# Optimal Classification Tree Case Studies.

OCT Project Report
Xiaoyan Lin, Zhuoqiao Ouyang
Advisor: Anton Malandii
Stony Brook University

August 2025

## 1 Abstract

Decision trees have become well-known tools in machine learning for solving classification problems. Given a set of classification data $X$ of $p$ features and targets $Y$ of $K$ classes, a decision tree is typically implemented as a binary tree, where each internal node applies a binary test to a particular feature and threshold. Depending on the features and threshold, the data point is routed to either the left or right branch. Each leaf node is assigned a predicted label, meaning every path from the root to a leaf defines a classification rule for data points that follow the path.

The most commonly implemented decision trees algorithm with interpretability is the Classification and Regression Tree (CART). CART resembles the state-of-the-art decision tree method, which relies on greedy, top-down heuristics that may lead to suboptimal solutions in terms of predictive performance. *Optimal Classification Tree*, introduced by Bertsimas and Dunn [1], is a more novel approach for classification problems. It aimed at formulating the construction process of the decision tree as one single mixed integer problem and building an optimal interpretable decision tree through solving the MIP optimally.

In this project, we formulate and solve the OCT problem in Python using the Gurobi optimization package. We apply this fitting algorithm to various real data and compare its predicting performance with those of CART, Random Forest and XGBoost. Finally, we discuss some potential methods and techniques that could enhance the performance of OCT.

## 2 Introduction

A Tree is a non-linear data structure where nodes are organized in a hierarchy. The root node is at the top of the tree and has only outgoing edges. Nodes at the bottom of the tree with no children are called leaf nodes, they only have incoming edges. Nodes that lie between the root and the leaves are called branch nodes or internal nodes; they have both incoming and outgoing edges. Any node with outgoing edges is a parent node, while nodes with incoming edges from the parent node are its children. A node can be both a parent node and a child node. A left ancestor of a node $t$ is any node that lies on the path from the root to $t$ and such that it lies on the left side of its parent node; a right ancestor has a similar definition, but it lies on the right side of its parent node. We denote the parent, the set of left ancestors and set of right ancestors of a node $t$ by $p(t)$, $A_L(t)$, and $A_R(t)$, respectively. The size of the tree refers to the total number of nodes, while the depth (or height) $D$ is the number of edges from the root to the deepest leaf. A Binary tree is a specific type of tree in which each node has at most two child nodes, typically labeled as the left and right child nodes; and thus the maximum number of nodes in a binary tree is $T = 2^{D+1} - 1$ and the maximum number of leaf nodes is $2^D$. In particular, we label the nodes of the binary tree such that the set of branch nodes $\mathcal{T}_B$ is $\{1, 2, \ldots, \lfloor T/2 \rfloor\}$ and the set of leaf nodes $\mathcal{T}_L$ is $\{\lfloor T/2 \rfloor + 1, \ldots, T\}$.

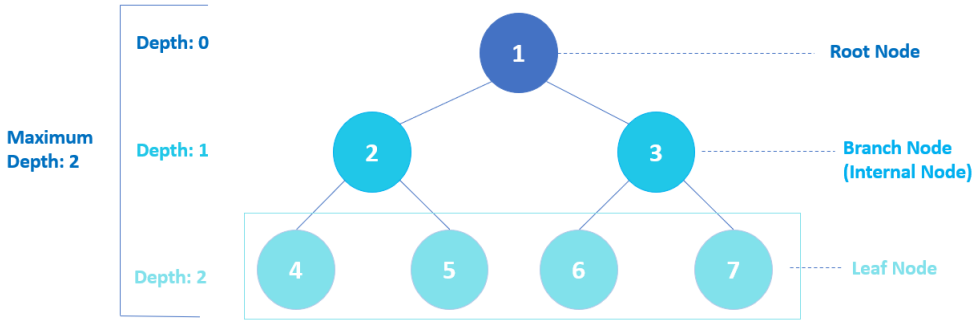


**Figure 1.** Illustration of a Binary Tree.

One important application of binary trees in machine learning is the decision tree or classification tree for modeling and automating decision-making processes. A well-known implementation of this approach is Classification and Regression Trees (CART) for its interpretability and efficiency. Given a classification data set with certain features, a decision tree makes decisions by recursively splitting the dataset univariately, where each split is based on a threshold value on only one of the features. The root node represents the entire dataset and serves as the starting point for analysis. At each step, the data is split into two or more sub-nodes, aiming to improve the classification or prediction accuracy. More specifically, CART selects a feature $j$ and a threshold value $b$ at each branch node $t$ as the set of splitting criterion, if for point $i$ $x_{ij} < b$, then it results in the left child node, else the right child node $t$. These sub-nodes are also called decision nodes if they continue to split. The process continues until any of the stopping criteria are met, resulting in leaf nodes, which are given class labels to represent points that fall into the nodes. To prevent overfitting, the technique called pruning is implemented to remove sub-nodes that do not provide significant improvement; and the complexity parameter $\alpha$, which balances the additional complexity of adding the split at the node against the increase in predictive accuracy that it offers, is used to decide whether to prune a node.

Traditional decision trees such as CART, ID3, and C4.5 rely on greedy top-down heuristics to produce good splits at each node. At each node, CART selects the best split based on a local impurity measure, aiming to minimize immediate impurity such as the Gini impurity metric, which evaluates how well a decision tree split separates the data. This local optimization strategy often fails to consider splits with lower impurity that could potentially lead to stronger splits in the latter nodes and thus fails to consider the tree's global optimality. Furthermore, it has the limitation that a two-stage process is required – constructing the tree from top down and pruning the tree. Other more recent machine learning algorithms for classification such as Neural Network, Random Forest and XGBoost, while provide good classification results, but are difficult or impossible to explain which features influence their decisions. This lack of interpretability can lead to issues such as overfitting or failing to provide meaningful and reliable solutions in solving real life problems.

In general, the traditional decision tree lacks global optimality as it seeks to minimize impurity rate at each split while the tree has the objective to minimize misclassification rate. The limitation, as noted by Breiman et al. (1984) [1], was due to lack of computational power at the time to search over all possible splits: "at this stage of computer technology, an overall optimal tree growing procedure does not appear feasible for any reasonably sized dataset". It was theoretically better to formulate the tree construction process as one single optimization problem, but it was not practical to solve. Nowadays, with improvements in the computational power in solving Mixed Integer Optimization problems (MIO) and with advanced MIO solvers such as GUROBI [2] and CPLEX [3], searching for an optimal tree becomes possible. In the paper *Optimal Classification Trees*, Bertsimas and Dunn [1]

introduced the Optimal Classification Trees, a new formulation of the decision tree problem using modern MIO techniques that produces optimal decision trees with univariate and multivariate splits.

# 3 Optimal Classification Trees (OCT)

## 3.1 Problem Formulation of OCT

In *Optimal Classification Tree*, Bertsimas and Dunn [1] stated the OCT as a problem that CART attempts to solve as a formal mixed integer optimization problem, aiming at determining the splitting criterion not necessarily give the optimal impurity rate at each split but optimally reduces the decision tree's misclassification loss. Given the training data set $(X, y)$, OCT has the same objective as CART:

$$\min \quad R_{xy}(T) + \alpha|T| \quad \text{s.t.} \quad N_x(l) \geq N_{\min} \quad \forall l \in \mathcal{T}_L.$$

where, $R_{xy}(T)$ is the misclassification error of tree $T$ on the training data, $|T|$ represents the number of branch nodes, $\alpha$ is the complexity parameter, $N_x(l)$ is the number of data points in each leaf, and $N_{\min}$ is the minimum number of points required in any leaf.

We are required to make a series of discrete decisions when constructing a decision tree:

- At each new node, OCT must choose to split or to stop;
- If nodes choose to split, then a feature or variable must be chosen to split on;
- If nodes choose not to split, a class label must be chosen for the resulting leaf node;
- When classifying the training points according to the tree under construction, we must choose which leaf node a point will be assigned to such that the structure of the tree is preserved.

With these considerations in mind, let D be the depth of the tree and $\{(X, y) \mid X \in \mathbb{R}^{n\times p},\ y_i \in \mathbb{N}\cup\{0\}\}$ be the classification dataset with each feature $j, j = 1, \ldots, p$, normalized such that $x_{ij} \in [0, 1]$ $\forall i = 1, \ldots, n$. The OCT model is formulated in the following procedures.

1. Let the splitting decision and criterion be tracked on the branch nodes with variables $a$, $b$, and $d$ with constraints (2), (3), and (4). The position $j$ of 1 in $a_t$ indicates which feature $j$ and $b_t$ indicates the threshold value for such feature to split on. Note that if $d_t = 0$, then constraints (2) and (3) give the option for the node not to split by forcing $a_t = 0$, $b_t = 0$, and thus forcing the point to follow the right split. Constraint (5) enforces the hierarchical tree structure by preventing a branch node to split if its parent does not split.

$$a = \left\{ a_t \in \mathbb{R}^p,\ t \in \mathcal{T}_B \;\middle|\; \sum_{j=1}^{p}(a_{tj}) = 1 \right\},$$

$$b = \{b_t \in [0, 1],\ t \in \mathcal{T}_B\},$$

$$d = \{\mathbf{1}\{\text{node } t \text{ applies a split}\},\ t \in \mathcal{T}_B\},$$

$$\sum_{j=1}^{p}(a_{tj}) = d_t, \quad \forall t \in \mathcal{T}_B, \tag{2}$$

$$0 \leq b_t \leq d_t, \quad \forall t \in \mathcal{T}_B, \tag{3}$$

$$a_{tj} \in \{0, 1\}, \tag{4}$$

$$d_t \leq d_{p(t)}, \quad \forall t \in \mathcal{T}_B \backslash \{1\}. \tag{5}$$

2. Let the allocation of a point to the leaf nodes be tracked by the indicator variables $z$ and $l$ with constraints (6), (7) and (8) to enforce the minimum number of points in each leaf and that no point can be assigned to more than one leaf.

$$z = \{z_{it} = \mathbf{1}\{x_i \text{ is in node } t\},\ t \in \mathcal{T}_L, i = 1, \ldots, n\},$$

$$l = \{\mathbf{1}\{\text{leaf } t \text{ contains any points}\},\ t \in \mathcal{T}_L\},$$

$$z_{it} \leq l_t, \quad \forall t \in \mathcal{T}_L, \tag{6}$$

$$\sum_{i=1}^{n} z_{it} \geq N_{min} l_t, \quad \forall t \in \mathcal{T}_L, \tag{7}$$

$$\sum_{t \in \mathcal{T}_L} z_{it} = 1, \quad i = 1, \ldots, n. \tag{8}$$

3. Let constraints (9) and (10) enforce the splits that are required by the structure of the tree when assigning points to leaves, where if leaf node $t$ contains point $x_i$, then for all left ancestors $m$ of node $t$, their splits must follow that $a_m^\top x_i < b_m$ and their right ancestors' splits must follow $a_m^\top x_i \geq b_m$. Since the MIO solver does not support strict inequality, Bertsimas et al. suggest adding a small constant number $\epsilon$ to the left-hand side of (9). Such value should be as large as possible without affecting the feasibility of the solution. We specify an $\epsilon_j$ for each feature $j$ and let $\mathbf{x}_j$ denotes the sorted $jth$ feature, then

$$\epsilon_j = \min\left\{\mathbf{x}^{(j+1)} - \mathbf{x}^{(j)} \middle| \mathbf{x}^{(j+1)} \neq \mathbf{x}^{(j)}, i = 1, \ldots, n\right\}, j = 1, \ldots, p; \quad \epsilon_{\max} = \max_j\{\epsilon_j\}.$$

which gives constraint (11). Note that since $a_m^\top x_i + \epsilon_{max} - b_m \leq 1 + \epsilon_{max}$, $M_1 = 1 + \epsilon_{max}$, giving constraint (12). Similarly, since $b_m - a_m^\top x_i \leq 1$, $M_2 = 1$, giving constraint (13).

$$a_m^\top x_i < b_m + M_1(1 - z_{it}), \quad i = 1, \ldots, n \quad \forall t \in \mathcal{T}_L, \quad \forall m \in A_L(t), \tag{9}$$

$$a_m^\top x_i \geq b_m - M_2(1 - z_{it}), \quad i = 1, \ldots, n \quad \forall t \in \mathcal{T}_L, \quad \forall m \in A_R(t), \tag{10}$$

$$a_m^\top x_i + \epsilon_{max} \leq b_m + M_1(1 - z_{it}), \quad i = 1, \ldots, n \quad \forall t \in \mathcal{T}_L, \quad \forall m \in A_L(t), \tag{11}$$

$$a_m^\top x_i + \epsilon_{max} \leq b_m + (1 + \epsilon_{max})(1 - z_{it}), \quad i = 1, \ldots, n \quad \forall t \in \mathcal{T}_L, \quad \forall m \in A_L(t), \tag{12}$$

$$a_m^\top \geq b_m - (1 - z_{it}), \quad i = 1, \ldots, n \quad \forall t \in \mathcal{T}_L, \quad \forall m \in A_R(t). \tag{13}$$

4. Let $k = 1, \ldots, K$ indicate the class label represented by integers, and let the $Y$ matrix track the class label $y_i$ of each point by comparing it with label $k$, where $Y_{ik} = 1$ if $y_i = k$, otherwise $Y_{ik} = -1$. We track the number of points of label $k$ in leaf node $t$ with variables $N_{kt}$ by checking the labels of all points $x_i$ in $t$: if $y_i = k$, then $\frac{1}{2}(1 + Y_{ik})z_{it} = 1$, and otherwise $\frac{1}{2}(1 + Y_{ik})z_{it} = 0$. $N_{kt}$ is then expressed by constraint (14). The total number of points in node $t$, denoted by $N_t$, is then the sum of points of label $k, k = 1, \ldots, K$, which is represented by constraint (15).

$$Y_{ik} = \begin{cases} +1, & \text{if } y_i = k \\ -1, & \text{otherwise} \end{cases}, \quad k = 1, \ldots, K, \quad i = 1, \ldots, n,$$

$$N_{kt} = \frac{1}{2}\sum_{i=1}^{n}(1 + Y_{ik})z_{it}, \quad k = 1, \ldots, K, \quad \forall t \in \mathcal{T}_L, \tag{14}$$

$$N_t = \sum_{i=1}^{n} z_{it}, \quad \forall t \in \mathcal{T}_L. \tag{15}$$

5. Let $c_t$ be the variables that track the predicted class label of the leaf node $t$, which is the most popular class among the leaf, with constraint (16). We restrict each leaf node, if it contains any points, to have only one class prediction by introducing $c_{kt}$ as a binary variable and constraint (17):

$$c_{kt} = \mathbf{1}\{c_t = k\}, \quad \forall k = 1, \ldots, K, \quad \forall t \in \mathcal{T}_L,$$

$$c_t = \arg \max_{k=1,\ldots,K} \{N_{kt}\}, \quad \forall t \in \mathcal{T}_L, \tag{16}$$

$$\sum_{k=1}^{K} c_{kt} = l_t, \quad \forall t \in \mathcal{T}_L. \tag{17}$$

6. Finally, let $L$ denote the misclassifications loss for the leaf nodes, where $L_t$ is defined to be the number of points in leaf t less the number of points of the most popular class, as expressed in constraint (18).

$$L_t = N_t - \max_{k=1,\ldots,K} \{N_{kt}\}. \tag{18}$$

Note that since the variables $N_t$ track the number of points and $N_{kt}$ track the number of points of label $k$ in leaf $t$, we have the following observations:

$$\begin{cases} N_{k^*t} = \max\limits_k \{N_{kt}\}, & \text{if class label of } t \text{ is } k^*, \\ N_{k^*t} \leq \max\limits_k \{N_{kt}\}, & \text{otherwise.} \end{cases} \quad t \in \mathcal{T}_L;$$

and thus

$$\begin{cases} L_t = N_t - N_{k^*t}, & \text{if class label of } t \text{ is } k^*, \\ L_t \leq N_t - N_{k^*t}, & \text{otherwise.} \end{cases} \quad t \in \mathcal{T}_L.$$

The first case corresponds to $c_{k^*t} = 1$ and the second corresponds to $c_{k^*t} = 0$. To ensure that both cases are satisfied simultaneously, we observe that $-n \leq L_t \leq n$. Therefore, the constraints can be linearized as (19)-(21).

$$L_t \geq N_t - N_{kt} - n(1 - c_{kt}), \quad k = 1, \ldots, K, \quad \forall t \in \mathcal{T}_L, \tag{19}$$

$$L_t \leq N_t - N_{kt} + nc_{kt}, \quad k = 1, \ldots, K, \quad \forall t \in \mathcal{T}_L, \tag{20}$$

$$L_t \geq 0, \quad \forall t \in \mathcal{T}_L. \tag{21}$$

The objective of OCT is to minimize the sum of misclassifications loss $\sum_{t \in \mathcal{T}_L} L_t$ for each leaf node, normalized against the baseline accuracy of the entire dataset $\hat{L}$, which is defined as

$$\hat{L} = \frac{\text{Number of points of the most popular class of the entire dataset}}{n = \text{Total number of observations}},$$

plus the regularization term on the complexity of the tree defined by the complexity parameter $\alpha$ times the total number of splits $\sum_{t \in \mathcal{T}_B} d_t$.

Below is the complete MIO formulation of the OCT model.

**Objective:**

$$\min \frac{1}{\hat{L}} \sum_{t \in \mathcal{T}_L} L_t + \alpha \sum_{t \in \mathcal{T}_B} d_t \tag{22}$$

**Subject to:**

$$
\begin{aligned}
&L_t \geq N_t - N_{kt} - n(1 - c_{kt}), \quad k = 1, \ldots, K, \ \forall t \in \mathcal{T}_L \\
&L_t \leq N_t - N_{kt} + nc_{kt}, \quad k = 1, \ldots, K, \ \forall t \in \mathcal{T}_L \\
&L_t \geq 0, \quad \forall t \in \mathcal{T}_L \\
&N_{kt} = \tfrac{1}{2} \textstyle\sum_{i=1}^{n} (1 + Y_{ik}) z_{it}, \quad k = 1, \ldots, K, \ \forall t \in \mathcal{T}_L \\
&N_t = \textstyle\sum_{i=1}^{n} z_{it}, \quad \forall t \in \mathcal{T}_L \\
&\textstyle\sum_{k=1}^{K} c_{kt} = l_t, \quad \forall t \in \mathcal{T}_L \\
&a_m^\top x_i \geq b_m - (1 - z_{it}), \quad i = 1, \ldots, n, \ \forall t \in \mathcal{T}_L, \ \forall m \in A_R(t) \\
&a_m^\top (x_i + \epsilon) \leq b_m + (1 + \epsilon_{\max})(1 - z_{it}), \quad i = 1, \ldots, n, \ \forall t \in \mathcal{T}_L, \ \forall m \in A_L(t) \\
&\textstyle\sum_{t \in \mathcal{T}_L} z_{it} = 1, \quad i = 1, \ldots, n \\
&z_{it} \leq l_t, \quad \forall t \in \mathcal{T}_L \\
&\textstyle\sum_{i=1}^{n} z_{it} \geq N_{\min} l_t, \quad \forall t \in \mathcal{T}_L \\
&\textstyle\sum_{j=1}^{p} a_{tj} = d_t, \quad \forall t \in \mathcal{T}_B \\
&0 \leq b_t \leq d_t, \quad \forall t \in \mathcal{T}_B \\
&d_t \leq d_{p(t)}, \quad \forall t \in \mathcal{T}_B \setminus \{1\} \\
&z_{it}, \ l_t \in \{0, 1\}, \quad i = 1, \ldots, n, \ \forall t \in \mathcal{T}_L \\
&a_{tj}, \ d_t \in \{0, 1\}, \quad j = 1, \ldots, p, \ \forall t \in \mathcal{T}_B
\end{aligned}
$$

## 3.2 Settings and Methodologies for Numerical Experiments

In this project, we formulated the OCT class in Python with the GUROBI [2] optimization package and tested its performance with real classification datasets retrieved from the UCI Machine Learning Repository [4].

The difficulty of the OCT model depends on the number of variables $z_{it}$, which is $n2^D$. Since solving the MIO can be computationally intensive, the speed for solving the MIO problem and quality of the solution can be largely enhanced with a warm start, which is a feasible solution that provides a strong initial upper bound on the optimal solution, enabling more effective pruning of the tree and serving as a valuable starting point for local search heuristics. In particular, we look for a warm start for the variables $a$ and $b$, and they can be obtained from the feature indices and threshold values selected by the CART model.

There are three hyperparameter that we need to determine: $\alpha$, $D$, and $N_{min}$. As in the paper *Optimal Classification Tree* [1], we set $N_{\min} = \lfloor 0.05n \rfloor$, where $n$ is the number of observations. Due to the limited computational ability of a laptop, we set the maximum depth $D = 2$. $\alpha$ can be tuned to produce potentially better results by regularizing the tree size and the method for tuning $\alpha$ introduced in the paper is described in section 4.2. In this project, instead of following this method, we regularize the tree size by adding the following sparsity constraint to the OCT problem o restrict the total number of branching nodes and setting $\alpha = 0$ in the objective.

$$\sum_{t \in \mathcal{T}_B} d_t = C, \quad C \in \{1, \ldots, D\}$$

We further prove by numerical experiments that for any choice of $C \in \{1, \ldots, D\}$, there exists a choice of $\alpha$ that yields the same solution in the original problem.

For each of the chosen datasets, we implemented the 5-fold cross-validation technique. Each split contains 50% training set, 25% calibration set and 25% testing set. We fit each training set into the OCT model and calibrate $C$ on the calibration set; after obtaining the optimal $C^*$, we combine the training set and calibration set as one train set with $C^*$ to train the model and report the

performance on the testing set. The time limit to run each model is 6000 seconds. To make predictions on the testing set, we obtain the split criterion variables $a$ and $b$ from the MIO solver. Each data point is then passed through the reconstructed tree by starting at the root node and following the path determined by the decision rules: at each branch node $t$, the point $x_i$ follows the left split if ${a_t}^T x_i < b_t$ or otherwise the right split. This recursive traversal continues until a leaf node is reached. The misclassifications loss $L_t$ of a leaf $t$ is the number of points where their true labels do not match the class label $c_t$ determined by the fitted model. The accuracy of the OCT is defined by:

$$\text{Accuracy} = \frac{\sum_{t \in \mathcal{T}_L} (\text{Points in Node } t - L_t)}{\text{Total number of observations}}$$

We report the average training set accuracy and testing set accuracy. We also fit the dataset to CART, XGBoost, and Random Forest and report the average training set and testing set accuracies for these models.

### 3.3 Numerical Experiments Result

**Table 1** presents the complete results of CART, OCT, XGBoost, and Random Forest (100 trees) across 6 datasets. Among all the models, XGBoost and Random Forest have the highest training accuracies, , highlighting its strong fitting abilities. However, their testing accuracies are not as good as the training accuracies, indicating that these non-interpretable classifiers may be overfitting in some cases. OCT's testing accuracies are highly comparable to that of XGBoost and Random Forest across the evaluated datasets and always outperform CART. The average runtime of OCT varies widely across datasets, with particularly longer times for more complex datasets.

Table 1: Full Results for OCT

| **Data Sets** | Blood | Iris | Wine | Soy-Bean Small | Land Mines | Fertility |
|---|---|---|---|---|---|---|
| **ID** | 176 | 53 | 109 | 91 | 763 | 244 |
| **Observations** | 748 | 150 | 178 | 47 | 338 | 100 |
| **Features** | 4 | 4 | 13 | 35 | 3 | 9 |
| **Class** | 2 | 3 | 3 | 4 | 5 | 2 |
| | | | | | | |
| **CART** | | | | | | |
| Train Set Accuracy | 0.76 | 0.959 | 0.932 | 0.823 | 0.481 | 0.899 |
| Test Set Accuracy | 0.767 | 0.958 | 0.853 | 0.7 | 0.431 | 0.84 |
| | | | | | | |
| **OCT** | | | | | | |
| Avg Runtime | 72 | 12 | 101 | 2.4 | 281 | 8 |
| Gap(%) | 0 | 0 | 0 | 0 | 0 | 0 |
| Train Set Accuracy | 0.76 | 0.946 | 0.949 | 1 | 0.521 | 0.893 |
| Test Set Accuracy | 0.767 | 0.968 | 0.942 | 0.967 | 0.482 | 0.84 |
| | | | | | | |
| **XGBoost** | | | | | | |
| Train Set Accuracy | 0.831 | 1 | 1 | 1 | 0.9 | 0.97 |
| Test Set Accuracy | 0.791 | 0.947 | 0.987 | 0.967 | 0.518 | 0.816 |
| | | | | | | |
| **Random Forest** | | | | | | |
| Train Set Accuracy | 0.772 | 0.952 | 0.98 | 1 | 0.527 | 0.893 |
| Test Set Accuracy | 0.764 | 0.974 | 0.996 | 0.967 | 0.4 | 0.84 |
| | | | | | | |
| Depth = 2 | | | | | | |

**Table 2** presents a comparison between CART and OCT models across 6 datasets with varying sample sizes ($n$), feature counts ($p$), and class labels ($K$) counts under a fixed tree depth of 2. The results indicate that OCT generally provides a better fit compared to CART, with OCT wins in 4 cases and 2 ties with CART. In particular, OCT achieves significant improvements on datasets such as Soybean Small.

Table 2: OCT vs. CART

| **Data Sets** | | | | **Testing Set Accuracy** | | **Improvement** |
|---|---|---|---|---|---|---|
| **Name** | **n** | **p** | **K** | **CART** | **OCT** | **OCT - CART** |
| Blood | 748 | 4 | 2 | 0.767 | 0.767 | 0 |
| Iris | 150 | 4 | 3 | 0.958 | 0.968 | 0.01 |
| Wine | 178 | 13 | 3 | 0.853 | 0.942 | 0.089 |
| Soy-Bean Small | 47 | 35 | 4 | 0.7 | 0.967 | 0.267 |
| Land Mines | 338 | 3 | 5 | 0.431 | 0.482 | 0.051 |
| Fertility | 100 | 9 | 2 | 0.84 | 0.84 | 0 |

In general, OCT is more likely to outperform CART. This result is consistent with the results presented in the paper in terms of the number of wins by OCT. Furthermore, we observe that the OCT has comparable accuracy with Random Forest and XGBoost; while Random Forest and XGBoost provide higher out of sample accuracy, they sacrifice the interpretability that OCT offers.

# 4 Improvement

There are multiple adjustable factors and hyperparameter that could impact the performance of the OCT model – the depth of the tree, the quality of the warmstart and the complexity parameter alpha. Other extended methods, such as the OCT with Hyperplane and Strong Optimal Classification Tree, provide a fundamental improvement based on the OCT model.

## 4.1 Adjust Depth of Tree $D$

The depth of the tree controls the overall complexity of the tree structure, and the ability of the OCT model to recognize relationships within the dataset increases as the depth increases. Based on the above results, the accuracy of OCT is not consistently better than CART. This could be due to the limited complexity of the tree structure. One way to improve the performance of OCT could be to increase the depth of the tree, such as $D = 3$, $D = 4$, $D = 5$. However as explained by Bertsimas and Dunn [1], while OCT of higher depth ($D = 3$, $D = 4$) still significantly outperform CART, the difference between the average accuracies of OCT and CART decreases in comparison to OCT of depth 2; and such situation could be caused by the lack of computational ability of MIO solver to solve the OCT model at higher depth.

## 4.2 Adjust Complexity Parameter Alpha

An important parameter in the OCT is the complexity parameter alpha. This parameter balances the additional complexity of adding splits at the node against the improvement in prediction accuracy that it provides. If alpha is too large, the model builds a tree that applies fewer splits and will be overly simple, which leads to underfitting; if alpha is too small, the model could build a maximum tree that results in overfitting.

$\alpha$ is a continuous variable. In order to find the right alpha, Bertsimas and Dunn [1] show in the paper that we do not randomly try multiple $\alpha$ values to find the one that gives optimal testing set performance. Instead, we minimize the number of MIO problems that need to be solved by

focusing on the "critical" values of $\alpha$ that would lead to different solutions. This was achieved by reformulating the penalty term as follows:

$$\begin{aligned} \min \quad & \frac{1}{\hat{L}} \sum_{t \in \mathcal{T}_L} L_t \\ \text{s.t.} \quad & \sum_{t \in \mathcal{T}_B} d_t \leq C \end{aligned}$$

where $C$ is a constant corresponding to $\alpha$.

First, we set the maximum tree depth as $D_{max}$. For each depth D ranging from 1 to $D_{max}$, since the sum of splits is integer valued, we have $C \in [1, ..., 2^{D_{max}} - 1]$. We solve the OCT problem for each $C$ to generate a pool of solutions and eliminate any non-optimal solutions to the original problem (22) for any $\alpha$ value. We evaluate all remaining candidate solutions on a validation set, then determine the interval of $C$ values for which the corresponding trees perform the best, and select the midpoint of this interval as the final tuned $\alpha$.

### 4.3 Optimal Classification Trees with Hyperplanes (OCT-H)

In OCT, each branch node of the decision tree performs a univariate split, meaning it uses only a single feature to make a decision. One way to improve the OCT is to extend our its problem formulation to yield a problem for determining the optimal tree with multivariate splits, known as the optimal classification trees with hyperplanes (OCT-H). The idea is to relax the binary variable $a_t \in \{0, 1\}^p$. Instead, we let $a_t \in [-1, 1]^p$ at each beach node $t$, enabling each node to split on a linear combination of features. In addition, OCT-H introduces the new binary variables $s_{jt} \in \{0, 1\}$ to track whether feature $j$ is used in split $t$ for all features and branch nodes. This allows the model to penalize the number of features used rather than the number of splits. Accordingly, the objective function for OCT-H is modified to:

**Objective:**

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

**Subject to:**

$$
\begin{aligned}
&L_t \geq N_t - N_{kt} - n(1 - c_{kt}), \quad k = 1, \ldots, K, \ \forall t \in \mathcal{T}_L \\
&L_t \leq N_t - N_{kt} + nc_{kt}, \quad k = 1, \ldots, K, \ \forall t \in \mathcal{T}_L \\
&L_t \geq 0, \quad \forall t \in \mathcal{T}_L \\
&N_{kt} = \tfrac{1}{2} \textstyle\sum_{i=1}^{n} (1 + Y_{ik}) z_{it}, \quad k = 1, \ldots, K, \ \forall t \in \mathcal{T}_L \\
&N_t = \textstyle\sum_{i=1}^{n} z_{it}, \quad \forall t \in \mathcal{T}_L \\
&\textstyle\sum_{k=1}^{K} c_{kt} = l_t, \quad \forall t \in \mathcal{T}_L \\
&a_m^\top x_i \geq b_m - 2(1 - z_{it}), \quad i = 1, \ldots, n, \ \forall t \in \mathcal{T}_L, \ \forall m \in A_R(t) \\
&a_m^\top (x_i + \epsilon) \leq b_m + (2 + \mu)(1 - z_{it}), \quad i = 1, \ldots, n, \ \forall t \in \mathcal{T}_L, \ \forall m \in A_L(t) \\
&\textstyle\sum_{t \in \mathcal{T}_L} z_{it} = 1, \quad i = 1, \ldots, n \\
&z_{it} \leq l_t, \quad \forall t \in \mathcal{T}_L \\
&\textstyle\sum_{i=1}^{n} z_{it} \geq N_{\min} l_t, \quad \forall t \in \mathcal{T}_L \\
&\textstyle\sum_{j=1}^{p} \hat{a}_{tj} \leq d_t, \forall t \in \mathcal{T}_B \\
&\hat{a}_{tj} \geq a_{tj}, j = 1, \ldots, p, \ \forall t \in \mathcal{T}_B \\
&\hat{a}_{tj} \geq -a_{tj}, j = 1, \ldots, p, \ \forall t \in \mathcal{T}_B \\
&-s_{jt} \leq a_{tj} \leq s_{jt}, j = 1, \ldots, p, \ \forall t \in \mathcal{T}_B \\
&s_{jt} \leq d_t, j = 1, \ldots, p, \ \forall t \in \mathcal{T}_B \\
&\textstyle\sum_{j=1}^{p} s_{jt} \geq d_t, \forall t \in \mathcal{T}_B \\
&-d_t \leq b_t \leq d_t, \quad \forall t \in \mathcal{T}_B \\
&d_t \leq d_{p(t)}, \quad \forall t \in \mathcal{T}_B \setminus \{1\} \\
&z_{it}, \ l_t \in \{0, 1\}, \quad i = 1, \ldots, n, \ \forall t \in \mathcal{T}_L \\
&s_{jt}, \ d_t \in \{0, 1\}, \quad j = 1, \ldots, p, \ \forall t \in \mathcal{T}_B
\end{aligned}
$$

For the OCT-H problem, the warm start generated by a univariate model such as CART becomes less effective. To generate a good warm start for OCT-H, Bertsimas and Dunn introduce the greedy top-down induction algorithm. We start at the root, solve the OCT-H problem of depth 1, and collect the solutions to the variables $a_1$ and $b_1$, and determine the points that follow the left split and right split. For the left child, we repeat the above procedure to collect $a_2$ and $b_2$ and determine the points that are split to node 4 and 5, respectively. We repeat this procedure for all branch nodes, then the resulting matrix $[a_1, ..., a_{|TB|}]^\top$ serves as the warmstart for a, and $[b_1, ..., b_{|TB|}]$ serves as the warmstart for b, where $|TB|$ is the total number of branch nodes. The procedure for tuning $\alpha$ remains the same as OCT.

## 4.4 Strong Optimal Classification

We learned base on the numerical experiments and the paper, solving the OCT model is time consuming and its performance is not consistently better than CART, especially for trees of greater depth due to the increasing difficulty of the model. In the paper *Strong Optimal Classification Trees*, Aghaei, Gómez, and Vayanos [5] focus on these technical difficulties and argue that the problem formulation of OCT does not fully leverage the solving power of MIO to its full extent. By proposing an alternative formulation to the OCT model – the flow-based MIO formulation and the corresponding Bender's decomposition of the formulation for learning optimal classification trees with binary features – Aghaei et al. demonstrate that such formulation "has a stronger LO relaxation", and the resulting Strong OCT model can be solved 29 times faster and achieve 8% improvement on out-of-sample accuracy in numerical experiments in comparison to OCT.

# 5 Conclusion

In this report, we introduced the fundamentals of tree structures for binary trees and decision trees. After that, we show how to formulate the OCT model following the methodology from the paper: *Optimal Classification Tree* by Bertsimas and Dunn [1]. Using the Gurobi mixed-integer optimization solver, we constructed the OCT class in Python and demonstrated that OCT is possible to be solved and obtain globally optimal solutions. We evaluated the performance of OCT model on real-world classification datasets from the UCI machine learning repository and compare against the performance of CART, XGBoost and Random Forest.

Our results indicate that OCT achieves an average testing set accuracy improvement around 7% over CART. Even though XGBoost outperformed OCT in some cases, its improvement is capped by 4%, considering that XGBoost is not interpretable and has overfitting problems. Overall, the result provides comprehensive evidence that OCT has interpretability with each split clearly traceable to specific features and its thresholds, and it leads to significant improvement over heuristic methods.

Finally, we mentioned the tuning methods for the parameters that impact the performance of OCT; and we also briefly highlight OCT with hyperplanes (OCT-H) proposed in the paper *Optimal Classification Tree*, Bertsimas and Dunn [1] and the extended *Strong Optimal Classification Trees* by Aghaei, Gómez, and Vayanos [5], which aim to fundamentally enhance the efficiency and predicting ability of OCT by alternating its problem formulation.

# 6 Appendix: Tree Structure Graphs

In this section we show some visual examples of the trees constructed by the CART, OCT and OCT-H models.

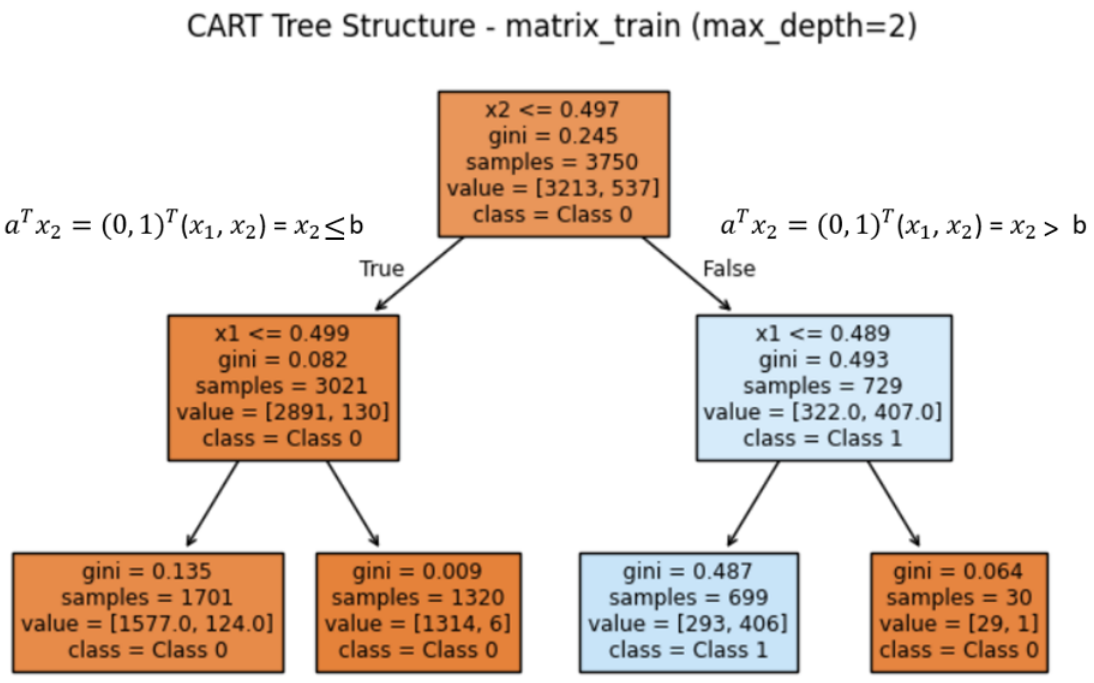


**Figure 2.** CART of depth 2 on the dataset Matrix Train.

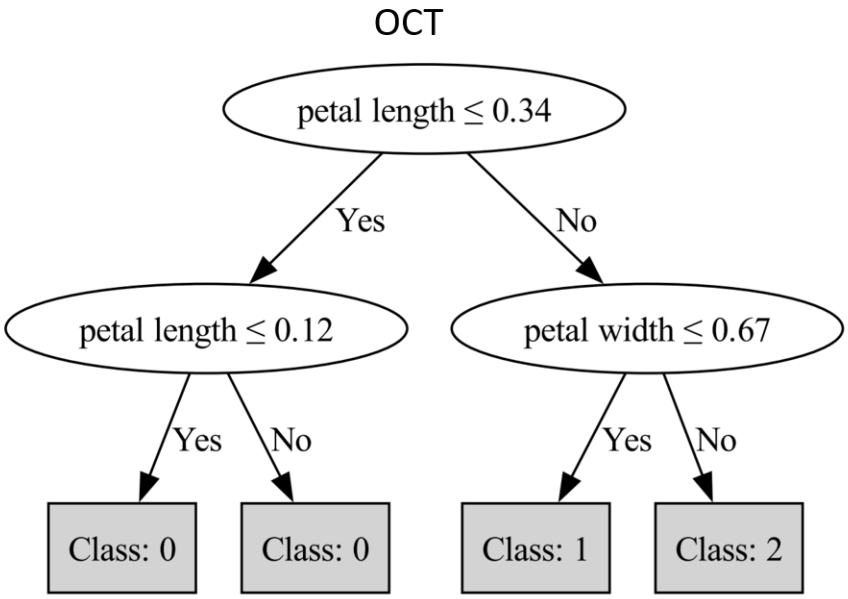


**Figure 3.** OCT of depth 2 on the dataset Iris.

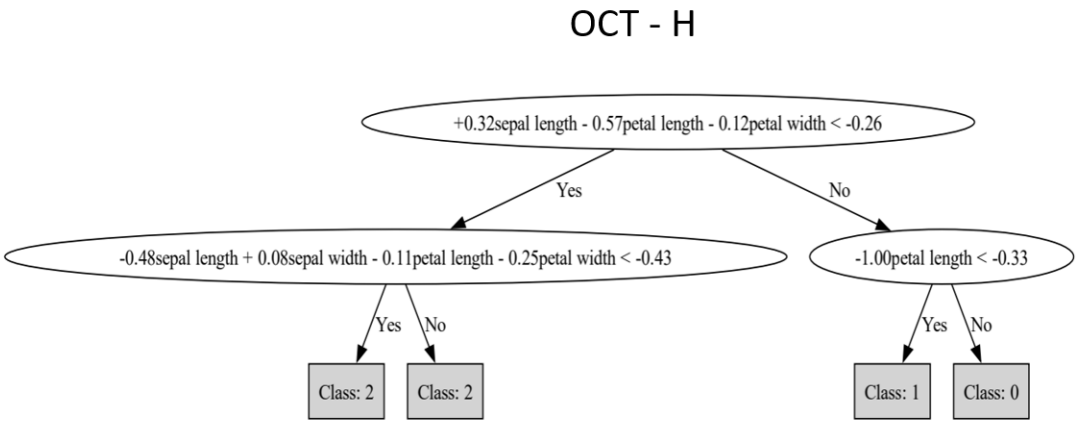


**Figure 4.** OCT-H of depth 2 on the dataset Iris.

# References

[1] D. Bertsimas and J. Dunn, "Optimal classification trees," *Machine Learning*, vol. 106, no. 7, pp. 1039–1082, 2017.

[2] Gurobi Optimization Inc., *Gurobi Optimizer Reference Manual*, 2015, http://www.gurobi.com.

[3] IBM ILOG CPLEX, *IBM ILOG CPLEX V12.1 User's Manual*, 2014, https://www-01.ibm.com/software/commerce/optimization/cplex-optimizer/.

[4] M. Lichman, "Uci machine learning repository," 2013, http://archive.ics.uci.edu/ml.

[5] S. Aghaei, A. Gómez, and P. Vayanos, "Strong optimal classification trees," 2021, https://optimization-online.org/wp-content/uploads/2021/01/StrongOCT-1.pdf.

[6] T. P. (2022, October) Overfitting and underfitting in machine learning. SuperAnnotate. Accessed: 2025-05-02. [Online]. Available: https://www.superannotate.com/blog/overfitting-and-underfitting-in-machine-learning

[7] X. Lin and Z. Ouyang, "Optimal classification trees," https://github.com/ZhuoqiaoO/OptimalClassificationTree, 2025.