Bachelor Thesis Econometrics & Operations Research

Erasmus University Rotterdam

Erasmus School of Economics

Trying to improve optimal classification trees through boundary margin maximization

Abstract

Traditionally, decision trees are constructed by greedy top-down search algorithms that recursively partition the feature space and find locally optimal decision rules. Recently, Bertsimas and Dunn [2017] introduced *optimal classification trees*, a new Mixed-Integer Optimization problem to solve decision trees to global optimality to better capture the underlying relations between the independent variables and the class variable. Moreover, the model can easily be adjusted to allow for multivariate splits. In this paper, we verify whether their results can be replicated. In the process, a few imperfections in their approach are detected and solved. In addition, we investigate whether maximizing the margins of the decision boundaries with support vector machines in both univariate and multivariate optimal trees can improve out-of-sample accuracy. The performance of all models is evaluated on 17 datasets from the UCI Machine Learning Repository. The results show that contrary to Bertsimas and Dunn [2017], univariate and multivariate optimal trees do not outperform recursive CART consistently at depths 1-3. In addition, maximizing the boundary margins in optimal classification trees may have a slightly positive effect on accuracy depending on the dataset, but provides no consistent or substantial improvements.

Author:
Tijs Olde

Student ID:
541582

Supervisor:
Dr. Riley Badenbroek

Second Assessor:
Dr. Twan Dollevoet

July 3, 2022

Contents

1 Introduction

Decision trees are a commonly used tool in operations research, data analysis and machine learning. Starting at the top, a series of decisions based on the characteristics of the data guide the researcher along a path down the tree, to finally arrive at the bottom in one of the leaf nodes. In case of a regression problem, the outcome is a prediction for a continuous target variable. For classification problems, the outcome is the predicted class. Figure 1 shows a simple example that predicts whether Covid-19 will be life-threatening or not if a particular person gets infected.

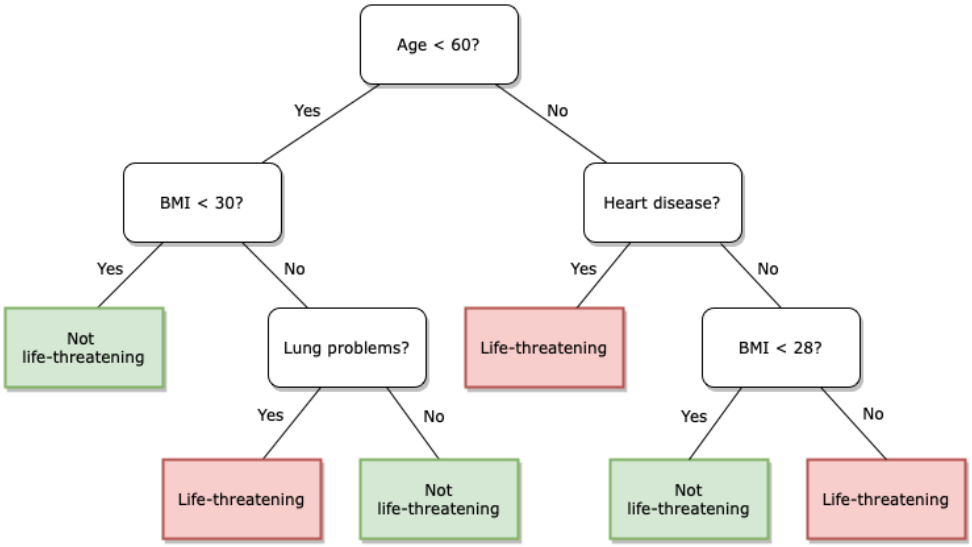

Figure 1: Decision tree example: will Covid-19 be life-threatening or not?

Decision trees have attained great popularity because they are intuitive and interpretable: every decision in the sequence leading to the final outcome can easily be understood. In addition, they don't make assumptions about the distribution of the data, they can model interactions between variables in a natural way and are not sensitive to outliers. Constructing a decision tree however is a much more complex task than using it. That is why in the last 60 years, a wide variety of decision tree algorithms have been proposed, the majority of which are based on recursive partitioning of the feature space.

This approach, however, yields trees that are only locally optimal, not globally. The possibility to solve decision trees to optimality has been known since Breiman et al. [1984], but it was deemed infeasible due to its enormous computational expenses. In light of the massive improvements in computational power over the last three decades, Bertsimas and Dunn [2017] decide that it is time to solve decision trees to optimality. They formulate it as a Mixed-Integer Optimization (MIO) problem and conclude that optimal univariate trees provide improvements over CART of around $0.5-2\%$ in out-of-sample average accuracy, while for optimal multivariate trees, this is between $3-5\%$, both depending on tree depth. This is measured on 53 datasets

and depths 1-4. In this paper, we aim to assess whether this approach indeed gives the results Bertsimas and Dunn claim they do. In addition, we want to explore whether their model can be further improved by maximizing the boundary margins. Therefore, the research questions are formulated as follows:

1. Can the results of Bertsimas and Dunn [2017] be validated in a replication study?
2. Do the performances of optimal univariate and multivariate optimal decision trees increase when the margins of the decision boundaries are maximized?

The first question is relevant in light of the trustworthiness of science: if the results are reliable, they should be reproducible. The second question is relevant because if that is the case, out-of-sample prediction accuracy can be increased without losing the interpretability of the model.

We use 17 out of 53 datasets Bertsimas and Dunn select from the UCI Machine Learning Repository to evaluate model performance. Our main contributions are that we correct the optimal univariate tree formulation by adding a constraint to avoid points from going to the left and right branch while no split is applied. Secondly, we provide a formal proof for a constant that is used in the model formulation. We formulate an adapted support vector machine model that finds boundaries that maximize margins after the optimal tree is obtained.

Our main findings are that only multivariate optimal trees with depth 1 consistently outperform CART. Overall, CART beats univariate optimal trees and multivariate optimal trees with depth 2 and 3. This means that Bertsimas and Dunn [2017] appear to be overambitious in their claims. In addition, maximizing boundary margins in optimal trees does not provide consistent or substantial improvements in out-of-sample accuracy, although positive effects are found for several combinations of depth and dataset.

This paper is organized as follows. In the next section, we will give an overview of the relevant literature. In Section 3, we will present the two optimal classification tree models in Bertsimas and Dunn [2017] and the procedure to obtain such trees with hyperparameter optimization. In Section 4, the models used to maximize the boundary margins for the optimal classification trees will be discussed. The results can be found in Section 5. Finally, in Section 6, we will summarize our findings and give recommendations for future research.

2 Literature review

For a tree to classify as a decision tree, decisions that have to be made at one stage have to depend on decisions made at a previous stage. From this point of departure, the roots of decision tree methods trace back to 1959, when William Belson considered a sequential, non-symmetrical "biological classification" [Belson, 1959]. From then on, researchers have proposed

various extensions and modifications.

We refer to Morgan and Sonquist [1963], who invented the AID algorithm (Automatic Interaction Detection), the first regression tree algorithm. It builds a binary and non-symmetrical tree in a stepwise, recursive procedure. At each step, for every possible split the impurity of the two child nodes are calculated, which is defined as the sum of squared deviations from the mean. The selected split minimizes the sum of the impurities from both child nodes. A novelty is that the authors include a stopping criterion: when the decrease in impurity is less than a prespecified fraction of the impurity at the parent node, recursion should stop. Messenger and Mandell [1972] adapted this algorithm for classification usage in their THAID algorithm (THeta Automatic Interaction Detection). They also propose other impurity functions, specifically the Gini index and the entropy, which would be used extensively in later methods.

However, at this stage, these methods suffered from various childhood illnesses, such as severe overfitting Einhorn [1972] and the inability to handle highly correlated variables appropriately [Doyle, 1973]. To overcome some of these issues, Kass [1980] created CHAID (Chi-Square AID), a modification of AID in which the stop condition and split variable for each step are determined based on the significance of chi-square test statistics and Bonferonni corrected p-values.

The ideas from AID and CHAID are carried further in Breiman et al. [1984], who developed the CART algorithm (Classification And Regression Trees). CART can be applied to both regression and classification and uses the Gini index as the node impurity criterion. It also consists of a top-down greedy search approach, but here, the tree is first grown as deep as possible. Then, pruning is applied by deleting nodes that contribute the least to error reduction in a crossvalidation procedure. This avoids the problem of ignoring powerful splits that are hidden behind weaker splits. With CART, the overfitting issues present in earlier methods is largely solved. In addition, Breiman et al. [1984] propose a way to split on a linear combination of variables to obtain multivariate trees.

A similar recursive top-down algorithm is proposed by Quinlan [1986], called ID3 (Iterative Dichotomiser 3). Compared to CART, ID3 does not apply pruning, but relies on stopping conditions for each subset. It can only be used with categorical independent variables. ID3 can build multiway trees, as opposed to CART, for which nodes in the tree can only have two or zero children. Ross Quinlan improved this algorithm in Quinlan [2014], calling it C4.5, which uses a certain gain ratio to determine splits. C4.5 also supports continuous variables, a heuristic method to prune trees after creation and ways to handle missing values appropriately. Both ID3 and C4.5 are only suited for classification.

Despite being very popular, greedy search algorithms, which includes all algorithms discussed

so far, face some major downsides. First of all, there is bias in the variable selection of splits. The selection procedure is biased towards variables with more distinct values, because more splits are possible for these variables. This is true for continuous and categorical variables, but for the latter, the difference in possible splits grows a lot faster (linearly versus exponentially). Doyle [1973] is the first to discuss this problem in the context of AID and THAID. Breiman et al. [1984] mentioned that it also exists in CART, while Loh and Shih [1997] established that this issue tends to arise in all exhaustive search methods. From White and Liu [1994] and Kononenko [1995], it follows that essentially all popular split measures, including entropy, Gini index, information gain and other information-based measures, are biased.

A second major disadvantage is that the trees resulting from greedy top-down algorithms are only locally optimal, not globally, as only in each step a local optimal decision is made [Bertsimas and Dunn, 2017]. Each split is determined in isolation, without considering the possible impact of future splits. As a result, it could be that the tree is not able to capture the underlying relations between the independent variables and the class or target variable. An optimal tree would require that each split is determined with full knowledge of all other splits, such that instead of recursion, the decision tree is built in one step. Breiman et al. [1984] already mentioned this possibility, but rendered it impractical due to technological limitations in computer speed, as this problem is NP-hard [Laurent and Rivest, 1976].

To address the selection bias, researchers in the more statistical branch of tree research have built algorithms that apply statistical tests at each branch node to decide on the splitting variable. However, we will not address this issue in this paper, as Bertsimas and Dunn [2017] focus on the second challenge. Their aim is to resolve the local optimality problem with globally optimal univariate and multivariate classification trees. These models will be the main subject of this paper.

3 Optimal classification tree models

In their paper, Bertsimas and Dunn discuss two models: one for univariate optimal classification trees (OCT) and one for multivariate optimal classification trees (OCT-H). In OCT-H, a split rule can be based on a linear combination of features, instead of only one feature in OCT. This is called a hyperplane split and it explains the "H". Technically, a univariate split based on one variable also produces a hyperplane in the feature space, but the intuition behind the naming is clear. Contrary to typical decision trees for which scaling is not necessary, in these models, the feature values for all observations are normalized to the $[0, 1]$ interval for each feature. In this section, we will introduce the OCT and OCT-H model formulations.

First of all, in Table 1 all sets, parameters and variables used in Bertsimas and Dunn [2017] and their interpretation are listed to clarify the notation that is needed in the formulation.

Table 1: Overview of sets, parameters and variables

Entity	*Interpretation*	*Domain*
Sets		
$\mathcal{T}_B$	branch nodes, $\mathcal{T}_B = \{1, ..., 2^{D-1}\}$	
$\mathcal{T}_L$	leaf nodes, $\mathcal{T}_L = \{2^D, ..., 2^{D+1} - 1\}$	
$\mathcal{A}_L(t)$	ancestor nodes of leaf node t whose left branch is followed in the path from the root node to t	$t \in \mathcal{T}_L$
$\mathcal{A}_R(t)$	ancestor nodes of leaf node t whose right branch is followed in the path from the root node to t	$t \in \mathcal{T}_L$
Parameters		
D	depth of the tree	
K	number of distinct classes	
p	number of features	
n	number of observations/points	
$N_{\min}$	minimum number of points in a leaf node	
α	complexity parameter	
$\mathbf{x}_i$	feature vector for observation i	$i = 1, ..., n$
Y_{ik}	1 if observation i belongs to class k, -1 otherwise	$i = 1, ..., n$, $k = 1, ..., K$
$\hat{L}$	baseline accuracy: accuracy when all observations are assigned to the majority class	
Variables		
$p(t)$	parent node of node t	$t = 2, ..., 2^{D+1} - 1$
N_t	total number of points in leaf node t	$t \in \mathcal{T}_L$
N_{kt}	number of points of label k in leaf node t	$k = 1, ..., K$, $t \in \mathcal{T}_L$
L_t	number of points that are misclassified in leaf node t	$t \in \mathcal{T}_L$
b_t	threshold value for a split at branch node t	$t \in \mathcal{T}_B$
Decision variables		
a_{jt}	1 if node t splits on feature j, 0 otherwise	$j = 1, ..., p$, $t \in \mathcal{T}_B$
d_t	1 if node t applies a split, 0 otherwise	$t \in \mathcal{T}_B$
z_{it}	1 if observation i ends up in leaf node t, 0 otherwise	$i = 1, ..., n$, $t \in \mathcal{T}_L$
l_t	1 if leaf node t contains any observations, 0 otherwise	$t \in \mathcal{T}_L$
c_{kt}	1 if leaf node t predicts class k, 0 otherwise	$k = 1, ..., K$, $t \in \mathcal{T}_L$

3.1 Univariate OCT

The complete OCT model is formulated as follows:

$$\min \quad \frac{1}{\hat{L}} \sum_{t \in \mathcal{T}_L} L_t + \alpha \sum_{t \in \mathcal{T}_B} d_t \tag{1}$$

$$\text{s.t.} \quad L_t \geq N_t - N_{kt} - n(1 - c_{kt}), \qquad k = 1, ..., K, \ \forall t \in \mathcal{T}_L, \tag{2}$$

$$L_t \leq N_t - N_{kt} + n c_{kt}, \qquad k = 1, ..., K, \ \forall t \in \mathcal{T}_L, \tag{3}$$

$$L_t \geq 0, \qquad \forall t \in \mathcal{T}_L, \tag{4}$$

$$N_{kt} = \frac{1}{2} \sum_{i=1}^{n} (1 + Y_{ik}) z_{it}, \qquad k = 1, ..., K, \ \forall t \in \mathcal{T}_L, \tag{5}$$

$$N_t = \sum_{i=1}^{n} z_{it}, \qquad \forall t \in \mathcal{T}_L, \tag{6}$$

$$\sum_{k=1}^{K} c_{kt} = l_t, \qquad \forall t \in \mathcal{T}_L, \tag{7}$$

$$\mathbf{a}_m^T (\mathbf{x}_i + \boldsymbol{\varepsilon}) \leq b_m + (1 + \varepsilon_{\max})(1 - z_{it}), \qquad i = 1, ..., n, \ \forall t \in \mathcal{T}_L, \ \forall m \in \mathcal{A}_L(t), \tag{8}$$

$$\mathbf{a}_m^T \mathbf{x}_i \geq b_m - (1 - z_{it}), \qquad i = 1, ..., n, \ \forall t \in \mathcal{T}_L, \ \forall m \in \mathcal{A}_R(t), \tag{9}$$

$$\sum_{t \in \mathcal{T}_L} z_{it} = 1, \qquad i = 1, ..., n, \tag{10}$$

$$z_{it} \leq l_t, \qquad i = 1, ..., n \ \forall t \in \mathcal{T}_L, \tag{11}$$

$$\sum_{i=1}^{n} z_{it} \geq N_{\min} l_t, \qquad \forall t \in \mathcal{T}_L, \tag{12}$$

$$\sum_{j=1}^{p} a_{jt} = d_t, \qquad \forall t \in \mathcal{T}_B, \tag{13}$$

$$l_t \leq d_m, \qquad \forall t \in \mathcal{T}_L, \ m \in \mathcal{A}_L(t), \tag{14}$$

$$0 \leq b_t \leq d_t, \qquad \forall t \in \mathcal{T}_B, \tag{15}$$

$$d_t \leq d_{p(t)}, \qquad \forall t \in \mathcal{T}_B \setminus \{1\}, \tag{16}$$

$$z_{it}, l_t \in \{0, 1\}, \qquad i = 1, ..., n, \ \forall t \in \mathcal{T}_L, \tag{17}$$

$$a_{jt}, d_t \in \{0, 1\}, \qquad j = 1, ...p, \ \forall t \in \mathcal{T}_B, \tag{18}$$

$$c_{kt} \in \{0, 1\}, \qquad k = 1, ..., K, \ t \in \mathcal{T}_L. \tag{19}$$

The objective function (1) minimizes the sum of the misclassification cost, normalized by the baseline accuracy $\hat{L}$, and the total number of splits, weighted by the complexity parameter α. α regulates the trade-off between accuracy and complexity: the higher its value, the fewer splits will be applied at the expense of more accuracy, resulting in smaller trees. Restriction sets (2) - (4) follow from the linearization of $L_t = N_t - \max_{k=1,...,K}\{N_{kt}\}$. Restrictions (5) and (6) set N_{kt} and N_t to the number of points of label k in each leaf node t and the total number of

points in each leaf node, respectively. Restrictions (7) ensure that if a leaf node t is non-empty, exactly one class is predicted at that node. If the leaf node is empty, no predictions are made. Constraint set (8) and (9) make sure that observations only end up in leaf nodes such that all splits applied on the path from the root node to the leaf node are followed in the correct way, given the feature values. Restrictions (9) ensure this for all splits in the path where the right branch is followed, while (8) does so for the left branch splits. Originally, they stem from the following restrictions:

$$\mathbf{a}_m^T \mathbf{x}_i < b_m + M_1(1 - z_{it}), \qquad i = 1, ..., n, \;\; \forall t \in \mathcal{T}_L, \;\; \forall m \in \mathcal{A}_L(t), \tag{20}$$

$$\mathbf{a}_m^T \mathbf{x}_i \geq b_m - M_2(1 - z_{it}), \qquad i = 1, ..., n, \;\; \forall t \in \mathcal{T}_L, \;\; \forall m \in \mathcal{A}_R(t), \tag{21}$$

However, because MIO solvers do not support strict inequalities, a small constant ε must be added to the left-hand side of (20) to turn it into a less than or equal inequality. Since a value that is too small could cause numerical instabilities, it is beneficial to choose ε as large as possible without changing the feasibility of any valid solution to the problem. This can be done by specifying a different ε_j for every feature j. [Bertsimas and Dunn, 2017, p. 1048] claim that the largest valid value of ε_j is equal to the smallest non-zero distance between adjacent values of this feature. They do not provide a proof, although this is by no means a trivial statement. We will formally prove in Section 3.2 that their statement is true. M_1 and M_2 are sufficiently large constants that make the constraints inactive if observation i is not in leaf node t. Since $\mathbf{a}_m^T \mathbf{x}_i, b_m \in [0, 1]$, setting $M_1 = 1 + \max_{j=1,...,p}\{\varepsilon_j\} = 1 + \varepsilon_{\max}$ and $M_2 = 1$ will suffice.

Restriction set (10) requires that each observation is assigned to exactly one leaf node. Restrictions (11) ensure that an observation is only assigned to non-empty leaf nodes. Constraints (12) make sure that when a leaf node is non-empty, at least $N_{\min}$ points are assigned to it. Restriction set (13) implies that whenever a split is applied at a branch node t, exactly one feature is used. If no split is applied, all a_{jt} are zero. Restriction set (14) then makes sure that all observations follow the correct branches. This constraint is not included in the model in Bertsimas and Dunn [2017]. We add it to avoid points at a branch node that does not apply a split being allowed to follow either the right or left branch. This is possible, since in that case both the left-hand side and right-hand side of (8) and (9) are zero, meaning that they both hold regardless of whether observations go left or right. In the model without (14), observations following the left branch at a node that does not apply a split would be possible in an optimal solution. This is incorrect. The OCT model in Bertsimas and Dunn [2017] is thus incomplete.

We detected this flaw because without (14), in an optimal solution the values of a_{jt}, b_t and d_t would be zero for all $t \in \mathcal{T}_B$ and $j = 1, ..., p$. The minimization favours a zero d; subsequently, (13) and (15) force a and b to also be zero. Then all model restrictions always hold, in particular

(8) and (9), regardless of the leaf node an observation is assigned to. Therefore, because of the minimization of the misclassification error, the observations could be distributed over the leaf nodes in an optimal way, and in fact z_{it}, l_t, c_{kt}, and then also L_t, N_t and N_{kt} could take on the 'correct' optimal values. With the additional restriction set (14), d_t is forced to 1 if there are observations present in any leaf node in the left subtree of t. This solves the problem.

To continue the explanation of the model, a branch node can only apply a split if its parent applies a split. This is ensured by restrictions (16). Lastly, (17) - (19) are the domain restrictions for the decision variables.

3.2 Proof for the small constant formula in OCT

As mentioned before, this section contains a formal proof to the claim that the largest valid value of ε_j is equal to the smallest non-zero distance between adjacent values of this feature. Suppose that a split $a_m^T x_i \leq b_m$ in branch node m is made using feature j, with $m \in \{1, ..., 2^D - 1\}$ and $j \in \{1, ..., p\}$, with D and p denoting the depth of the tree and the number of features, respectively. Sort the n observations in non-decreasing order of feature j, resulting in the sequence $0 \leq x_j^1 \leq x_j^2 \leq ... \leq x_j^n \leq 1$, where x_j^k denotes the feature j-value of the k-th observation when ordered on feature j. With univariate trees, decision boundaries are orthogonal to the axis corresponding to the feature j the split is based on. Hence, there exists an index s such that $x_j^1 \leq ... \leq x_j^{s-1} \leq x_j^s < b_m$ and $b_m \leq x_j^{s+1} \leq ... \leq x_j^n$. Let $S_L = \{x_j^1, ..., x_j^{s-1}, x_j^s\}$ and $S_R = \{x_j^{s+1}, ..., x_j^n\}$. The observations in S_L should travel into the left subtree of node m, while the observations in S_R should travel into the right subtree of node m.

We have to prove that the largest ε_j such that $0 + \varepsilon_j \leq x_j^1 + \varepsilon_j \leq ... \leq x_j^s + \varepsilon_j \leq b_m$ equals

$$\varepsilon_j^* = \min \{x_j^{i+1} - x_j^i \,|x_j^{i+1} \neq x_j^i,\, i = 1, ..., n-1\}, \tag{22}$$

that is, the smallest non-zero distance in feature j between adjacent observations when placed in non-decreasing order of feature j. To do this, we have to prove that this particular choice of ε_j can turn the strict inequality into a less than or equal inequality. Then, we have to prove that for any $\delta_j^{'} > \varepsilon_j^*$, this is not possible.

First of all, we need to prove the following lemma about the OCT model:

Lemma 1. *For any optimal OCT solution, suppose that branch node m produces a split, which is the case if and only if $d_m = 1$. Let it split on feature j. Then there exists an optimal solution to the problem formulated in Equations* (1) - (7), (9) - (20) *in which the value of b_m equals the smallest feature j-value among all observations in the right subtree of m.*

Proof. First of all, we can prove by contradiction that in any optimal solution $x_j^1 < b_m \leq x_j^n$. Suppose that $b_m \leq x_j^1$. Then a split is applied that would never occur in an optimal solution. In

this case all observations would be sent to the right branch of m. Such a split can never occur in an optimal solution, because the objective value can always be reduced if this split is removed: the number of splits decreases, while the misclassification errors stay the same. For $b_m > x_j^n$, a similar argument holds: all observations would be sent to the left branch, and again a split is applied that would never occur in an optimal solution. Hence, these two cases are impossible in an optimal solution.

Therefore, we know that it in an optimal solution it holds that $b_m \in (x_j^i, x_j^{i+1}]$ for some $i \in \{1, ..., n-1\}$. A change in the value of b_m can only cause the objective value $\frac{1}{\hat{L}} \sum_{t \in T_L} L_t + \alpha \sum_{t \in T_B} d_t$ to change if b_m leaves this interval. This is because for all $b_m \in (x_j^i, x_j^{i+1}]$, whether an observation is sent to the right or left branch of node m does not change. Thus, the distribution of observations over the leaf nodes stays the same, and therefore the number of misclassified points $\sum_{t \in T_L} L_t$ does not change. In addition, because $b_m > 0$, which follows from $b_m > x_j^1 \geq 0$, d_m also remains equal to 1 due to Equations (13). From this, we can conclude that there exists an optimal solution with $b_m = x_j^{i+1}$. Lastly, it needs to hold that $x_j^1 \leq ... \leq x_j^{s-1} \leq x_j^s < b_m$ and $b_m \leq x_j^{s+1} \leq ... \leq x_j^n$. It follows that there exists an optimal solution with $b_m = x_j^{s+1}$. □

With this lemma, we can now prove the following theorem.

Theorem 1. *In the optimal solution in which all b_m-values are equal to the smallest values in the feature they split on among all observations in the right subtree of m, ε^* is the largest value that can be added to the left-hand side of* (20) *without changing the feasibility or optimality of the solution, turning the strict inequality into a less than or equal inequality as in* (8)*.*

Proof. Let $\delta_j = \min \{x_j^{i+1} - x_j^i \,|x_j^{i+1} \neq x_j^i,\, i = 1, ..., n-1\}$. Suppose that $\delta_j \neq x_j^{s+1} - x_j^s$. Then we know by the definition of δ_j that $x_j^s + \delta_j < x_j^{s+1} = b_m$, where Lemma 1 is used to obtain the last equality. This means that we can safely add δ_j to the left-hand side of Equation (20) and turn the strict inequality into a less than or equal to inequality as in Equation (8), as whether this condition is satisfied or not does not change for any of the observations i in any leaf node t in the left subtree of node m.

For $\delta_j = x_j^{s+1} - x_j^s$, which means that the split precisely occurs between the two adjacent x_j-values that are closest to each other, we have $x_j^s + \delta_j = x_j^{s+1} = b_m$ and Equation (8) will have an equality for the observation with the largest feature j-value in S_L. Again, we can conclude that we can safely add δ_j to the left-hand side of Equation (20) and turn the strict inequality into a less than or equal to inequality, as whether this condition is satisfied or not does not change for any of the observations i in any leaf node t in the left subtree of node m.

This proves that setting $\varepsilon_j = \delta_j$, it is indeed possible to turn the strict inequality in Equation (20) into a less than or equal inequality in Equation (8).

To prove that δ_j is really the largest valid value to do so, consider a $\delta_j^{'} > \varepsilon_j^*$. Consider the case $\varepsilon_j^* = x_j^{s+1} - x_j^s$. Since $x_j^s + \delta_j^{'} > x_j^s + \varepsilon_j^* = x_j^{s+1} = b_m$, observation x^s cannot travel into the left subtree of m, as Equation (8) is violated. However, it is also not allowed to travel into the right subtree of m, because $x_j^s \geq b_m$ also does not hold. This is a contradiction with the structure of a binary tree, and hence, ε_j^* really is the largest valid value that we can select. $\square$

This proves that $\varepsilon_j = \min\{x_j^{i+1} - x_j^i \,|x_j^{i+1} \neq x_j^i,\, i = 1, ..., n-1\}$ for every feature $j = 1, ..., p$ is the largest valid value that can be added to the left-hand side of Equation (20) such that there exists an optimal solution to problem (1) - (19) which is also an optimal solution to problem (1) - (7), (9) - (20). This concludes the proof.

3.3 Multivariate OCT-H

As opposed to a univariate decision tree, in a multivariate tree, multiple variables can be used to apply a split. In the OCT-H model, the split condition can be a linear combination of variables. As a result, a few restrictions slightly change and a we have to add a few extra restrictions; the general idea of the model however stays the same. (18) should be relaxed to $-1 \leq a_{jt} \leq 1$ for $j = 1, \ldots, p$, $t \in \mathcal{T}_B$ and the equality in (13) should change into a less than or equal inequality. Because negative values are allowed, we take the absolute value, resulting in:

$$\sum_{j=1}^{p} |a_{jt}| \leq d_t, \qquad \forall t \in \mathcal{T}_B. \tag{23}$$

This can be linearized using auxiliary variables $\hat{a}_{jt}$ to give

$$\sum_{j=1}^{p} \hat{a}_{jt} \leq d_t, \qquad \forall t \in \mathcal{T}_B, \tag{24}$$

$$\hat{a}_{jt} \leq a_{jt}, \qquad j = 1, ..., p, \;\; \forall t \in \mathcal{T}_B, \tag{25}$$

$$-\hat{a}_{jt} \geq -a_{jt}, \qquad j = 1, ..., p, \;\; \forall t \in \mathcal{T}_B. \tag{26}$$

b_t should have the same range as $\mathbf{a}_t^T \mathbf{x}_i$, therefore the left-hand side of (15) changes from 0 to $-d_t$. Because of this change in range, the maximum value of $\mathbf{a}_m^T \mathbf{x}_i - b_m$ in (21) and (20) now becomes 2, which means that M_1 and M_2 have to be set to 2. This time, we can no longer choose the sufficiently small constant in a smart way, as multivariate splits do not necessarily produce boundaries that are orthogonal to the axis. Instead, we introduce the parameter μ and set it equal to 0.005, in the footsteps of Bertsimas and Dunn [2017].

Instead of penalizing the number of splits in the tree, in the multivariate scenario the total number of variables used in all splits is penalized. To achieve this, introduce the new binary variables s_{jt}, which is equal to 1 if feature j is used in a split at branch node t. Consequently, the second summation in the objective function (1) changes to $\alpha \sum_{t \in \mathcal{T}_B} \sum_{j=1}^{p} s_{jt}$. To ensure

the correct relation between the s_{jt} variables and the a_{jt} and d_t variables, we add the following three sets of restrictions:

$$-s_{jt} \leq a_{jt} \leq s_{jt}, \qquad j = 1, ..., p, \;\; \forall t \in \mathcal{T}_B, \tag{27}$$

$$s_{jt} \leq d_t, \qquad j = 1, ..., p, \;\; \forall t \in \mathcal{T}_B, \tag{28}$$

$$\sum_{j=1}^{p} s_{jt} \geq d_t, \qquad \forall t \in \mathcal{T}_B. \tag{29}$$

Restrictions (27) set s_{jt} equal to 1 if feature j is used in the split at branch node t. Restrictions (28) make sure that a feature can only be used in a split at a branch node if the branch node applies a split. Finally, (29) implies that if a split is applied at a particular branch node, at least one feature is used in the split condition. The complete model formulation of the OCT-H model can be found in Appendix A.

3.4 Hyperparameter optimization

To tune the value of the complexity parameter α in an efficient way, we can remove the second summation from the objective function and include it in the restrictions. By minimizing $\frac{1}{\hat{L}} \sum_{t \in \mathcal{T}_L} L_t$ with the additional constraint $\sum_{t \in \mathcal{T}_B} d_t \leq C$ for OCT and $\sum_{t \in \mathcal{T}_B} \sum_{j=1}^{p} s_{jt} \leq C$ for OCT-H, C, unlike α, will be integer-valued. The advantage is that we can only search over critical values of α that would lead to different solutions, such that the number of MIO problems to solve is minimized. As an extra benefit, a solution to the problem for C is a feasible warm start for the problem for $C + 1$. For OCT, C is the maximum number of splits and $C_{\max} = 2^{D_{\max}} - 1$. For OCT-H, C is the maximum number of features used in all splits across the tree, with $C_{\max} = p(2^{D_{\max}} - 1)$. The entire range of different solutions is now the set containing all solutions with $C = 1, ..., C_{\max}$.

Bertsimas and Dunn [2017] now give the following algorithm to find appropriate values for depth D and complexity parameter α:

Algorithm 1 Optimal tree tuning algorithm for α

Require: maximum depth $D_{\max} > 0$, minimum leaf size $N_{\min} > 0$

1: **for** $D = 1, ..., D_{\max}$ **do**
2: **for** $C = 1, ..., C_{\max}$ **do**
3: a) Run CART with $N_{\min}$, but without further pruning. Trim the solution to depth
4: D and a maximum of C splits (OCT) or a maximum of C split features (OCT-H).
5: b) Search through pool of candidate warm starts (including CART) and choose
6: the one with the lowest misclassification error.
7: c) Solve MIO problem for D and C using the selected warm start.
8: d) Add the MIO solution to the warm start pool.
9: **end for**
10: **end for**
11: Remove all solutions from the solution pool that are not optimal for any value of α.
12: Select the solution with the lowest misclassification error on a separate validation set
13: and find the range of α for which this solution optimal on the training set.
14: Use the midpoint of this interval as tuned value for α.

To clarify line 9, the objective value $\frac{1}{\hat{L}} \sum_{t \in \mathcal{T}_L} L_t + \alpha \sum_{t \in \mathcal{T}_B} d_t$ of all solutions in the pool is an affine function in α with intercept $\frac{1}{\hat{L}} \sum_{t \in \mathcal{T}_L} L_t$ and positive slope $\sum_{t \in \mathcal{T}_B} d_t$. If one lies strictly above another, that is, the intercept is higher and the slope is equal or steeper, or strictly above a piece-wise linear combination of other solutions, the former solution is strictly worse for all values of α, such that this one can be eliminated.

Bertsimas and Dunn [2017] also show how to tune $N_{\min}$ by means of a third for loop, but they do not use this in their results. For this reason, neither do we. After the optimal hyperparameter α corresponding to the specified $D_{\max}$ and $N_{\min}$ is found, it can be used to train the final tree on the combined training and validation set. Here again for each depth $1, ..., D_{\max}$, a trimmed CART solution with equal depth serves as a warm start.

3.5 Modelling choices and changes in the hyperparameter optimization

Bertsimas and Dunn [2017] do not provide all the details necessary to reproduce the training algorithm. In this section, we will discuss and motivate our modelling choices, and propose some modifications in order to decrease running times.

3.5.1 CART trimming

Bertsimas and Dunn [2017] suggest in line 3 to first grow a tree until maximum depth, such that all leaf nodes contain at least $N_{\min}$ observations. However, when post-pruning a tree to a given

depth, all splits below this depth have to be removed regardless. That is why we generated CART solutions grown until depth D and no further.

Bertsimas and Dunn do not mention what split pruning criterion is to be used to trim the CART solution. We choose to iteratively remove the split with the lowest relative reduction in Gini impurity [Breiman et al., 1984]. The Gini impurity of a node t is $1 - \sum_{i=1}^{K} p_{it}^2$, with p_{it} the proportion of points from class i in node t compared to the total number of observations in the node. With $l(t)$ and $r(t)$ denoting the left and right child nodes of node $t \in \mathcal{T}_B$, N_t the number of observations in node t and G_t the Gini impurity in node t, the relative reduction in Gini impurity of a particular split is calculated as follows:

$$q(t) = \frac{N_{l(t)}}{N_t} \frac{G_{l(t)} - G_t}{G_t} + \frac{N_{r(t)}}{N_t} \frac{G_{r(t)} - G_t}{G_t}, \qquad \forall t \in \mathcal{T}_B. \tag{30}$$

In this way, removals of splits do not necessarily occur at the deepest level of the tree. In case of a tie between different branch nodes in $q(t)$, we remove the split at the deepest level. The root node is exempted from pruning, as this would just result in a majority class prediction. These trees of depth zero therefore do not have much added value to the warm starts. After a split removal, all observations in the left and right subtree are pushed back to the new leaf node. This procedure is repeated as long as more than C splits are applied or the depth exceeds D.

3.5.2 Duplicate solutions

One disadvantage of this tuning algorithm is that it is guaranteed that across different iterations in the for loop in line 1, duplicate MIO solutions are generated and added to the warm start or solution pool. We define duplicate solutions as trees that apply the same decision rules in the same order for every possible observation. These could be solutions of the same depth, but it can also concern two solutions that do not have the same depth. Duplicate solutions arise because, for any $C < D$, with C the maximum number of splits in OCT, the same problem is already solved in an iteration with a lower D-value. The reason is that it is not allowed for a child node to apply a split if their parent does not apply a split. For example, for depth 2, allowing only one split has to result in a tree with a split at the root node. This is essentially a tree with depth 1, and this has already been solved for $D = 1$. The for loop in line 2 can therefore start with D.

Another trick to speed up computation is to notice that for OCT or OCT-H models with different depths, the solution pool that is generated in the double for loop for the lower depth is a subset of the solution pool for the higher depth model. To take advantage of this, all CART and MIO warm starts for the model with the highest depth can be generated first. For the lower depths, the corresponding subset constitutes the solution pool.

3.5.3 Single solution in the solution pool

Another important issue is that, especially with low depths which do not have a large solution pool, it is possible that after removing duplicate solutions, only one solution in the solution pool remains. For OCT of depth one, it is always true that the solution pool contains only one OCT solution, but it can happen for OCT-H as well. This solution will then in line 12 turn out to be optimal on the interval $[0, \upsilon]$. υ here is the upper bound for α. This is equal to $n/\hat{L}$, the upper bound of the first term in the objective, obtained when all observations are classified incorrectly. For $\alpha > \upsilon$, the objective function is always minimized when zero splits are applied, such that there is no trade-off between accuracy and complexity any more. More generally, this situation occurs when the solution with the best misclassification error on the validation set turns out to be the one with the lowest $\sum_{t \in \mathcal{T}_B} d_t$ or $\sum_{t \in \mathcal{T}_L} \sum_{j=1}^{p} s_{jt}$. This solution will be optimal on the range starting from an intercept with another solution (or zero) until υ.

The problem with taking the midpoint of this interval as tuned value for α is that it will be too large, resulting in trees that are very small. To show this, we can derive that the value of α such that an extra split (or an extra split variable, in OCT-H) is preferred only if the accuracy on the training set increases with more than θ percent of the baseline accuracy because of the extra split is equal to $\theta n_t/100$ (see Appendix C). For $\alpha = n/2\hat{L}$, this means that an extra split in OCT is only allowed if it increases the training set accuracy with $50/\hat{L}$ percent. For most datasets, this will be too high to generate any splits. Consider that the average baseline accuracy across the 17 datasets we use from the UCI Machine Learning Repository is 59.1%. On average, a split is then only allowed if it brings an $50/0.591 \approx 84.6\%$ accuracy increase over the baseline. The corresponding required training set accuracy is $1.846 * 59.1 \approx 109.1$. This can never be achieved.

Instead, if there is only one solution, we choose to select a value of α based on 3-fold cross validation. We combine the training and validation sets and randomly split them into three equal parts. Then, for every α, two samples are used to train the model and one to calculate the test accuracy. The best value of α is the one with the highest average test accuracy across the three iterations. The difficulty lies in the fact that the range of α-values that are reasonable to consider for a particular dataset depends on its size. With larger samples, the complexity parameter has to be increased to avoid overfitting of the model. That is why we consider a range of α-values corresponding to a pre-specified range of θ-values $\{0.01, 0.1, 0.5, 1, 2.5, 5, 7.5, 10, 15, 20, 25, 30\}$. In this way, we can make a more reasonable choice for α.

When there are multiple solutions in the solution pool after removal of duplicates and the best performing solution is optimal for values of α until υ, we apply crossvalidation on a range

of α-values starting from the smallest value for which this solution is optimal.

3.5.4 Final comments

For OCT-H, a CART algorithm that can produce multivariate instead of univariate splits is preferred in step 2a. Unfortunately, we were unable to locate a working implementation. That is why we only generate CART warm starts for $C = 1, ..., 2^{D_{\max}} - 1$; for the remaining C-values, no different solutions will be found. In addition, it is important to mention that OCT-H should not be solved with $\alpha = 0$ in step 2c, because the s_{jt} variables are then no longer pushed to zero when feature j is not used in branch node t. This can cause future warm starts to be incorrect. That is why we use $\alpha = 10^{-4}$.

4 Maximizing boundary margins

In this section, we will consider two useful extensions to the OCT and OCT-H models, and show how the main models can be adjusted to incorporate these extensions.

When constructing the OCT and OCT-H models in their paper, Bertsimas and Dunn only care about minimizing the classification error while making a trade-off with the complexity of the tree. As a result, the actual boundaries might be suboptimal for generalisation purposes. Consider Figure 2, which visualizes the decision boundary when the OCT model is trained on the Iris dataset with only two features and depth one. The boundary is the dashed grey line.

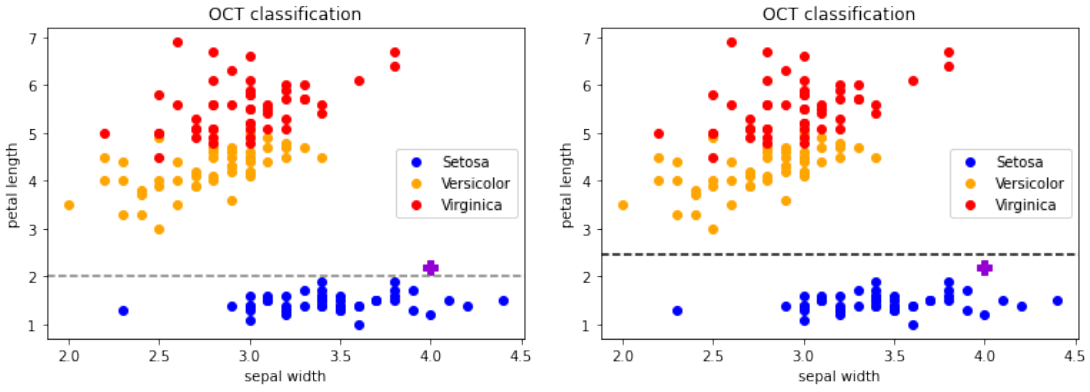

Figure 2: Less desirable (left) and more desirable (right) OCT split

The boundary is relatively close to the Setosa class and relatively far from the Versicolor class datapoints. This means that new observations, such as the purple plus, will not be classified as Setosa, which would be the obvious choice, but as either Versicolor or Virginica, as it ends up in the other node. To improve generalisation of the model, a boundary that lies more in the middle is more desirable. In this way, the new observation will be classified as Setosa. Note that

the model with the adjusted boundary does not change the distribution of observations in the training sample over the leaf nodes, as OCT and OCT-H already perform optimal classification.

4.1 Maximizing OCT boundary margins

For the univariate OCT model, the adjustment of boundaries is relatively straightforward. The only addition to the original model is that in each branch node t, we have to find the maximum feature value of all observations that are sent to the left branch and the minimum feature value of all observations that are sent into the right branch. This concerns the one feature where the split condition is based on. With this difference, we know the range in which the boundary b_t can be shifted without changing the classification on the training sample. The most obvious choice is to place the new boundary in the middle of this interval. It is most efficient to make this adjustment after the model has been solved. We will call this extension of the OCT model Boundary-margin-maximizing OCT, or BOCT for short.

4.2 Maximizing OCT-H boundary margins

With multivariate splits, a simple adjustment of b-values is not possible: if a split is based on multiple features, the range in which one coefficient in the linear combination can be shifted without changing the classification also depends on the values of the other coefficients. Therefore, it seems more logical to fully move to a Support Vector Machine (SVM) model [Boser et al., 1992, Cortes and Vapnik, 1995]. Indeed, in each branch node, we can solve a SVM to find the boundary that maximizes the margin around the hyperplane. In this case, we have two classes; the left branch and the right branch observations, and the two classes are linearly separable. This is because OCT-H applies a linear hyperplane, and the optimal boundary should not change the optimal classification on the training sample provided by the OCT-H model.

Mathematically, we know that the current hyperplane at branch node t can be described by the equation $\mathbf{a_t}^T\mathbf{x} - b_t = 0$, with $\mathbf{x}$ the feature vector. All observations i where $\mathbf{a_t}^T\mathbf{x}_i - b_t < 0$ go to the left branch of node t, all observations i where $\mathbf{a_t}^T\mathbf{x}_i - b_t \geq 0$ go to the right branch. Let N_t denote the number of nodes present in branch node t, for $t \in \mathcal{T}_B$, and let y_{kt} denote the label of observation k, $k = 1, .., N_t$, with $y_{kt} = -1$ if this observation goes into the left branch, and $y_{kt} = 1$ if the observation goes into the right branch of branch node t.

The goal of the SVM in node t is to find a weight vector $\mathbf{w}_t$ and a constant γ_t such that $\mathbf{w}_t^T\mathbf{x} - \gamma_t = 1$ and $\mathbf{w}_t^T\mathbf{x} - \gamma_t = -1$ are the two parallel hyperplanes that 1) both separate the two classes without missclassification errors and 2) have maximum distance between them. It can be derived (see Appendix B) that the distance between the two hyperplanes is $\frac{2}{||\mathbf{w}_t||}$, such that $||\mathbf{w}_t||$ is to be minimized under the no misclassification constraint. The resulting decision

boundary is the middle hyperplane $\mathbf{w}_t^T\mathbf{x} - \gamma_t = 0$, such that $\mathbf{w}_t$ and γ_t contain the adjusted a_{jt}- and b_t-values respectively for one node t and all features $j = 1, ..., p$. Note that we fixed the constant at the right-hand side in the equations for the two class-separating hyperplanes at 1, as is done traditionally in SVM models. It could be any non-zero real number, but freezing it is possible because the equation of a hyperplane can be multiplied with a non-zero constant without changing the orientation and position of the hyperplane. As a consequence, we have to relax the domain of $\mathbf{w}_t$ and γ_t from $[-1, 1]^p$ and $[-1, 1]$ to $\mathbb{R}^p$ and $\mathbb{R}$, respectively.

The no-misclassification constraint entails that in every branch node t that applies a split, $d_t = 1$, for all observations k with $y_{kt} = 1$ it must hold that $\mathbf{w}_t^T\mathbf{x}_k - \gamma_t \geq 1$. Similarly, for all observations k in the node that go into the left branch with $y_{kt} = -1$, $\mathbf{w}_t^T\mathbf{x}_k - \gamma_t \leq -1$ must be satisfied. These can be combined into one, resulting in restriction set (32). However, it is not allowed that the SVM in branch node t uses a feature that is not used in the original split condition. To exclude this possibility, w_{jt} must be set equal to 0 if $s_{jt} = 0$. Since $\mathbf{s}$ is known, we can just impose this.

Taking all the above into account, for each branch node t where $d_t = 1$, we can solve the following optimization problem:

$$\begin{aligned}
\min \quad & ||\mathbf{w}_t|| && && (31)\\
\text{s.t.} \quad & y_{kt}(\mathbf{w}_t^T x_k - \gamma_t) \geq 1, && k = 1, ..., N_t, && (32)\\
& w_{jt} = 0, && j = 1, ..., p, \;\; s_{jt} = 0, && (33)\\
& \mathbf{w}_t \in \mathbb{R}^p, && && (34)\\
& \gamma_t \in \mathbb{R}. && && (35)
\end{aligned}$$

By minimizing v_t and adding the constraint $||\mathbf{w}_t|| \leq v_t$, this becomes a Second Order Cone Program. Naturally, we will name this extension of the OCT-H model BOCT-H. Although problem (31) - (35) can be used to improve the boundaries for all branch nodes t where $d_t = 1$, when $\sum_{j=1}^{p} s_{jt} = 1$, only one feature is used in the split and the method described in Section 4.1 will find the same optimal solution faster.

5 Results

To be able to do a proper comparison with the results of Bertsimas and Dunn [2017], we randomly select 17 of the 54 datasets they use. We transform nominal and ordinal variables with $m > 2$ categories into $m - 1$ dummy variables. In this way, a split can be based on one of the categories in the univariate case, and on a combination of categories in the multivariate case. The resulting number of features per dataset indicate that Bertsimas and Dunn followed the same procedure,

because their numbers coincide.

The OCT and OCT-H models are obtained according to the training algorithm described in Section 3.4. After these solutions are computed, we apply the algorithms in Section 4 to find boundaries that maximize margins between branches. For OCT and OCT-H, these extensions on the models are called BOCT and BOCT-H, respectively. Following Bertsimas and Dunn [2017], we first use 50% of the data to train the model and 25% to tune α. The final model is trained with the training and validation set combined, and tested on the remaining 25% to find the out-of-sample accuracy.

The exception is OCT with depth one and every other model at a depth with only one unique solution in the solution pool. As explained in Section 3.5.3, the tuning algorithm cannot be applied in this case, and we perform a crossvalidation procedure on heuristic values of α on the combined training and validation set to tune the hyperparameter and train the final model. The set of α-values that we consider corresponds with the set of θ-values $\{0.01, 0.1, 0.5, 1, 2.5, 5, 7.5, 10, 15, 20, 25, 30\}$. Remember that θ is the value such that an extra split or split variable is preferred only if the accuracy on the training set increases with more than θ percent of the baseline training accuracy.

We set $N_{\min}$ to 5% of the total number of datapoints. This procedure is repeated three times per model and tree depth. In addition, to make sure that the comparison between OCT, BOCT, OCT-H and BOCT-H is not infected by the random training, test and validation set splits, the same data splits are used for all four models and three depths.

We have to give a special treatment to variables in the training or test set that only take on one value, as the value of ε_j (Equation 22) is not defined. In case this is true for the entire dataset, this variable is removed. If this occurs in the training or test set, the data is again divided randomly until there are at least two values. The only exception is the optical recognition dataset, where this procedure is not feasible. There, we delete a variable immediately from the training and test set if it takes on only one value in one of them.

For OCT and OCT-H, time limits are needed to obtain results in a reasonable time frame. For all models solved during the tuning algorithm for α, also during the crossvalidation procedure, the time limit is 30 seconds. For all final models, trained on the combined training and test sets, we use 300 seconds for all depths. Both the number of iterations and the time limits appear to be on the low end, but bear in mind that for OCT-H, generating all solutions in the tuning algorithm takes $(1 + 2 + 5) * p * 30$ seconds if all time limits are reached. With 30 features, this is more than two hours for one iteration. In all iterations for which the solution pool contains only one unique solution, 3-fold crossvalidation has to be performed on twelve values, which also

takes eighteen minutes. To make this more feasible, only when $p \leq 20$, we use the prescribed algorithm to tune in OCT-H. If $20 < p \leq 40$, we use crossvalidation on the same heuristic values of α on 75% of the data for depth 3. For $p > 40$, this is done for depths 2 and 3. For OCT, the tuning algorithm remains the same for all depths, as the number of iterations required does not depend on the number of features.

CART, the benchmark decision tree model, is obtained as follows. First, we grow the tree using $N_{\min}$ as early stopping hyperparameter. Then, we post-prune the tree using the method described in Section 3.5.1, removing the split with the lowest relative reduction in Gini impurity. Again, we exempt the root node from pruning. The tree is trained on a 75% training set and evaluated on a 25% test set. This is implemented in Python, using the Scikit-learn Tree module. We found this program to run rather fast, so we decided to conduct 100 iterations per maximum depth instead of five to obtain more reliable results.

To give a broader comparison, both on the lower and upper end, we also include the baseline accuracy, equal to the proportion of the most common class in the complete dataset, and a random forest. The random forest consists of 100 trees, trained with depth and $N_{\min}$ as early stopping parameters. Here, $\sqrt{p}$ features are considered when looking for the best split at each branch node. In addition, each tree is trained with 50% of the 75% training set, obtained by random sampling with replacement. This prevents overfitting and improves stability [Breiman, 2001]. Again 100 iterations are performed with a test of 25%. The random forest is implemented in Python, using this time the Ensemble module from the Scikit-learn package.

For the implementation of the four MIO models, we use Julia, a high-performance dynamic programming language well-suited for scientific computing [Bezanson et al., 2017] and JuMP, a package for mathematical optimization embedded in Julia. The solver is Gurobi 9.5.1, one of the most powerful mathematical optimization solvers [Gurobi Optimization Inc., 2022]. CART, random forest and the optimal tree models for datasets with $p \leq 20$ are solved on a machine with an Intel(R) Core(TM) i5-8265U CPU, 4 physical cores, 8 logical processors and 8 GB RAM. The MIO programs for the larger datasets are solved on a machine with an Apple M1 Pro chip, 8 cores and 16 GB RAM.

The following table shows the average accuracy on the test set in percentage for the six models with three different depths, where BASE indicates the baseline accuracy, and RF the random forest ensemble model. If the average accuracy of OCT or OCT-H is boldfaced, it means that it beats CART. If the average accuracy of BOCT or BOCT-H is boldfaced, it means that it beats OCT or OCT-H, respectively.

Table 2: Average test set accuracy of all models for 17 datasets and depth 1-3

	balance_scale	banknote_authentication	breast_cancer_diagnostic	breast_cancer_prognostic	car_evaluation	haberman_survival	hayes_roth	iris	mammographic_mass	optical_recognition	parkinsons	qsar_biodegradation	seismic_bumps	spambase	thoracic_surgery	tic_tac_toe_endgame	wall_following_robot_2
n	625	1372	569	194	1728	306	132	150	830	3823	195	1055	2584	4601	470	957	5456
p	4	4	30	32	15	3	4	4	9	64	21	41	20	57	24	18	2
K	3	2	2	2	4	2	3	3	2	10	2	2	2	2	2	2	4
BASE	46.1	55.5	62.7	76.3	70.0	73.5	38.6	33.3	51.4	10.2	75.4	66.3	93.4	60.6	85.1	65.3	40.4
depth 1																	
CART	59.5	84.8	89.8	74.8	70.4	72.1	38.8	61.9	77.7	18.2	83.1	74.3	93.3	78.6	85.2	70.0	78.8
OCT	58.8	84.4	86.7	**76.9**	69.1	70.1	**39.4**	53.1	76.9	**18.4**	81.0	67.3	93.0	50.6	85.0	67.8	78.4
BOCT	58.8	84.4	**87.2**	76.9	69.1	70.1	39.4	53.1	76.9	18.4	**82.3**	67.3	93.0	50.6	85.0	67.8	78.4
OCT-H	**84.3**	**85.0**	**93.5**	72.8	**86.2**	**75.8**	**61.6**	53.1	77.2	**18.7**	**84.4**	70.3	93.0	61.4	85.0	**95.1**	43.3
BOCT-H	**84.5**	85.0	**93.7**	72.1	86.2	75.3	61.6	53.1	77.2	18.7	83.7	70.3	93.0	61.4	85.0	**95.3**	43.3
RF	78.1	87.6	92.3	76.3	70.1	73.7	49.8	91.1	78.1	71.3	82.4	75.6	93.5	83.8	85.6	64.8	93.6
depth 2																	
CART	60.0	88.2	92.7	74.2	77.6	72.5	44.8	92.9	77.3	19.2	82.5	74.4	93.5	83.0	85.4	70.6	78.6
OCT	**65.2**	85.2	90.4	**76.9**	77.2	71.4	**47.5**	**94.7**	65.7	**28.9**	82.3	59.7	93.0	53.0	85.0	67.4	**85.5**
BOCT	65.2	85.2	90.4	76.9	77.2	**71.9**	47.5	93.9	65.5	28.9	82.3	59.7	93.0	53.0	85.0	67.4	85.2
OCT-H	**75.8**	60.5	**94.2**	**76.2**	**80.9**	**75.3**	**69.7**	**93.0**	76.3	**21.6**	**84.4**	67.9	93.0	61.4	85.0	**96.0**	41.5
BOCT-H	75.6	60.5	94.2	**76.9**	**81.0**	74.9	68.7	**94.7**	**76.4**	**22.0**	83.7	67.9	93.0	61.4	85.0	95.1	41.5
RF	81.4	89.8	93.3	76.1	70.2	73.3	62.4	94.9	79.1	79.8	83.9	79.2	93.4	87.7	85.4	70.0	94.0
depth 3																	
CART	61.4	89.8	92.2	74.1	79.1	73.0	55.5	94.3	78.0	20.5	82.3	76.6	94.4	85.8	84.6	70.0	100.0
OCT	**65.2**	84.2	**93.0**	**76.9**	75.3	72.3	52.5	**94.7**	76.0	18.5	**83.0**	58.3	93.0	56.5	**85.0**	69.4	89.4
BOCT	65.2	84.2	93.0	76.9	75.3	72.3	52.5	**95.6**	75.8	18.5	**83.7**	58.1	93.0	56.5	85.0	69.4	89.2
OCT-H	**81.1**	57.4	81.6	**76.9**	**85.9**	71.0	**67.7**	92.1	70.8	19.1	81.0	67.9	93.0	61.4	**85.0**	**91.8**	41.5
BOCT-H	81.1	57.4	**81.8**	76.9	85.7	71.0	67.7	**95.6**	70.8	**19.2**	81.0	67.9	93.0	61.4	85.0	91.8	41.5
RF	82.7	91.2	93.1	76.7	69.9	73.3	67.5	95.1	79.1	83.9	83.5	79.3	93.4	89.0	84.5	72.3	94.0

Generally, we find that the optimal trees perform better than CART in one third of all depth and dataset combinations, whereas maximizing the boundary margins does not seem to have an effect in most cases. Fortunately, most models give significantly higher average accuracies than the baseline accuracies, but not all. For roughly one fourth of the datasets, even optimal trees with depths higher than one cannot beat the baseline at all depths. This mostly concerns univariate optimal trees. The multivariate optimal trees loses against the baseline less frequently. This indicates its superior performance, which is to be expected.

To perform an in-depth analysis at once, we directly introduce Table 3 and Table 4. They give a clearer overview of the results in Table 2. Table 3 shows the average out-of-sample accuracy for each depth and model. Table 4 presents a pairwise comparison between relevant models, with the number of wins, losses and ties at each depth and the average improvement in test set accuracy. From the results of Bertsimas and Dunn [2017], we selected the same 17 datasets and computed the associated statistics. These are presented in the colored cells.

Table 3: Average test set accuracy across all datasets for each model and depth

Depth	CART	CART	OCT	OCT	BOCT	OCT-H	OCT-H	BOCT-H	RF
1	71.25	72.15	68.05	-	68.16	72.98	78.78	72.91	79.28
2	74.55	76.96	72.29	76.78	72.25	73.69	82.16	73.68	81.99
3	77.15	79.45	73.13	79.37	73.19	72.07	84.35	72.28	82.85

Table 4: Pairwise comparison of models across different depths, where wins, losses and ties indicate the number of datasets for which the leftmost model has on average a higher test set accuracy then the benchmark model, and the average accuracy improvement on the test set

Model	Benchmark model	Depth	Wins	Losses	Ties	Avg. accuracy improvement
OCT	CART	1	3	14	0	-3.20
		2	6	11	0	-2.26
		3	6	11	0	-4.02
OCT	CART	2	5	6	5	-0.19
		3	7	4	6	-0.08
OCT-H	CART	1	9	8	0	+1.73
		2	10	7	0	-0.86
		3	6	11	0	-5.08
OCT-H	CART	1	10	1	6	+6.62
		2	10	4	3	+5.20
		3	9	5	3	+4.90
BOCT	OCT	1	2	0	15	+0.11
		2	1	3	13	-0.05
		3	2	3	12	+0.06
BOCT-H	OCT-H	1	3	3	11	-0.08
		2	5	5	7	-0.01
		3	3	1	13	+0.21
OCTH	RF	1	7	10	0	-6.29
		2	7	10	0	-8.31
		3	5	12	0	-10.78

5.1 OCT and OCTH

The number of boldfaced accuracies in the OCT rows in Table 2 show that one third of the time, OCT outperforms CART. At depth 1, OCT hardly ever provides a higher average test accuracy. Also at depths 2 and 3, CART wins twice as often. Overall, CART outperforms OCT quite significantly at all depths and this is visible both in the number of wins (Table 4) and the average test set accuracies of OCT and CART (Table 3.

For OCT-H, this is different. OCT-H is able to beat CART more often and with larger margins, depending on the dataset. Approximately half the time across all depths, OCT-H beats CART. However, only at depth 1 a higher average accuracy than CART is obtained, although at depth 2 it has more wins than losses. Interestingly, OCT-H at depth 3 often scores lower than at depth 2, and it also beats CART far less often. This shows in the decrease in average out-of-sample accuracy from depth 2 to depth 3. At depth 3, on average, univariate optimal trees even perform better than multivariate optimal trees. The difference in mean accuracy between OCT-H and CART at depth 3 is rather large. It could be that at this depth, multivariate optimal trees already overfit too much on the training data, which decreases their out-of-sample performance.

This version of CART is not able to apply univariate split, and therefore the comparison is between multivariate optimal trees and univariate trees built by recursion. Considering this, the results of OCT-H are worse than expected. The slight advantage OCT-H has over CART for depth 1 and possibly depth 2 can probably be fully explained by the fact that OCT-H can apply multivariate split rules. At the same time, in approximately one third of all datasets and depths, OCT-H outperforms the random forest model, mainly at depths 1 and 2. Often, it comes close to the random forest result. This shows the advantage of multivariate splits: for some datasets, a single OCT-H model is able to defeat an ensemble of 100 CART trees. Still, on average (Table 3, the random forest is significantly better with more than 9% positive average difference across all depths. This shows that a random forest is a more consistent model; the result of ensembling.

5.2 BOCT and BOCT-H

Judging by the low amounts of boldfaced numbers in Table 2 in the rows of the models that maximize boundary margins and the marginal increases in the cases they do outperform their base models, maximizing the boundary margins does not have a large effect for most depths and datasets. Indeed, in the majority of cases, the BOCT average accuracy is exactly equal to the OCT average accuracy, and the same holds, but to a lesser extent, for BOCT-H and OCT-H. Adjusting the boundary causes more differences in accuracy between OCT-H and BOCT-H com-

pared to the differences between OCT and BOCT. Table 4 shows that in the multivariate case, fewer ties are obtained. With multivariate splits, we can shift the boundary in all orientations. With univariate splits, the boundary can only shift up and down. Therefore, we expect that adjusting the boundary in multivariate trees leads to more changes in classification. However, the effect is still not substantial. Only at depth 3 BOCT-H has a slightly positive effect, with a higher average out-of-sample accuracy and more wins than losses against OCT-H.

The roughly equal performance of BOCT and BOCT-H compared to OCT and OCT-H can partly be explained by the sizes of the datasets and their distributions. If there are more points, the probability of a new point being classified differently because of the adjusted boundary is larger. On the other hand, if the sample size becomes too large, the boundary has not much or even no room for adjustment and the effect disappears. For a given dataset, there might be a range of the sample size n for which maximizing the boundary margin does have a significant effect. Across all datasets, there does not appear to be a range in sample size n or ratio between n and p in which the effect is most noticeable, but this may be the case within one dataset.

The BOCT and BOCT-H columns in Table 3 show that at depths 1 and 3, BOCT on average has a higher test accuracy, while for BOCT-H, this is only true for depth 3. However, the differences are rather small. The average accuracy improvements in Table 4 of BOCT compared to OCT and BOCT-H compared to OCT-H confirm this. Only for univariate optimal trees at depth 1 and 3 and multivariate trees at depth 3, there appears to be a small but noticeable positive effect of the boundary margin maximization, although the majority of instances still show exactly the same accuracies. There seems to be no clear reason for this pattern. It seems more likely that whether maximizing the boundary margin gives a different result or not depends on the dataset, more than on the depth. This is because in approximately half the datasets, the results are always equal. In the other half, most of the time the results differ at more than one depth. Secondly, across datasets it happens a lot more often that BOCT and BOCT-H both give different results from OCT and OCT-H than only one of them giving different results.

Maximizing the boundary margin therefore has the potential to beat OCT and OCT-H, but only for certain datasets and not necessarily at all depths. It seems logical to think that the effect is more significant if observations that can be seen as boundary cases between different classes occur more often. Maximizing the boundaries can also have a large impact if the boundaries produced by OCT or OCT-H are too close to one side, such as the one in Figure 2. Unfortunately, it is not immediately clear how to determine that a boundary is skewed; this could only be inspected visually with three or fewer variables. One situation in which adjusting the boundary certainly has no effect, is when a split is based on categorical variables only. For OCT for

example, the boundary coefficients will then change from 1.0 to 0.5, because categorical variables are transformed to dummies.

5.3 Comparison with Bertsimas and Dunn [2017]

For almost all depths and datasets, Bertsimas and Dunn [2017] report substantially higher mean out-of-sample accuracies. We can compare Table 2 with Table 5 in Appendix D, which contains the results from their paper for CART, OCT and OCT-H on the same datasets, to see this. Table 3 supports this statement: for CART, the mean test accuracy is about 2.5% higher. For OCT, we observe a $4.5-7\%$ increase and for OCT-H, $6-13\%$. For the random forest, they only mention an average accuracy of 85.8% across all 53 datasets, which is also substantially higher than the averages we obtained in this sample. For CART and random forests, this difference could be explained by different implementation choices. Furthermore, the number of iterations (five and three) is rather small, leading to more variation in the results.

However, it is remarkable that the accuracies they obtain for CART, OCT and OCT-H are substantially higher most of the time. In addition, they rarely decrease when the depth increases for a dataset; either the accuracy remains equal or it increases. In our results, this phenomenon occurs quite often. The models in Belson [1959] do not seem to suffer from overfitting.

Bertsimas and Dunn compare both OCT and OCT-H with CART. They conclude that OCT has a higher accuracy of around 0.5–2% at all depths. Table 3 shows that for this sample of datasets, OCT has a lower average out-of-sample accuracy than CART at depths 2 and 3 in the results from Bertsimas and Dunn [2017]. This corresponds with our results, however, the difference in average accuracy between OCT and CART they report is much smaller. According to Table 4, OCT and CART roughly have the same number of wins, losses and ties across two depths, with the average accuracy improvement again slightly in favor of CART. According to our results however, CART clearly outperforms OCT on all depths and the difference in average test set accuracy is not small.

Bertsimas and Dunn also compare OCT-H and CART. On average, OCT-H gives approximately 5.5% higher test set accuracy than CART over all depths for the 17 datasets. Table 4 shows that according to their results, OCT-H beats CART two times more often than the other way around for depths 2 and 3, and nearly always at depth 1. This result cannot be replicated in this paper either. In fact, the average accuracy improvement over CART for depths 2 and 3 are even negative in our results. Only at depth 1 the mean out-of-sample accuracy is higher than CART, but still OCT-H only beats CART in 9 out of 17 datasets. Thus, according to our results, for depth 1, OCT-H has a slight advantage over CART, but it loses this advantage at the higher depths.

Because of this, the claim that across all datasets, OCT closes the gap between CART and random forest by about one-sixth, and OCT-H by about half, is contradicted by our results. The averages in Table 3 clearly expose that this does not hold, since on average, OCT and OCT-H perform worse than CART for most depths.

For OCT and OCT-H, a large portion of the difference can probably be explained by the rather tight time limits we impose on the solver. We noticed that with large datasets and datasets with many features, the solver often could not find a better solution than the first-step heuristic or the warm start within 5 minutes, let alone 30 seconds. Moreover, OCT-H returned solutions that were clearly not optimal, such as trees containing splits $0 < 0$, $-x_1 < -1$ or $x_1 < 0$. Indeed, Bertsimas and Dunn [2017] use time limits of 30 minutes to 2 hours for OCT, 5 minutes to 15 minutes for OCT-H and 60 seconds for tuning. Unfortunately, these time limits are necessary due to constraints in time and number of machines. It is a possibility that the time limits do account for these different findings to a large extent.

5.4 Computational performance

However, next to the numeric results, we also have to evaluate the computational performance of these models. There, the difference could hardly be larger. 100 iterations of CART with three different depths takes a few second at most. Even 100 random forests can be trained and tested within 10 seconds for the larger datasets. In contrast, all OCT and OCT-H models are rarely solved to true optimality within the time limits. Gaps between the current best solution and the current lower bound between 60 and 80% are not uncommon for the larger datasets. The running time also increases quickly with depth. The great computational cost is a large disadvantage of these optimal tree models. Afterwards, maximizing the margins between the branches by adjusting the split equations is not a difficult task for a SVM, as there are only two classes that are perfectly linearly separable. This only takes milliseconds. If one is prepared to solve an OCT(-H) problem first, adding this extension is computationally not an issue.

What makes it even worse, is that in the tuning algorithm, many solutions are generated more than once. After keeping track of the number of MIO solutions generated in one iteration of the tuning algorithm that are approximately equal to a solution already generated before, we can conclude that for datasets with relatively many features (≥ 10), this occurs rather often. Although a warm start is used, the solver is not able to close the gap even with 60 seconds. This behaviour is very time consuming, which is not desirable with models that are computationally already very expensive without tuning its parameters.

6 Conclusion

In this paper, our aim is to answer two research questions regarding the novel Mixed-Integer Optimization method Bertsimas and Dunn [2017] propose to obtain optimal classification trees. Firstly, can their results be validated in a replication study? Secondly, do the performances of optimal univariate and multivariate optimal trees increase when the margins of the decision boundaries are maximized?

To answer the first question, we have investigated whether univariate and multivariate optimal classification trees provide a consistent and substantial increase in mean out-of-sample accuracy over recursive CART. Our results show that for univariate optimal trees with depth 1-3 and multivariate trees with depths 2 and 3, this is not the case within the imposed time limits. On average, CART provides $2-4\%$ higher accuracy than univariate optimal trees. Multivariate optimal trees with depths 2 and 3 have on average 1% and 5% lower average test set accuracy compared to CART. The fact that multivariate optimal trees beat CART at depth 1 merely confirms the advantage of multivariate instead of univariate splits. Therefore, the paper appears to be overambitious in its claims and we have to answer the first question in the negative.

To answer the second question, we have adjusted the decision boundaries from optimal trees in order to maximize the margins, but find that this has no substantial or consistent positive effect on the test set accuracy. In the majority of cases, their performances are equal. Only for certain datasets, and not necessarily at all depths, we find a slightly positive effect in the magnitude of 0.1%. For multivariate trees, the adjusted boundaries result in different classifications more often than for univariate trees.

For future research, we highly suggest to develop models and techniques such that faster computation times for optimal trees are possible. Currently, this remains a major obstacle that prevents optimal trees from being used in practice.

References

W. A. Belson. Matching and prediction on the principle of biological classification. *Journal of the Royal Statistical Society: Series C (Applied Statistics)*, 8(2):65–75, 1959.

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

J. Bezanson, A. Edelman, S. Karpinski, and V. B. Shah. Julia: A fresh approach to numerical computing. *SIAM review*, 59(1):65–98, 2017.

B. E. Boser, I. M. Guyon, and V. N. Vapnik. A training algorithm for optimal margin classifiers. In *Proceedings of the fifth annual workshop on Computational learning theory*, pages 144–152, 1992.

L. Breiman. Random forests. *Machine learning*, 45(1):5–32, 2001.

L. Breiman, J. Friedman, R. Olshen, and C. Stone. Cart. *Classification and Regression Trees*, 1984.

C. Cortes and V. Vapnik. Support-vector networks. *Machine learning*, 20(3):273–297, 1995.

P. Doyle. The use of automatic interaction detector and similar search procedures. *Journal of the Operational Research Society*, 24(3):465–467, 1973.

H. J. Einhorn. Alchemy in the behavioral sciences. *Public Opinion Quarterly*, 36(3):367–378, 1972.

Gurobi Optimization Inc. *Gurobi optimizer reference manual*, 2022. `http://www.gurobi.com/`.

G. V. Kass. An exploratory technique for investigating large quantities of categorical data. *Journal of the Royal Statistical Society: Series C (Applied Statistics)*, 29(2):119–127, 1980.

I. Kononenko. On biases in estimating multi-valued attributes. In *Ijcai*, volume 95, pages 1034–1040. Citeseer, 1995.

H. Laurent and R. L. Rivest. Constructing optimal binary decision trees is np-complete. *Information processing letters*, 5(1):15–17, 1976.

W.-Y. Loh and Y.-S. Shih. Split selection methods for classification trees. *Statistica sinica*, pages 815–840, 1997.

R. Messenger and L. Mandell. A modal search technique for predictive nominal scale multivariate analysis. *Journal of the American statistical association*, 67(340):768–772, 1972.

J. N. Morgan and J. A. Sonquist. Problems in the analysis of survey data, and a proposal. *Journal of the American statistical association*, 58(302):415–434, 1963.

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

J. R. Quinlan. *C4. 5: programs for machine learning*. Elsevier, 2014.

A. P. White and W. Z. Liu. Bias in information-based measures in decision tree induction. *Machine Learning*, 15(3):321–329, 1994.

A Model formulation OCT-H

$$\min \quad \frac{1}{\hat{L}} \sum_{t \in \mathcal{T}_L} L_t + \alpha \sum_{t \in \mathcal{T}_B} \sum_{j=1}^{p} s_{jt} \tag{36}$$

$$\text{s.t.} \quad L_t \geq N_t - N_{kt} - n(1 - c_{kt}), \qquad k = 1, ..., K, \ \forall t \in \mathcal{T}_L, \tag{37}$$

$$L_t \leq N_t - N_{kt} + n c_{kt}, \qquad k = 1, ..., K, \ \forall t \in \mathcal{T}_L, \tag{38}$$

$$L_t \geq 0, \qquad \forall t \in \mathcal{T}_L, \tag{39}$$

$$N_{kt} = \frac{1}{2} \sum_{i=1}^{n} (1 + Y_{ik}) z_{it}, \qquad k = 1, ..., K, \ \forall t \in \mathcal{T}_L, \tag{40}$$

$$N_t = \sum_{i=1}^{n} z_{it}, \qquad \forall t \in \mathcal{T}_L, \tag{41}$$

$$\sum_{k=1}^{K} c_{kt} = l_t, \qquad \forall t \in \mathcal{T}_L, \tag{42}$$

$$\mathbf{a}_m^T \mathbf{x}_i + \mu \leq b_m + (2 + \mu)(1 - z_{it}), \qquad i = 1, ..., n, \ \forall t \in \mathcal{T}_L, \ \forall m \in \mathcal{A}_L(t), \tag{43}$$

$$\mathbf{a}_m^T \mathbf{x}_i \geq b_m - 2(1 - z_{it}), \qquad i = 1, ..., n, \ \forall t \in \mathcal{T}_L, \ \forall m \in \mathcal{A}_R(t), \tag{44}$$

$$\sum_{t \in \mathcal{T}_L} z_{it} = 1, \qquad i = 1, ..., n, \tag{45}$$

$$z_{it} \leq l_t, \qquad i = 1, ..., n \ \forall t \in \mathcal{T}_L, \tag{46}$$

$$\sum_{i=1}^{n} z_{it} \geq N_{\min} l_t, \qquad \forall t \in \mathcal{T}_L, \tag{47}$$

$$\sum_{j=1}^{p} \hat{a}_{jt} \leq d_t, \qquad \forall t \in \mathcal{T}_B, \tag{48}$$

$$\hat{a}_{jt} \leq a_{jt}, \qquad j = 1, ..., p, \ \forall t \in \mathcal{T}_B, \tag{49}$$

$$-\hat{a}_{jt} \geq -a_{jt}, \qquad j = 1, ..., p, \ \forall t \in \mathcal{T}_B, \tag{50}$$

$$-s_{jt} \leq a_{jt} \leq s_{jt}, \qquad j = 1, ..., p, \ \forall t \in \mathcal{T}_B, \tag{51}$$

$$s_{jt} \leq d_t, \qquad j = 1, ..., p, \ \forall t \in \mathcal{T}_B, \tag{52}$$

$$\sum_{j=1}^{p} s_{jt} \geq d_t, \qquad \forall t \in \mathcal{T}_B, \tag{53}$$

$$-d_t \leq b_t \leq d_t, \qquad \forall t \in \mathcal{T}_B, \tag{54}$$

$$d_t \leq d_{p(t)}, \qquad \forall t \in \mathcal{T}_B \setminus \{1\}, \tag{55}$$

$$z_{it}, l_t \in \{0, 1\}, \qquad i = 1, ..., n, \ \forall t \in \mathcal{T}_L, \tag{56}$$

$$s_{jt}, d_t \in \{0, 1\}, \qquad j = 1, ...p, \ \forall t \in \mathcal{T}_B, \tag{57}$$

$$c_{kt} \in \{0, 1\}, \qquad k = 1, ..., K, \ t \in \mathcal{T}_L. \tag{58}$$

B Proof for the distance between the two hyperplanes in SVM

In an optimal solution to the SVM problem, there exists at least one point $\mathbf{x}^+$ with label 1 that lies on the hyperplane $\mathbf{w}^T\mathbf{x} - \gamma = 1$, such that

$$\mathbf{w}^T\mathbf{x}^+ - \gamma = 1.$$

If this would not be the case, the distance between the two hyperplanes $\mathbf{w}^T\mathbf{x} - \gamma = 1$ and $\mathbf{w}^T\mathbf{x} - \gamma = -1$ would not be maximized, as $\mathbf{w}^T\mathbf{x} - \gamma = 1$ can be shifted further away until it goes through a point with label 1, resulting in a larger distance. This is a contradiction with the requirements of a solution to the SVM problem. A symmetric argument holds for the existence of at least one point $\mathbf{x}^-$ with label -1 that lies on the hyperplane $\mathbf{w}^T\mathbf{x} - \gamma = -1$, such that

$$\mathbf{w}^T\mathbf{x}^- - \gamma = -1.$$

Graphically, the distance between the two hyperplanes r is equal to the length of the projection of $\mathbf{x}^+ - \mathbf{x}^-$ (or equivalently, $\mathbf{x}^- - \mathbf{x}^+$) on the normal vector $\mathbf{w}$. Because the dot product between two vectors is equal to the length of one vector multiplied by the length of the orthogonal projection of the other vector on that vector, we have $\mathbf{a} \cdot \mathbf{b} = ||\mathbf{a}|| * ||\text{proj}_{\mathbf{b}}(\mathbf{a})||$. We can rewrite this to $||\text{proj}_{\mathbf{b}}(\mathbf{a})|| = \frac{\mathbf{a}\cdot\mathbf{b}}{||\mathbf{a}||}$. This gives

$$\begin{aligned} r &= \frac{\mathbf{w} \cdot (\mathbf{x}^+ - \mathbf{x}^-)}{||\mathbf{w}||} \\ &= \frac{\mathbf{w} \cdot \mathbf{x}^+ - \mathbf{w} \cdot \mathbf{x}^-}{||\mathbf{w}||} \\ &= \frac{\gamma + 1 - (\gamma - 1)}{||\mathbf{w}||} \\ &= \frac{2}{||\mathbf{w}||}. \end{aligned}$$

C Proof for the θ parameter in the tuning algorithm

Remember that in OCT, the objective function is $\frac{1}{\hat{L}} \sum_{t \in \mathcal{T}_L} L_t + \alpha \sum_{t \in \mathcal{T}_B} d_t$, where $\sum_{t \in \mathcal{T}_L} L_t$ is the total number of misclassified points and $\sum_{t \in \mathcal{T}_B} d_t$ is the number of applied splits in the branch nodes. We want to find α^* such that an extra split is preferred if the accuracy on the training data increases with more than θ percent of the baseline accuracy $\hat{L}$. This increase amounts to a fixed number of extra points being classified correctly, which is equal to $\theta\hat{L}n/100$, with n the total number of observations in the training data.

It follows that an extra split is allowed if the following equation holds:

$$\frac{1}{\hat{L}} \sum_{t \in \mathcal{T}_L} L_t + \alpha \sum_{t \in \mathcal{T}_B} d_t \geq \frac{1}{\hat{L}} \left(\sum_{t \in \mathcal{T}_L} L_t - \frac{\theta \hat{L} n}{100} \right) + \alpha \left(\sum_{t \in \mathcal{T}_B} d_t + 1 \right). \tag{59}$$

The equality is where we can find α^*. We can quickly see that most terms cancel out against each other, such that it is not difficult to derive that $\alpha^* = \theta n/100$.

D Results from Bertsimas and Dunn [2017]

Table 5: Average test set accuracy for 17 datasets and depth 1-3 from OCT, OCT-H and CART according to the results from Bertsimas and Dunn [2017]

	balance_scale	banknote_authentication	breast_cancer_diagnostic	breast_cancer_prognostic	car_evaluation	haberman_survival	hayes_roth	iris	mammographic_mass	optical_recognition	parkinsons	qsar_biodegradation	seismic_bumps	spambase	thoracic_surgery	tic_tac_toe_endgame	wall_following_robot_2
n	625	1372	569	194	1728	306	132	150	830	3823	195	1055	2584	4601	470	957	5456
p	4	4	30	32	15	3	4	4	10	64	21	41	20	57	24	18	2
K	3	2	2	2	4	2	3	3	2	10	2	2	2	2	2	2	4
BASE	46.1	55.5	62.7	76.3	70.0	73.5	38.6	33.3	51.4	10.2	75.4	66.3	93.4	60.6	85.1	65.3	40.4
depth 1																	
CART	60.9	83.6	88.5	75.5	69.9	72.7	44.8	64.9	81.2	19.4	84.9	73.7	93.3	78.4	**85.5**	70.6	78.8
OCT-H	**87.6**	**89.8**	**93.1**	75.5	**87.5**	**73.0**	**60.6**	64.9	81.2	19.4	**86.5**	**83.3**	93.3	**83.6**	83.9	**97.2**	78.8
depth 2																	
CART	64.5	89.0	90.5	**75.5**	73.7	**73.2**	**52.7**	92.4	81.2	**29.7**	**84.1**	**76.4**	93.3	84.2	**85.5**	68.5	94.0
OCT	**67.1**	**90.1**	**91.9**	75.1	73.7	73.2	45.5	92.4	81.2	29.4	83.7	76.1	93.3	**84.3**	84.6	**69.6**	94.0
OCT-H	**87.6**	**91.5**	**93.1**	75.5	**87.5**	73.0	**61.2**	**95.1**	81.2	29.3	**84.9**	**83.0**	93.3	**85.7**	83.9	**97.0**	94.0
depth 3																	
CART	**70.4**	89.0	90.5	75.5	77.4	**73.5**	**56.4**	92.4	**81.7**	**41.4**	**86.1**	78.5	93.3	**86.0**	**85.5**	73.1	100.0
OCT	68.9	**89.6**	**91.5**	75.5	77.4	73.2	53.3	**93.5**	81.7	**41.6**	**86.5**	**78.6**	93.3	86.0	84.6	**74.1**	100.0
OCT-H	**87.6**	**98.7**	**94.0**	75.5	**87.5**	73.0	**71.5**	**95.1**	81.4	41.0	84.9	**83.0**	93.3	**86.6**	83.9	**97.0**	100.0

The winner between CART and OCT and between CART and OCT-H is highlighted in bold. A bold number for CART therefore signifies that either CART beats OCT or OCT-H individually, or both.

E Description of programming files

The file MIO_Models.jl contains all Julia code needed to replicate the mean out-of-sample accuracies for the univariate and multivariate optimal tree models. The file Cart.py contains the Python code to obtain the CART model out-of-sample accuracies. Finally, RandomForest.py contains the code that generates the random forest average test set accuracies.