Optimal Decision Tree - MIP Formulations, Solution Methods, and Stability

Bo Lin, Bo Tang

March 2021

Abstract

This project aims to better understand the computational and prediction performance of MIP-based classification tree learning approaches proposed by recent research papers. In particular, the OCT, binOCT, and flowOCT formulations are implemented and evaluated on publicly available classification datasets. Moreover, we propose a stable classification tree formulation incorporating the training and validation split decision into the model fitting processes. Computational results suggest flowOCT has the most consistent performance across datasets and tree complexity. OCT is competitive on small datasets and shallow trees, while binOCT yields the best performance on large datasets and deep trees. The proposed stable classification tree achieves stronger out-of-sample performance than flowOCT under random training and validation set splits.

Keywords— Classification tree, mixed integer programming, model stability

1 Introduction

Decision tree is one of the most popular machine learning models for its great human interpretability and effectiveness in various prediction tasks. Given a set of training data, a decision tree recursively applies splitting rules to partition the training set into groups and assigns a label to each resulting partition. The tree can then be used to predict the labels for future examples with the rules it learns from the training data.

Despite its simplicity, learning an optimal decision tree is $\mathcal{NP}$-hard (Laurent and Rivest 1976). Classic training methods, such as CART (Breiman et al. 1984) and C4.5 (Quinlan 2014), that grow decision trees top-down have been widely adopted as they are able to find "good enough" trees efficiently. Nevertheless, these "top-down" approaches are greedy and myopic in nature, thus often lead to sub-optimal trees (Hu et al. 2019). Moreover, these greedy approaches are not always robust, they tend to overfit the training data (Günlük et al. 2018). Post-pruning heuristics are typically required to achieve trees that generalize well (Bertsimas and Dunn 2017).

Research efforts have been made to harness the powerful mixed-integer programming (MIP) techniques to learn optimal decision trees. Given the depth of a decision tree, Bertsimas and Dunn (2017) formulate the learning processes as an MIP problem where decisions about the split rule at each node, label assignment for each leaf node, and the routing of each data point from the root node to a leaf node are made to minimize the misclassification rate. The model can deal with both continuous and categorical features. Günlük et al. (2018) focus specifically on decision trees that intake categorical features. They exploit the resulting combinatorial structure to carefully design a formulation that can be solved relative efficiently with commercial solvers.

Both Bertsimas and Dunn (2017) and Günlük et al. (2018) use binary decision variables that are associated with each training example, the problem size thus grow dramatically with the size of training set and can easily become intractable. To address this issue, Verwer and Zhang (2019) propose a new formulation whose size is largely independent of the size of the dataset. Computational results are presented to demonstrate its advantages in terms of solution time and prediction accuracy. To further improve the scalability, Firat et al. (2020) design a path-based formulation. They regard the problem as selecting multiple decision paths from the root node to leaf nodes. Branching rules are determined for nodes along each path such that paths with common intermediate nodes agree with each other on their shared branching rules. Thanks to this path-based formulation, they are able to apply column generation to solve the problem. From a row-generation perspective, Aghaei et al. (2021) come up with a max-flow formulation without "big-M" to train decision trees that intake binary features. Accordingly, they propose a benders decomposition approach to solve the problem efficiently.

Learning an optimal decision tree solely based on training data may result in overfitting issues. To tackle the challenge, Bertsimas and Dunn (2017) incorporate a generalization term in the objective function to penalize trees with too many branches. Same method is applied by Aghaei et al. (2021) to control tree complexity. In addition, Bertsimas and Dunn (2017) propose a constraint to guarantee that splits only

happens at nodes where at least N_{min} data points in the training dataset could reach. Günlük et al. (2018), Verwer and Zhang (2019), and Firat et al. (2020) do not specifically consider overfitting prevention mechanism in their proposed formulations.

Although all the aforementioned works showcase nice out-of-sample performance, the generalization capability of MIP-based decision trees has not been well understood (Zantedeschi et al. 2020). The reasons are two-fold. First, the exisiting MIP approaches can only deal with small trees (max depth = 5) that do not have serious overfitting issues even with the greedy training methods. Second, few systematic computational experiments have been done to evaluate the up-to-date MIP-based methods. To the best our knowledge, the most recent numerical study is presented by Aghaei et al. (2021) where features are one-hot encoded before being fed to the MIP models. This method can potentially hinders the performance of models that can deal with continuous variables, such as the ones by Bertsimas and Dunn (2017) and Verwer and Zhang (2019).

Motivated by the aforementioned observations, this project aims to better understand the performance of existing MIP decision tree formulations. In particular, we implement the MIP formulations proposed by Bertsimas and Dunn (2017), Verwer and Zhang (2019), and Aghaei et al. (2021) and compare their out-of-sample performance on instances from the UCI data repository (Dua and Graff 2017). Moreover, we develop a stable decision tree formulation incorporating the ideas of Bertsimas and Paskov (2020) that achieves better out-of-sample performance according to our computational experiments.

The rest of the report is organized as follows. We formally define the optimal decision tree problem and reviews the three formulations in Section 2. We then illustrate the formulation and solution method of stable classification tree in Section 3. Computational experiments are presented in Section 4. Finally, we conclude the report with Section 5.

2 Problem Formulations

In this section, we first formally introduce the classification tree learning problem in subsection 2.1, and then review three formulations proposed by Bertsimas and Dunn (2017) (OCT), Verwer and Zhang (2019) (binOCT), and Aghaei et al. (2021)(FlowOCT) in subsections 2.2, 2.3, and 2.4, respectively.

To highlight the connections and differences, OCT employs a large number of binary decision variables associated with each node to describe split rules at branch nodes as well as label and data point assignments to leaf nodes. BinOCT exploits a binary encoding methods to model the threshold selection at branch nodes, which together with some "big-M" constraints reduce the number of binary decision variables needed. Assuming all the features are binary, flowOCT is free of "big-M" constraint. Furthermore, they regard classifying a data point as sending a unit of flow from the root to a dummy sink connected with all nodes in the decision tree. They develop a benders decomposition algorithm to accelerate the solution processes of this "max-flow" problem.

We next present the formulations in details. We note that some notations in the original papers are changed for the ease of illustration and comparison. Notations are summarized in Table 1.

2.1 Problem Description

We are given a training dataset $D = \{x^i, y^i\}_{i \in I}$ consisting of data points indexed by set I. Each data point (x^i, y^i) has $|F|$ features $x^i \in \mathbb{Z}^{|F|}$ and a label $y^i \in \{1, 2, \ldots, K\}$. Note that the OCT and binOCT formulations can deal with continuous features, yet the flowOCT can only deal with binary features. For the sake of comparison, we assume features are discrete so that we can convert them to binary features using one-hot encoding.

The decision tree is defined on a binary tree of depth d. Let B and L denote the sets of branch nodes and leaf nodes in this tree. We seek to recursively partition the feature space $\mathbb{Z}^F$ by applying a split rule parameterized by $a_n \in \mathbb{R}^F$ and $b_n \in \mathbb{R}$ at each branch node $n \in B$. For a data point (x^i, y^i) arrives at the node, it would be directed to the left branch if $a_n^T x^i < b_n$ and to the right branch otherwise. Each leaf node is assigned a label in $\{1, 2 \ldots, K\}$ which will be the prediction for all data points fall into this node.

Fitting a decision tree is to determine the split rules and the label assignments such that the number of misclassified data points is minimized. Note that applying a split rule at every branch node can potentially lead to overfitting issues, especially when d is large. The OCT and flowOCT formulations thus allow decisions about whether to apply split rules at each branch node or not, but binOCT enforce all branch nodes to split data points.

2.2 Optimal Classification Tree

The decision variables for the OCT formulation are listed as follows. In particular, we need decision variables $a_{t,f}$ and b_t at each branch node to describe the corresponding split rule, and c_{tk} at each leaf node to illustrate the label assignments. Moreover, in order to prevent overfitting, the OCT allows not applying any split rule at a branch node, the authors use d_t to indicate this decision. Decision variable z_{it} represents how each data point in the training set is allocated to the leaf nodes. Furthermore, they use e_t, l_t, m_{tk} and n_{tk} as intermediate variables to describe the prediction performance at each leaf node on the training data set.

Table 1: Summary of notation

	Set
I	Set of training data
F	Set of features
K	Set of labels
B	Set of branch nodes in the decision tree
L	Set of leaf nodes in the decision tree
$A(t)$	Set of ancestors of node t in the decision tree
$A_R(t)$	Set of right-branch ancestors of node t
$A_L(t)$	Set of left-branch ancestors of node t
$L_R(t)$	Set of right-branch leaf nodes of node t
$L_L(t)$	Set of left-branch leaf nodes of node t
	Index
i	Instance in training data, $i \in I$
f	Feature in training data, $f \in F$
k	Labels in training data, $k \in K$
t	Node of tree, $t \in B \cup T$
$p(t)$	Parent of node t
$l(t)$	Left child of node t
$r(t)$	Right child of node t
	Global constants
$\hat{e}$	Baseline prediction accuracy obtained by predicting the most popular class
α	Regularization parameter in the objective function
$S_{\min}$	minimum number of instances at each leaf node
	Constants for OCT
ϵ_f	The smallest non-zero value distance of feature f
$\epsilon_{\max}$	$\max_{f \in F}\{\epsilon_j\}$
	Constants and sets for binOCT
$\text{bin}(f)$	binary encoding ranges of feature f
$\text{lr}(b)$	lower range of feature values bin b, $b \in \text{bin}(f)$
$\text{ur}(b)$	upper range of feature values bin b, $b \in \text{bin}(f)$
$\text{tl}(b)$	set of encoding threshold variables for values bin b, $b \in \text{bin}(f)$
$f_{\min}$	minimum decision threshold value of feature f, $f \in F$
$f_{\max}$	maximum decision threshold value of feature f, $f \in F$
M_0^f	$\sum_{i \in I: i \in \text{lr}(b)} 1$, $f \in F$
M_1^f	$\sum_{i \in I: i \in \text{ur}(b)} 1$, $f \in F$
M_2^f	$\sum_{i \in I: i \in \text{ur}(b)} 1$, $f \in F$
M_3^f	$\sum_{i \in I: x_f^i < f_{\min}} 1 + \sum_{i \in I: x_f^i > f_{\max}} 1$, $f \in F$
M_4^k	$\sum_{i \in I: y^i = k} 1$, $k \in K$

- a_{tf}, if feature f is selected to split at branch node t (= 1) or not (= 0)
- b_t, the threshold at branching node t
- c_{tk}, if label k is assigned to leaf node t (= 1) or not (= 0)
- d_t, if a split rule is applied to branch node t (= 1) or not (= 0)
- z_{it}, if data point i is assign to leaf node t (= 1) or not (= 0)
- e_t, the number of mis-classified instances at leaf node t
- l_t, if leaf nodes t has at least S_{min} data points (= 1) or not (= 0)
- m_{tk}, the number of data points with label k at leaf node t
- n_{tk}, the total number of data points at leaf node t

We then formulate the problem as a linear MIP as follows. The objective function (1) minimizes the weighted sum of the number of mis-classified data points and the number of branch nodes where split rules apply. Constraints (2) and (3) make sure that split rules are generated only at nodes we decide to

perform split. Constraints (4) mean that if we stop splitting at branch node t, then no split rule applies to the its children in the decision tree. Constraints (5) guarantee that no label would assigned to empty leaf nodes. Constraints (6) enforce every data point in the training set to fall into one of the leaf nodes. Constraints (7) and (8) ensure each leaf node, if not empty, has at least S_{min} data points, which prevents the decision tree from learning label assignment from a very small subset of the training data. Constraints (9) and (10) describe how each data point i is routed from the root to one of the leaf nodes in the decision tree. Cosntraints (11)-(14) are used to calculate the number of mis-classified data points in each leaf node. Constraints (15) and (16) describe the ranges and types of the decision variables.

$$\min \quad \frac{1}{\hat{e}} \sum_{t \in L} e_t + \alpha \sum_{t \in B} d_t \tag{1}$$

$$\text{s.t.} \quad \sum_{f \in F} a_{tf} = d_t \qquad \forall t \in B \tag{2}$$

$$0 \leq b_t \leq d_t \qquad \forall t \in B \tag{3}$$

$$d_t \leq d_{p(t)} \qquad \forall t \in B \setminus \{1\} \tag{4}$$

$$\sum_{k \in K} c_{tk} = l_t \qquad \forall t \in L \tag{5}$$

$$\sum_{t \in L} z_{it} = 1 \qquad \forall i \in I \tag{6}$$

$$z_{it} \leq l_t \qquad \forall i \in I, \forall t \in L \tag{7}$$

$$\sum_{i \in I} z_{it} \geq S_{\min} l_t \qquad \forall t \in L \tag{8}$$

$$\sum_{f \in F} x_f^i a_{sf} \geq b_t - (1 - z_{it}) \qquad \forall i \in I,\ \forall t \in B,\ \forall s \in A_R(t) \tag{9}$$

$$\sum_{f \in F} (x_f^i + \epsilon_f) a_{sf} \leq b_t + (1 + \epsilon_{\max})(d_s - z_{it}) \qquad \forall i \in I,\ \forall t \in B,\ \forall s \in A_L(t) \tag{10}$$

$$m_{tk} = \sum_{i \in I : y^i = k} z_{it} \qquad \forall t \in L,\ \forall k \in K \tag{11}$$

$$n_t = \sum_{i \in I} z_{it} \qquad \forall t \in L \tag{12}$$

$$e_t \geq n_t - m_{tk} - |I|(1 - c_{tk}) \qquad \forall t \in L,\ \forall k \in K \tag{13}$$

$$e_t \leq n_t - m_{tk} + |I| c_{tk} \qquad \forall t \in L,\ \forall k \in K \tag{14}$$

$$a_{t,f}, d_t \in \{0, 1\} \qquad \forall t \in B,\ \forall f \in F \tag{15}$$

$$c_{tk}, z_{i,t}, l_t \in \{0, 1\} \qquad \forall i \in I,\ \forall t \in L,\ \forall k \in K \tag{16}$$

The OCT formulates the decision tree learning problem in a straightforward manner. However, it requires a large number of binary decision variable associated with each node in the decision tree and each data point in the training set. We next present the binOCT formulation proposed by Verwer and Zhang (2019) which drastically reduces the number of binary decision variables at the expense of utilizing several "big-M" constraints.

2.3 Optimal Classification Tree with Binary Encoding

To reduce the number of binary decision variable needed, binOct formulation models the selection of decision threshold at each node with a binary search procedure. They represent each possible decision thresholds as a binary encoding. For ease of explanation, we consider using a decision tree of depth 2 to classify 9 data points with one feature as presented in Table 2.

Table 2: Example: a dataset of 9 data points with one feature

Feature value	0	0	1	2	2	3	4	5	6
Label	-	-	-	+	-	+	-	-	+

Although the feature has 7 distinct values, there is no reason to split data with the same label or with the same feature value. There are only four possible thresholds - 1.5, 2.5, 3.5 and 5.5. Instead of using 4 binary variables to represent threshold selection, we need use $\log_2 4 = 2$ binary variables r_{t1} and r_{t2} to encode this decision. In particular, as illustrated in Table 3, $r_{t1} = r_{t2} = 1$ corresponds to setting the threshold as 1.5. Consequently, data points with feature values of 0 or 1 are directed to the right branch while the rest goes to the left branch. Similarly, other thresholds are encoded as other value combinations of r_{t1} and r_{t2}.

Table 3: Example: threshold enconding

r_{t1}	r_{t2}	0,1	2	3	4,5	6	threshold
1	1	1	0	0	0	0	1.5
1	0	1	1	0	0	0	2.5
0	1	1	1	1	0	0	3.5
0	0	1	1	1	1	0	5.5

The decision variables for the binOCT formulation are listed as follows. Similar to OCT, a_{tf} and b_t describe the corresponding split rule at branch nodes, c_{tk} presents label assignments at leaf nodes, z_{it} identifies that instance i reaches leaf node t, and e_{tk} defines the number of misclassified instances. In addition, there are binary encoding variables r_{tj} to represent feature threshold at branch node t. It is worth noting that, by using a set of "big-M" type constraints, the decision variables z_{it} can be regarded as continuous variables in the formulation.

- a_{tf}, if feature f is selected to split at branch node t (= 1) or not (= 0)
- c_{tk}, if label k is assigned to leaf node t (= 1) or not (= 0)
- e_{tk}, the number of misclassified instances with predicted label k at the leaf node t
- l_t, if leaf node t containts at least S_{min} data points (= 1) or not (= 0)
- r_{tj}, encoding threshold variables at branch node t (binary)
- z_{it}, if data point i finally falls into leaf node t (= 1) or not (= 0)

$$\min \quad \frac{1}{\hat{e}} \sum_{t \in L, k \in K} e_t \tag{17}$$

$$\text{s.t.} \quad \sum_{f \in F} a_{tf} = 1 \qquad \forall t \in B \tag{18}$$

$$\sum_{k \in K} c_{tk} = 1 \qquad \forall t \in L \tag{19}$$

$$\sum_{t \in L} z_{it} = 1 \qquad \forall i \in I \tag{20}$$

$$z_{it} \leq l_t \qquad \forall i \in I, \forall t \in L \tag{21}$$

$$\sum_{i \in I} z_{it} \geq S_{\min} l_t \qquad \forall t \in L \tag{22}$$

$$M_1^f a_{tf} + \sum_{\substack{i \in \text{ur}(b) \\ s \in L_L(t)}} z_{is} + \sum_{j \in \text{tl}(b)} M_1^f (r_{tj} - 1) \leq M_1^f \qquad \forall t \in B,\ \forall f \in F,\ \forall b \in \text{bin}(f) \tag{23}$$

$$M_2^f a_{tf} + \sum_{\substack{i \in \text{lr}(b) \\ s \in L_R(t)}} z_{is} - \sum_{j \in \text{tl}(b)} M_2^f r_{tj} \leq M_2^f \qquad \forall t \in B,\ \forall f \in F,\ \forall b \in \text{bin}(f) \tag{24}$$

$$M_3^f a_{tf} + \sum_{\substack{i \in I: x_f^i > f_{\max}, s \in L_R(t), \\ i \in I: x_f^i < f_{\min}, s \in R_R(t)}} z_{is} \leq M_3^f \qquad \forall t \in B,\ \forall f \in F \tag{25}$$

$$\sum_{i \in I: y^i = k} z_{it} - M_4^k c_{tk} \leq e_{tk} \qquad \forall t \in L,\ \forall k \in K \tag{26}$$

$$a_{t,f},\ d_t,\ r_{tj} \in \{0, 1\} \qquad \forall t \in B,\ \forall f \in F,\ \forall j \in \text{tl}(b) \tag{27}$$

$$c_{tk},\ l_t \in \{0, 1\} \qquad \forall t \in L,\ \forall k \in K \tag{28}$$

The MIP formulation of binOCT is as follows. The objective function (17) minimize the number of misclassified data points. Constraints (18) decide which features to split at each branch node. Constraints (19) enforce a label assignment to the leaf nodes. Constraints (20) make sure all data points reach one of the leaf nodes. Constraints (21) and (22) limit the least number of instances at the leaf nodes. Constraints (23), (24) and (25) guarantee each data points follows the split rule. Constraints (26) calculate the number of misclassified data points. Constraints (27) and (28) are domains of decision variables.

2.4 Max-Flow Optimal Classification Tree

Both the OCT and binOCT formulation utilizes constraints with "big-Ms" which can potentially result in loose LP relaxations, thus poor performance of the branch-and-bound method. Assuming all the features are binary, Aghaei et al. (2021) propose the following flowOCT formulation which is "big-M" free.

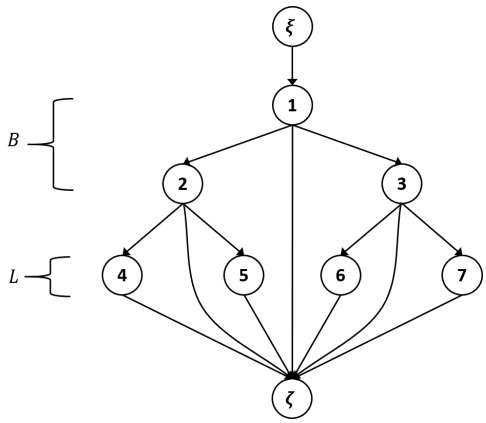

Figure 1: Decision tree with a dummy sink ζ

As presented below, the decision variables for flowOCT are very similar to those for OCT. In particular, we have a_{tf} and b_t to illustrate the split rule at each branch node. Decision variable c_{tk} indicate the label assignment for each node. We note that label assignment to a branch node is allowed in the flowOCT. Every node that is assigned a label is called terminal node. Decision variable p_t is used to indicate the selection of terminal node. Furthermore, as shown in Figure 1, we add a node ζ as a dummy sink connecting with every nodes in the decision tree. We regard correctly classifying a data point as routing a unit of flow from the root of the decision tree to this dummy sink. They use decision variable $h^i_{a(n),n}$ to describe the flow routing associated with each data point.

- a_{tf}, if feature f is selected to split at branch node t $(= 1)$ or not $(= 0)$
- c_{tk}, if label k is assigned to node t $(= 1)$ or not $(= 0)$
- p_t, if branch node t is selected as a terminal node $(= 1)$ or not $(= 0)$
- $h^i_{a(n),n}$, if data point i is correctly classified and traverses the arc $(a(n), n)$ $(= 1)$ or not $(= 0)$

We then formulate the problem with a max-flow objective. The objective function (29) maximizes the weighted some of total flows reach the dummy sink and the penalties for splits at branch nodes. We change the format of the regularization to align with the aforementioned to methods. Constraints (30) make sure only one of the following can happen at each branch node: split the feature space based on a feature, select this node as a terminal node, and select one of the ancestor nodes as a terminal node. Constraints (31) guarantee labels are assigned only to terminal nodes. Constraints (32) ensure each data point correspond to a unit flow, and constraints (33) indicate such a flow can reach the dummy sink only if the corresponding data point is correctly classified. Constraints (34)-(37) illustrate the routing of each data point from the root node to the dummy sink, if possible. Constraints (38) and (39) describe the types and ranges of the decision variables.

$$\max \quad \frac{1}{\hat{e}} \sum_{i\in I, t\in L} h^i_{t\zeta} - \alpha \sum_{t\in B, f\in F} a_{tf} \tag{29}$$

$$\text{s.t.} \quad \sum_{f\in F} a_{tf} + p_t + \sum_{s\in A(t)} p_s = 1 \qquad \forall t \in B \tag{30}$$

$$\sum_{k\in K} c_{tk} = p_t \qquad \forall t \in B \cup L \tag{31}$$

$$h^i_{\xi 1} \le 1 \qquad \forall i \in I \tag{32}$$

$$h^i_{t\zeta} \le c_{ty^i} \qquad \forall i \in I, \forall t \in B \cup L \tag{33}$$

$$h^i_{a(t),t} = h^i_{t,l(t)} + h^i_{t,r(t)} + h^i_{t\zeta} \qquad \forall i \in I,\ \forall t \in B \tag{34}$$

$$h^i_{a(t),t} = h^i_{t\zeta} \qquad \forall i \in I,\ \forall t \in L \tag{35}$$

$$h^i_{t,l(t)} \le \sum_{f\in F: x^i_f = 0} a_{tf} \qquad \forall i \in I,\ \forall t \in B \tag{36}$$

$$h^i_{t,r(t)} \le \sum_{f\in F: x^i_f = 1} a_{tf} \qquad \forall i \in I,\ \forall t \in B \tag{37}$$

$$a_{t,f} \in \{0, 1\} \qquad \forall t \in B,\ \forall f \in F \tag{38}$$

$$c_{tk}, p_t, h_{a(t),t}, h_{t\zeta} \in \{0, 1\} \qquad \forall i \in I,\ \forall t \in B \cup L,\ \forall k \in K \tag{39}$$

The flowOCT is free of "big-M" thus has the potential to yield strong branch-and-bound performance. To further accelerate the solution processes, we employ a benders decomposition algorithm. We introduce

a new decision variable g^i for each data point indicating if the associated flow reaches the dummy sink or not. And then, the benders master problem is formulated as follows. Some facet-defining min-cuts are generated on the fly to describe the relationships among g^i, a_{tf}, and p_t. For the interest of brevity, we refer the readers to the original paper by Aghaei et al. (2021) for more details.

$$
\begin{aligned}
\max \quad & \frac{1}{\hat{e}} \sum_{i \in I} g^i - \alpha \sum_{t \in B, f \in F} a_{tf} & \\
\text{s.t.} \quad & \sum_{f \in F} a_{tf} + p_t + \sum_{s \in A(t)} p_s = 1 & \forall t \in B \\
& \sum_{k \in K} c_{tk} = p_t & \forall t \in B \cup L \\
& a_{tf} \in \{0,1\} & \forall t \in B,\ \forall f \in F \\
& p_t, g^i \in \{0,1\} & \forall i \in I,\ \forall t \in B \cup L
\end{aligned}
$$

Then, for $\forall i \in I$, g^i is the optimal objective value of the following problem.

$$
\begin{aligned}
\max \quad & \sum_{t \in L} h^i_{t\zeta} & \\
\text{s.t.} \quad & h^i_{t\zeta} \le c_{ty^i} & \forall i \in I, \forall t \in B \cup L \\
& h^i_{a(t),t} = h^i_{t,l(t)} + h^i_{t,r(t)} + h^i_{t,\zeta} & \forall t \in B \\
& h^i_{a(t),t} = h^i_{t\zeta} & \forall t \in L \\
& h^i_{t,l(t)} \le \sum_{f \in F: x^i_f = 0} a_{tf} & \forall t \in B \\
& h^i_{t,r(t)} \le \sum_{f \in F: x^i_f = 1} a_{tf} & \forall t \in B \\
& h_{a(t),t}, h_{t\zeta} \in \{0,1\} & \forall t \in B \cup L,\ \forall k \in K
\end{aligned}
$$

3 Stable Classification Tree

All the three papers we review in Section 2 follow the widely adopted model training pipeline: randomly split the given data set into training, validation, and test sets; train the model on the training set and tune the hyper-parameters based on model performance on the validation set; and finally, report performance on the test set. However, as presented in Section 4, the performance of optimal decision tree can be affected by data assignment to different sets, which is undesirable in practice. To tackle this challenge, we introduce the stable optimal classification tree (SOCT) which has the potential to yield robust performance with respect to the random training, validation, and test splits.

3.1 Formulation

The idea of SOCT is based on the "stable regression" proposed by Bertsimas and Paskov (2020). Instead of randomly splitting data into training and validation set, we integrate this selection into the model training processes. In particular, let P_{flow} denote the formulation of flowOCT. We introduce a new decision variable q^i for each data point $i \in I$.

$$
q^i = \begin{cases} 1, & \text{data point } i \text{ belongs to the training set,} \\ 0, & \text{otherwise.} \end{cases}
$$

We then define the stable flowOCT as follows.

$$
\begin{aligned}
\max_{a,c,p,h \in P_{flow}} \frac{1}{\hat{e}} \min_{q} \quad & \sum_{i \in I} \sum_{t \in B \cup L} q^i h^i_{t\zeta} - \alpha \sum_{t \in B, f \in F} a_{tf} \\
\text{s.t.} \quad & \sum_{i \in I} q^i = \mathcal{N} \\
& q^i \in \{0,1\}, \quad \forall i \in I
\end{aligned}
$$

This formulation can be interpreted as training the classification tree on the "hardest" subset of size $\mathcal{N}$ of the given dataset. The trained model thus should be robust to any subset of the given dataset. However, the non-linear terms in the objective function impose new computational challenges to this problem, we next introduce two potential methods to linearize the objective function without changing the structure of the original flowOCT formulation such that the benders decomposition algorithm is still applicable.

3.2 A Robust Optimization Solution Method

In this sub-section, we apply the robust optimization method similar to Bertsimas and Paskov (2020) to solve the SOCT. We observe that if we relax the integrality constraints imposed on q's and assume all the decision variables associated with the original flowOCT problem, i.e. a, c, p, and h, are fixed, the coefficient matrix of the inner problem would be totally unimodular (proof in appendix). Given that the right-hand sides of such a relaxed problem are all integer, solving the original stable tree formulation is equivalent as solving the following relaxed problem.

$$\begin{aligned}\max_{a,c,p,h\in P_{flow}} \ \frac{1}{\hat{e}} \ \min_{q} \quad & \sum_{i\in I}\sum_{t\in B\cup L} q^i h^i_{t\zeta} - \alpha \sum_{t\in B}\sum_{f\in F} a_{tf} \\ \text{s.t.} \quad & \sum_{i\in I} q^i = \mathcal{N} \\ & 0 \le q^i \le 1, \quad \forall i \in I\end{aligned}$$

We then associate dual variables θ and u's with the two constraints and write the dual of the inner minimization problem as:

$$\begin{aligned}\max \quad & \mathcal{N}\theta + \sum_{i\in I} u^i & \\ \text{s.t.} \quad & \theta + u^i \le \sum_{t\in B\cup T} h^i_{t\zeta} & \forall i \in I \\ & u_i \le 0 & \forall i \in I\end{aligned}$$

The SOCT is then equivalent as the following linear mixed integer programming problem.

$$\begin{aligned}\max_{a,c,p,h\in P_{flow}} \quad & \frac{1}{\hat{e}}(\mathcal{N}\theta + \sum_{i\in I} u^i) - \alpha \sum_{t\in B}\sum_{f\in F} a_{tf} & \\ \text{s.t.} \quad & \theta + u^i \le \sum_{t\in B\cup T} h^i_{t\zeta} & \forall i \in I \\ & u_i \le 0 & \forall i \in I\end{aligned}$$

This problem can be solved in a similar fashion as flowOCT. We initialize the problem with decision variables that describe the tree structure, i.e. a, c, and p, and the dual variables θ and u as well as constraints in the benders' master problem. Every time we encounter a integer solution in branch-and-bound, we apply the same algorithm as illustrated in Aghaei et al. (2021) to search a source set S for each data point and add the following min-cut to the benders master problem if violated.

$$\theta + u^i \le \sum_{(n_1,n_2)\in C(S)} c^i_{n_1,n_2}(b,w)$$

As presented in Section 4, this robust optimization method is able to solve the SOCT problem relative efficiently for small dataset on small trees. However, it requires an additional decision variable u^i for each data point, which hinders its scalability. We next introduce a cutting plane method that has less decision variables.

3.3 A Cutting Plane Solution Method

To apply the cutting method, we introduce a new decision variable μ indicating the objective value for the inner minimization problem in SOCT and re-write the SOCT as follows.

$$\max_{\substack{w,b,p,z\in P_{opt}\\ t\ge 0}} \quad \frac{1}{\hat{e}}\mu - \alpha \sum_{t\in B, f\in F} a_{tf} \tag{40}$$

$$\text{s.t.} \quad \mu \le \sum_{i\in I}\sum_{t\in B\cup T} q^i_j h^i_{t\zeta} \qquad \forall q_j \in Q_{\mathcal{N}} \tag{41}$$

where $Q_{\mathcal{N}} = \{q \in [0,1]^{|F|} : \sum_{j\in F} q_j = \mathcal{N}\}$ is the set of all feasible q's corresponding to selections of data points for the training set. Given the size of $Q_{\mathcal{N}}$, it is impractical to enumerate all the elements in the set. Instead, we add the constraints as cutting planes.

Specifically, similar to the robust optimization method, we first initialize the problem with constraints from the benders master problem of the flowOCT formulation and apply the same algorithm to add min-cut every time we encounter an integer solution. In addition, at each integer solution, we also check the sets of data points that are correctly and wrongly classified, denoted by D_{mis} and D_{corr} respectively. Given the integer solution, the minimal right-hand-side of the constraints in (41) would be $\max\{0, \mathcal{N} - |D_{mis}|\}$ corresponding to the case where we put as many data points in D_{mis} as possible to the training set and fill the remaining positions with data points from D_{corr}. Therefore,

- if $\mu \leq \max\{0, \mathcal{N} - |D_{mis}|\}$, all the constraints in (41) are satisfied,
- otherwise, we should add the following cutting planes, namely *worst-cuts*:

$$\mu \leq \sum_{i \in D_{mis}} \sum_{t \in B \cup T} h^i_{t\zeta} + \sum_{i \in D'} \sum_{t \in B \cup T} h^i_{t\zeta} \quad \forall D' \subset D_{corr},\ |D'| = \mathcal{N} - |D_{mis}|.$$

It is worth noting that adding all worst cuts as described above requires enumerating all subsets of D_{corr}. It can potentially lead to a large number of constraints that impose computational burdens to the benders master problem, especially when $|D_{mis}| \ll |D_{corr}|$. To accelerate the solution process, we employ the following two tricks in our implementation:

- *If a min-cut is added, we do not consider adding any worst cut at the same integer solution.*
- *If no min-cut is added and worst cuts are needed, we randomly generate a subset of D_{corr} of size $\mathcal{N} - |D_{mis}|$ and add the corresponding worst cut.*

The intuition behind the first trick is that, the solutions we obtain at earlier phases of the solution processes are likely to violate lots of constraints leading to a large number of cuts. However, if we first focus on satisfying the original flowOCT constraints without considering worst-cuts, we can hopefully obtain better-behaved solutions that violate less constraints in (41). Consequently, less worst-cuts would be needed. On the other hand, the second trick is based on the observation that lots of worst cuts at the same integer solution are redundant. Although both tricks can potentially result in more branch-and-bound node to explore, they significantly speed up the solution processes according to our computational experiment.

4 Computational Experiments

In this section, we present the computational experiments for the aforementioned formulations and solution methods. We first introduce the experiment setup in subsection 4.1, followed by performance comparisons among the OCT, binOCT and flowOCT formulations as well as CART (Breiman et al. 1984) focusing on solution time, in-sample, and out-of-sample prediction accuracy in subsection 4.2. The performance of the stable classification tree is presented in subsection 4.3.

Table 4: Summary of datasets

Dataset	$\lvert\mathcal{I}\rvert$	$\lvert\mathcal{F}\rvert$	$\lvert\mathcal{K}\rvert$
soybean-small	47	45	4
monk3	122	15	2
monk1	124	15	2
hayes-roth	132	15	3
monk2	169	15	2
house-votes-84	232	16	2
spect	267	22	2
breast-cancer	277	38	2
balance-scale	625	20	3
tic-tac-toe	958	27	2
car-evaluation	1728	20	4

4.1 Experiment Setup

Dataset: Our computational experiments are based on the datasets from UCI data respository (Dua and Graff 2017) which are summarized in Table 4. Aghaei et al. (2021) employ the same set of datasets to evaluate OCT, binOCT and flowOCT by one-hot encoding all the features before feeding datasets to the three formulations. However, this method can potentially hinder the performance of OCT and binOCT as they are actually able to deal with discrete features. Using binary features would largely increase the feature space, deeper trees (and longer solution time) might be needed to achieve the same level of performance as using original features. We thus perform one-hot encoding only for flowOCT and use the original features for OCT, binOCT, and CART.

Model Training: We create 3 random splits for each dataset into training (50%), validation (25%), and test sets (25%). For each split, we train the three models for each depth $d \in \{2, 3, 4, 5\}$, and each regularization parameter $\alpha \in \{0, 0.01, 0.1\}$ on the training set. For each model and depth, we select the α with the best performance on the validation set and report the associated in-sample and out-of-sample performance. We implement CART on each split using scikit-learn (Pedregosa et al. 2011) with default hyper-parameters.

Testing Methodology for Stable Classification Tree: We compare the in-sample and out-of-sample performance of the stable classification tree and flowOCT which is the most efficient one among

Table 5: Mean out-of-sample prediction accuracy for OCT, binOCT, flowOCT, and CART

instance	depth	OCT	binOCT	flowOCT	CART
soybean-small	2	0.8148	**0.9722**	0.9630	0.7778
soybean-small	3	0.9537	0.7500	0.9444	**1.0000**
soybean-small	4	0.8704	0.8333	**0.9352**	0.9167
soybean-small	5	0.8704	0.7222	0.9444	**1.0000**
monks-3	2	0.9384	0.8345	**0.9664**	**0.9664**
monks-3	3	0.9680	0.9113	**0.9824**	0.9640
monks-3	4	0.9824	0.6739	0.9824	**0.9904**
monks-3	5	0.9824	0.6835	0.9824	**0.9904**
monks-1	2	0.7434	0.7050	**0.7442**	0.7314
monks-1	3	0.8034	**0.8393**	0.8161	0.7890
monks-1	4	0.8457	0.9041	**0.9105**	0.7770
monks-1	5	0.8185	0.8993	**0.9105**	0.7602
hayes-roth	2	0.5806	0.5500	**0.5972**	0.5333
hayes-roth	3	**0.7028**	0.7000	0.6861	0.6083
hayes-roth	4	0.7417	0.6417	**0.7611**	0.6917
hayes-roth	5	0.7556	0.6167	0.7750	**0.7833**
monks-2	2	0.6461	0.6137	0.6468	**0.6623**
monks-2	3	**0.6328**	0.5960	0.6034	0.5717
monks-2	4	**0.6313**	0.5784	0.6181	0.6026
monks-2	5	0.6269	**0.7660**	0.6394	0.7064
house-votes-84	2	**0.9713**	0.9655	**0.9713**	**0.9713**
house-votes-84	3	0.9674	**0.9713**	0.9674	**0.9713**
house-votes-84	4	**0.9617**	0.9253	**0.9617**	0.9598
house-votes-84	5	**0.9693**	0.9598	0.9617	0.9598
spect	2	**0.7993**	0.7811	**0.7993**	0.7811
spect	3	0.7910	0.7562	0.7794	**0.8060**
spect	4	0.7297	0.7413	**0.7844**	0.7811
spect	5	**0.7894**	0.7214	0.7811	0.7662
breast-cancer	2	0.6873	0.6857	0.6825	**0.7095**
breast-cancer	3	0.7048	0.7143	0.6952	**0.7286**
breast-cancer	4	0.6825	0.7048	0.7095	**0.7143**
breast-cancer	5	0.6921	0.6286	**0.7016**	0.6714
balance-scale	2	0.6638	0.6497	**0.6667**	0.6285
balance-scale	3	0.6667	0.6603	0.6900	**0.7197**
balance-scale	4	0.6971	0.6624	0.6907	**0.7707**
balance-scale	5	0.6624	0.5754	0.6858	**0.7707**
tic-tac-toe	2	0.6861	0.6889	0.6903	**0.7097**
tic-tac-toe	3	0.7269	**0.7458**	0.7366	0.7125
tic-tac-toe	4	0.7417	0.7875	0.7287	**0.8000**
tic-tac-toe	5	0.7079	0.7944	0.7236	**0.8097**
car-evaluation	2	0.7418	0.7654	0.7418	**0.7739**
car-evaluation	3	0.7446	**0.7863**	0.7639	0.7724
car-evaluation	4	0.7449	0.8333	0.7711	**0.8364**
car-evaluation	5	0.7022	0.8279	0.7631	**0.8341**

the three MIP formulations. For the sake of time, we fix the tree depth as 2. We perform 10 random 50/25/25 splits for each dataset. For each split, we use the method described above to train flowOCT on the training set and determine hyper-parameter α based on model performance on the validation set. For SOCT, we train the model on the union of training and validation sets and let the model to determine the training and validation split. The value of α is also determined based on performance on the validation set. We compare the average out-of-sample prediction accuracy and the associated standard deviation over the 10 trails on each dataset for flowOCT and the SOCT.

Test Environment: All the tests are conducted on Intel(R) Core(TM) CPU i7-7700HQ @ 2.80GHz

Table 6: Training time for OCT, binOCT, flowOCT, and CART (seconds)

instance	depth	OCT	binOCT	flowOCT	CART
soybean-small	2	1.11	0.25	0.32	0.00
soybean-small	3	2.09	0.20	0.46	0.00
soybean-small	4	4.46	0.48	0.43	0.00
soybean-small	5	7.53	1.34	0.69	0.00
monks-3	2	11.89	0.69	0.79	0.02
monks-3	3	391.99	40.54	5.36	0.00
monks-3	4	417.42	601.01	203.52	0.00
monks-3	5	449.67	602.93	201.92	0.00
monks-1	2	62.27	2.26	1.64	0.00
monks-1	3	600.74	364.60	20.16	0.00
monks-1	4	601.94	601.16	18.56	0.00
monks-1	5	604.41	29.07	19.90	0.00
hayes-roth	2	13.32	1.51	0.63	0.00
hayes-roth	3	600.26	476.06	10.12	0.00
hayes-roth	4	600.63	600.87	274.22	0.00
hayes-roth	5	601.47	601.82	570.76	0.00
monks-2	2	164.98	5.55	11.45	0.00
monks-2	3	600.79	601.06	405.49	0.00
monks-2	4	601.97	601.56	407.86	0.00
monks-2	5	604.79	603.65	405.16	0.00
house-votes-84	2	2.99	0.55	0.31	0.00
house-votes-84	3	39.57	383.97	6.65	0.00
house-votes-84	4	175.62	203.04	14.16	0.00
house-votes-84	5	228.76	4.77	5.55	0.00
spect	2	5.11	4.62	1.10	0.00
spect	3	305.43	419.86	94.97	0.00
spect	4	402.07	601.38	298.74	0.00
spect	5	403.85	604.00	400.11	0.00
breast-cancer	2	49.98	9.50	2.96	0.00
breast-cancer	3	566.47	601.07	401.79	0.00
breast-cancer	4	598.23	601.45	401.25	0.00
breast-cancer	5	561.77	603.09	401.46	0.00
balance-scale	2	199.01	5.50	1.68	0.00
balance-scale	3	600.82	601.13	90.03	0.00
balance-scale	4	602.13	601.79	419.37	0.00
balance-scale	5	604.82	603.78	496.10	0.00
tic-tac-toe	2	323.67	105.23	35.48	0.00
tic-tac-toe	3	601.56	601.49	437.20	0.00
tic-tac-toe	4	604.04	603.74	432.64	0.00
tic-tac-toe	5	610.13	612.51	427.99	0.00
car-evaluation	2	546.85	13.35	24.65	0.00
car-evaluation	3	602.16	601.48	416.04	0.00
car-evaluation	4	605.61	603.61	428.08	0.00
car-evaluation	5	616.89	610.86	443.64	0.00

and a memory of 16 GB. Algorithms are implemented in Python 3.7 with Gurobi 9.1 as an optimization solver. The time limit is set to 600 seconds. Our implementations can be found online at `https://github.com/LucasBoTang/MIP_Decision_Tree`.

4.2 Performance of Different MIP formulations

Out-of-Sample Performance: The mean prediction accuracy on the test set of OCT, binOCT, flowOCT and CART is summarized in Table 5 where datasets are ordered according to their sizes. The

three MIP-based methods beat CART in 22 out of the 44 cases, mostly on small datasets. When it comes to large datasets, CART is more efficient in finding a good solution. This is partially because we choose a time limit of 10 minutes which is much less then the 60 minutes time limit used by Aghaei et al. (2021). However, since CART solve all the problems instantaneously, we believe this is not an unfair comparison.

As for the comparison between MIP-based methods, the OCT showcases strong performance on small datasets and shallow trees (depth ≤ 3), while binOCT achieves highest prediction accuracy on the six biggest problems ("tic-tac-toe" and "car-evaluation" with depth of 2, 3 and 4) partially because its formulation size is largely independent of dataset sizes. The flowOCT accomplish the highest prediction accuracy on 19 out of 44 cases which is the best over all approaches. In general, the flowOCT yields the most consistent performance across datasets and tree complexity. In contrast, the performance of OCT and binOCT is less stable as they are not solved to optimality within the time limit for lots of cases. However, when training a small dataset on shallow trees, they could be competitive as their feature representations are more compact and informative than those for flowOCT.

Training Time: Table 6 presents the average training time of the four approaches. The solution time for flowOCT are way less than the time limit (600 seconds), implying that it is solved to optimality in most cases. The OCT and binOCT can solve small datasets on shallow trees efficiently, yet hits the time limit for most cases where tree depth is 4 or more. The training time for CART is almost zero for all the cases.

4.3 Performance of Stable Classification Tree

Average Prediction Accuracy: The out-of-sample performance of flowOCT and SOCT implemented with the robust optimization and cutting plane methods are presented in Table 7. For six out of 11 datasets, the SOCT outperforms flowOCT in terms of average prediction accuracy, while flowOCT yields better performance than SOCT in only one dataset. We note that the cutting plane and robust optimization methods yield different results for "monk1", "hayes-roth" and "breast-cancer" because under certain test-training splits, there exists multiple optimal solutions for the SOCT and the two solution methods can lead to different solutions. In general, the SOCT achieves better out-of-sample prediction performance than the flowOCT.

Standard Deviation of Prediction Accuracy: The SOCT formulation has lower standard deviation than the flowOCT formulation in four out of 11 datasets, while flowOCT is less variant in three of them. The results in Table 7 suggest that the improvement in prediction stability due to the SOCT formulation is marginal.

Table 7: Out-of-sample prediction performance for flowOCT, SOCT(CP) and SOCT(Robust)

	Mean Accuracy			Standard Deviation of Accuracy		
Instance	flowOCT	SOCT(CP)	SOCT(Robust)	flowOCT	SOCT (CP)	SOCT (Robust)
soybean-small	0.9667	**1.0000**	**1.0000**	0.1054	**0.0000**	**0.0000**
monk3	0.9419	0.9419	0.9419	0.0366	0.0366	0.0366
monk1	0.7032	**0.7677**	0.7613	**0.0677**	0.0959	0.0876
hayes-roth	**0.6227**	0.6176	0.6167	**0.0609**	0.0801	0.0798
monk2	0.6186	**0.6395**	**0.6395**	0.0622	**0.0481**	**0.0481**
house-votes-84	0.9707	0.9707	0.9707	0.0083	0.0083	0.0083
spect	0.7940	0.7940	0.7940	0.0280	0.0280	0.0280
breast-cancer	0.7000	**0.7500**	0.7400	0.0526	**0.0310**	0.0355
balance-scale	0.6605	**0.6694**	**0.6694**	**0.0235**	0.0301	0.0301
car_evaluation	0.7757	0.7757	0.7757	0.0160	0.0160	0.0160
tic-tac-toe	0.6817	**0.6821**	**0.6821**	0.0338	**0.0161**	**0.0161**

Solution Time: The solution time for flowOCT and SOCT implemented in the two solution methods are illustrated in Table 8. Incorporating "robustness" considerations into the flowOCT formulation leads to much longer solution time, especially for datasets with larger sample sizes such as "tic-tac-toe" and "car-evaluation". Regarding the two solution methods for SOCT, the robust optimization method showcases better computational performance. It outperforms the cutting plane method in 10 out of the 11 datasets. One possible reason is that the cut sampling method we employed is not able to efficiently cut-off non-integer solutions, resulting in a large number of branch-and-bound nodes to explore. Future research can be conducted to develop more efficient and effective cut sampling methods for the cutting plane implementation.

5 Conclusion

In this project, we implement the three most famous MIP-based decision tree approaches and compare their out-of-sample performance and training time on several publically available classification datasets. The results show that the flowOCT achieves the best out-of-sample prediction accuracy among the three MIP-based methods. It showcases the most consistent performance across datasets and tree complexity partially because it is the only MIP formulation that can solve most cases to optimality within the time limit allotted by the test methodology. The OCT binOCT is competitive on small datasets and shallow trees

Table 8: Solution Time for flowOCT and SOCT (seconds)

Instance	flowOCT	SOCT (CP)	SOCT (Robust)
soybean-small	0.32	0.62	1.28
monk3	0.32	0.79	0.62
monk1	0.47	1.18	0.81
hayes-roth	0.67	4.10	1.23
monk2	0.77	5.81	2.66
house-votes-84	0.58	1.31	1.20
spect	1.80	5.67	5.14
breast-cancer	6.18	16.53	11.84
balance-scale	3.30	15.97	7.17
tic-tac-toe	71.50	246.28	88.11
car_evaluation	24.16	108.59	54.20

as they can deal with continuous feature representations which are more compact and informative than binary representation employed by flowOCT. However, the performance of OCT deteriorate dramatically as dataset size and tree complexity increase, while binOCT has the potential to achieve good performance on large datasets as its formulation is largely independent of dataset size. The three MIP-based methods have comparable out-of-sample prediction accuracy as CART on small datasets, yet performs worse on large datasets. Moreover, the training time of MIP-based method is generally much longer than CART.

The proposed stable classification tree incorporates the selection of training and validation sets into the modelling training processes. Computational experiments illustrate its capability to achieve more stable out-of-sample prediction power than the traditional training method, i.e. randomly split the given dataset into training and validation sets. Nevertheless, solving stable classification tree is computationally demanding. Developing more efficient algorithms to solve SOCT presents an interesting direction for future research.

References

Aghaei, S., Gómez, A., and Vayanos, P. (2021). Strong optimal classification trees. *arXiv preprint arXiv:2103.15965*.

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

Bertsimas, D. and Paskov, I. (2020). Stable regression: On the power of optimization over randomization in training regression problems. *Journal of Machine Learning Research*, 21:1–25.

Breiman, L., Friedman, J., Stone, C. J., and Olshen, R. A. (1984). *Classification and regression trees*. CRC press.

Dua, D. and Graff, C. (2017). UCI machine learning repository.

Firat, M., Crognier, G., Gabor, A. F., Hurkens, C. A., and Zhang, Y. (2020). Column generation based heuristic for learning classification trees. *Computers & Operations Research*, 116:104866.

Günlük, O., Kalagnanam, J., Menickelly, M., and Scheinberg, K. (2018). Optimal decision trees for categorical data via integer programming. *arXiv preprint arXiv:1612.03225*.

Hu, X., Rudin, C., and Seltzer, M. (2019). Optimal sparse decision trees. *Advances in Neural Information Processing Systems (NeurIPS)*, 32.

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

Pedregosa, F., Varoquaux, G., Gramfort, A., Michel, V., Thirion, B., Grisel, O., Blondel, M., Prettenhofer, P., Weiss, R., Dubourg, V., Vanderplas, J., Passos, A., Cournapeau, D., Brucher, M., Perrot, M., and Duchesnay, E. (2011). Scikit-learn: Machine learning in Python. *Journal of Machine Learning Research*, 12:2825–2830.

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

Verwer, S. and Zhang, Y. (2019). Learning optimal classification trees using a binary linear program formulation. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 33, pages 1625–1632.

Zantedeschi, V., Kusner, M. J., and Niculae, V. (2020). Learning binary trees via sparse relaxation. *arXiv preprint arXiv:2010.04627*.