**93.9**	93.8	94.0/93.3	93.3	**93.6**	92.8	93.1/93.3	93.3	93.6	92.6	92.6/**93.3**	**93.3**
Spambase	85.3	**85.7**	85.4/84.2	84.3	85.8	**88.9**	88.5/86.0	86.0	84.8	89.5	**89.7**/86.0	86.1
Statlog-sat.	**67.5**	65.7	63.4/63.2	63.2	71.6	**79.2**	77.3/77.7	77.9	65.9	**79.9**	**79.9**/78.2	78.0
Tic-ta-toe	67.3	67.3	67.2/68.5	**69.6**	72.8	70.6	73.8/73.1	**74.1**	77.7	78.8	**80.1**/74.2	73.3
Wine	90.7	91.1	88.0/81.3	**91.6**	88.0	92.0	88.0/80.9	**94.2**	85.8	89.8	88.9/80.9	**94.2**

Table 6: Testing accuracy of BinOCT, BinOCT*, CART, R (CART in (Bertsimas and Dunn 2017)) and OCT.

and possible threshold values. The performance of BinOCT and BinOCT* are comparable when learning tees of depths 2 and 3. When the model becomes large (depth 4), it is more beneficial to have a starting solution. DTIPs was also able to return optimal solutions on Iris for trees of different depths. The performance of our methods on the other two datasets is consistently higher than DTIPs, although we ran the instances much shorter than theirs (10 instead of 30 minutes).

In Table 6 we report the testing accuracy. For depth 2 and 3, BinOCT outperforms the other methods on many problem instances. Interestingly, it does not outperform OCT when it finds 100% accurate models on the training data for the Wine instances. On depth 4 problems, the solutions found by BinOCT are frequently outperformed by CART although it's training accuracy is always better. This shows the strength of CART and OCT in making a trade-off between accuracy and model complexity. By purely maximizing accuracy on the training data, BinOCT is essentially overtraining. Although necessary, this trade-off makes it hard to compare the quality of the different formulations. A different trade-off decision creates a different objective function and thus a different problem to solve. The different cross-validation folds and dissimilar CART performance Overall make this comparison even harder. The best we can do is compare the improvement over CART. For depth 2 and 3, BinOCT or BinOCT* clearly outperforms CART, but so does OCT. For the depth 4 instances, OCT's performance was very close or worse than their CART results. BinOCT* gives slightly better results than CART overall, but sometimes worse due to overfitting. Overall, the results are impressive as we ran only 10 minutes to achieve performance often better than OCT, which was run for 2 hours.

Conclusion

We propose an efficient encoding of learning classification trees using a binary linear program formulation, where the numbers of decision variables and constraints are much smaller than those in the state-of-the-art formulations. Importantly, the size of the decision variables used in our model BinOCT is independent from the size of datasets. The advantage of this independence has been demonstrated through a set of experiments. BinOCT, with or without starting solutions, gave overall better solutions than the existing formulations OCT and DTIPs, despite the fact that the running time of BinOCT was much shorter than OCT and DPIPs. In the future, we plan to extend our model with different learning objectives, such as adding fairness criteria. In addition, we will add a trade-off between accuracy and model complexity to the objective function. Lastly, we will investigate further improving the formulation by reducing the number of constraints, or using approximation strategies such as selecting a subset of data rows or possible threshold values.

Acknowledgements

The work is partially supported by the NWO funded project Real-time data-driven maintenance logistics (project number: 628.009.012).

References

Bennett, K. P., and Blue, J. A. 1996. Optimal decision trees. Technical report, R.P.I. Math Report No. 214, Rensselaer Polytechnic Institute.

Bennett, K. P., and Mangasarian, O. L. 1993. Bilinear separation of two sets inn-space. *Computational Optimization and Applications* 2(3):207–227.

Bennett, K. P., and Parrado-Hernández, E. 2006. The interplay of optimization and machine learning research. *Journal of Machine Learning Research* 7:1265–1281.

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

Bertsimas, D., and Shioda, R. 2007. Classification and regression via integer optimization. *Operations Research* 55(2):252–271.

Bessiere, C.; Hebrard, E.; and OSullivan, B. 2009. Minimising decision tree size as combinatorial optimisation. In *International Conference on Principles and Practice of Constraint Programming*, 173–187. Springer.

Blanquero, R.; Carrizosa, E.; Molero-Rıo, C.; and Morales, D. R. 2018a. Optimal randomized classification trees.

Blanquero, R.; Carrizosa, E.; Molero-Rıo, C.; and Morales, D. R. 2018b. Sparsity in optimal randomized classification trees.

Breiman, L.; Friedman, J.; Olshen, R.; and Stone, C. 1984. *Classification and regression trees*. Wadsworth International Group.

Chang, M.; Ratinov, L.; and Roth, D. 2012. Structured learning with constrained conditional models. *Machine Learning* 88(3):399–431.

Dash, S.; Günlük, O.; and Wei, D. 2018. Boolean decision rules via column generation. *arXiv preprint arXiv:1805.09901*.

De Raedt, L.; Guns, T.; and Nijssen, S. 2010. Constraint programming for data mining and machine learning. In *Proceedings of the Twenty-Fourth AAAI Conference on Artificial Intelligence (AAAI-10)*, 1671–1675.

Firat, M.; Crognier, G.; Gabor, A. F.; Hurkens, C.; and Zhang, Y. 2018. Constructing classification trees using column generation. *arXiv preprint arXiv:1810.06684*.

Flach, P. 2012. *Machine learning: the art and science of algorithms that make sense of data*. Cambridge University Press.

Hyafil, L., and Rivest, R. L. 1976. Constructing optimal binary decision trees is np-complete. *Information Processing Letters* 5(1):15 – 17.

Lichman, M. 2013. UCI machine learning repository.

Narodytska, N.; Ignatiev, A.; Pereira, F.; Marques-Silva, J.; and RAS, I. S. 2018. Learning optimal decision trees with sat. In *IJCAI*, 1362–1368.

Norouzi, M.; Collins, M. D.; Johnson, M.; Fleet, D. J.; and Kohli, P. 2015. Efficient non-greedy optimization of decision trees. In *Proceedings of the 28th International Conference on Neural Information Processing Systems*, NIPS'15, 1729–1737. Cambridge, MA, USA: MIT Press.

Quinlan, J. R. 1986. Induction of decision trees. *Machine learning* 1(1):81–106.

Rhuggenaath, J.; Zhang, Y.; Akcay, A.; Kaymak, U.; and Verwer, S. 2018. Learning fuzzy decision trees using integer programming. In *2018 IEEE International Conference on Fuzzy Systems*.

Silva, A. P. D. 2017. Optimization approaches to supervised classification. *European Journal of Operational Research* 261(2):772–788.

Uney, F., and Turkay, M. 2006. A mixed-integer programming approach to multi-class data classification problem. *European Journal of Operational Research* 173(3):910–920.

Verwer, S., and Zhang, Y. 2017. Learning decision trees with flexible constraints and objectives using integer optimization. In *International Conference on AI and OR Techniques in Constraint Programming for Combinatorial Optimization Problems*, 94–103. Springer.

Verwer, S.; Zhang, Y.; and Ye, Q. C. 2017. Auction optimization using regression trees and linear models as integer programs. *Artificial Intelligence* 244:368–395.