ROLLING LOOK-AHEAD APPROACHES FOR OPTIMAL CLASSIFICATION TREES

A Thesis

by

Zeynel Batuhan Organ

Submitted to the
Graduate School of Sciences and Engineering
In Partial Fulfillment of the Requirements for
the Degree of

Master's Degree

in the
Department of Industrial Engineering

Özyeğin University
June 2022

ROLLING LOOK-AHEAD APPROACHES FOR OPTIMAL CLASSIFICATION TREES

Approved by:

Assistant Professor Enis Kayış,
Advisor
Department of Industrial Engineering
Özyeğin University

Assistant Professor Dilek Günneç Danış
Department of Industrial Engineering
Özyeğin University

Assistant Professor Tagi Hanalioğlu,
Co-Advisor
Department of Industrial Engineering
Bilkent University

Assistant Professor Erinç Albey
Department of Industrial Engineering
Özyeğin University

Date Approved: 30 June 2022

Professor Mehmet Güray Güler
Department of Industrial Engineering
Yıldız Technical University

To my beloved family and friends...

ABSTRACT

Classification trees have gained tremendous attention in machine learning applications due to their inherently interpretable nature. Current state-of-the-art formulations for learning optimal binary classification trees suffer from scalability for larger depths or larger instances. Moreover, they mostly fail to prove optimality after long run times and fit perfectly to the training data while minimizing misclassification error which is likely fail to generalize to the test data. We present a simple but powerful new formulation which we call rolling look-ahead learning approach. By dropping tractability variables which are dependent on instance size, we present a novel two-depth optimal binary classification tree formulation with the objective to minimize gini impurity or misclassification error. The approach can be thought of as a middle ground between myopic and global optimization methods. For larger depths, we developed a hybrid approach which learns by looking ahead 2-steps rolling horizon. It is much faster than the fastest known global optimization methods which can solve an instance with around 50K rows & 135 features in less than 4 minutes, for depth 8. Also, in majority of cases, the proposed approach outperforms global optimization methods & CART in terms of win count tested for 7 depths, 10 Fold and 19 benchmark datasets, and increase in out-of-sample accuracy up to 16.8% and 11.9% with respect to global optimization methods and CART, respectively.

ÖZETÇE

Son zamanlardaki ikili sınıflandırma karar ağacı öğrenim en iyileme formülasyonları, büyük derinlikler veya büyük verisetleri ölçeklenebilirlikle ilgili problem yaratmaktadır. Uzun saatler süren çalışma sürelerine rağmen, en iyi değeri kanıtlamakta sorun yaşamaktadır. Ayrıca, eğitim sürecinde amaç fonksiyonu olarak yanlış sınıflandırma metriği sonucu, test setinde yanlı bir sonuç çıkabilmekte, iyi bir sınıflandırma yapmakta sorun yaşamaktadır. Bu çalışmada, önceki çalışmaların veri boyutuna bağlı olan ve takip eden değişkenlerinden kurtulup, CART gibi algoritmalardan da esinlenerek, 2 derinlikli ağaçlar için yenilikçi bir formülasyon sunuyoruz. Algoritma açgözlü yaklaşımlar ve global optimizasyon yöntemleri arasında kalan spektrumdan faydalanıyor. Daha büyük derinlikli ağaçlar için ise, 2 seviye ileri görerek öğrenen hibrit bir algoritma geliştirdik. Bu algoritma, yaklaşık 50 bin satırlı, 135 özellikli bir veride 8 derinlikli bir ağacı 4 dakikadan daha kısa sürede çözebilir ve gelişime açıktır. Ayrıca, mevcutta bulunan en iyi global optimizasyon ve CART yaklaşımını skor sayısıyla geçebilirken, global optimizasyon modellerine göre test setinde %16.8'e kadar bir artış, CART'a göre ise %11.9'a kadar bir artış gözlemlenmiştir. Gözlemler 19 veri seti, 7 farklı derinlik ve 10 katlamalı veri grubunda test edilmiştir.

ACKNOWLEDGEMENTS

I'd like to show my gratitude and appreciation to the people who helped me complete this thesis.

Firstly, I would like to express my most sincere gratitude to Asst. Prof. Enis Kayış, who was also my undergraduate advisor, and Asst. Prof. Taghi Khaniyev for all their support, kindness and enthusiasm throughout my research journey. They always pushed me to accomplish more, are always supportive, appreciative. This thesis would not have been finished if it hadn't been for their guidance. I'm grateful that I'd a chance to work with them and deeply inspired by their dynamism, vision, genuineness.

I wish to acknowledge the continuous support of my parents. I consider myself blessed to have them at my side. I'd like to thank my grandma in particular, who is always enthusiastic with me throughout the masters. Even though she didn't understand, she was always there for me during my graduate studies, and it was because of her unwavering faith in me that I didn't give up. And of course, I'm sending my best wishes to my father and grandpa, and I wish they could be here to witness it.

I want to mention great friends Volkan, Alperen, Feyza, Melis, Çağatay who accompanied me during my master's degree, they were always patient while I had to work busy. Special thanks to Eren for being a great colleague, for everything he'd done. İrem and Çelen, thanks for the continuous support & motivation throughout thesis.

Finally, I am extremely grateful to my partner Ecem for her support, patience, trust and cheeriness during the most crucial times of my graduate study. She always motivated and believed for me to do more.

TABLE OF CONTENTS

LIST OF TABLES

LIST OF FIGURES

CHAPTER I

INTRODUCTION

Decision trees are a widely used methodology which is used in regression and classification problems. Decision Trees are relatively old concepts, they maintained their popularity and are used in many industries due to their practicality & interpretability. Recent years have seen a proliferation of new methodologies for classification tasks, some of them are very fast, sophisticated, complex and very well researched, also called black-box algorithms, such as Gradient Boosting Trees. Gradient Boosting Trees are a collection of week learners, which are decision trees, to build a strong model. They mostly perform well but lack of interpretability.

Classification Trees are also the main building blocks of some of the popular algorithms, like Random Forests [1]. Recent trends, especially in the healthcare sector or finance where people are also interested in the reasoning of the decision instead of the pure outcome, showed that the interpretability is as valuable as than the outcome quality. Some want to invest their money based on interpretable & provable decisions rather than an inexplicable decisions. Therefore, decision trees always maintain their importance with their simplicity, practicality and interpretability. The main idea behind the decision trees is to build a tree in which a data point can follow a path up until the end of the tree, called a "leaf", resulting in a class assignment to itself. When all the training data points end up in leaves, then "training" of the tree is finalized to give a prediction for unseen data. When constructing the tree, in every step, there has to be decided on which attribute (feature) to branch up on. This is a crossroads-like step that separates data points into different paths.

As mentioned, there are several approaches to train decision trees. One of the most acknowledged ones is classification and regression trees, so-called *CART* [2],

C4.5 [3] and *ID3* [4]. These approaches use a similar idea when generalizing the trends in the input (training) data. Starting from the top of the tree, it looks for the next best option, locally optimizing the pre-defined objective function. In other words, the tree with a root node decides to split from a feature and separate data into two leaves by minimizing the impurity, then for both of the leaf nodes, repeat the split process and decide again which feature to separate the data into two leaves, again locally optimizing. This process continues recursively as the tree grows, or as the depth of the tree increases. This approach is interpreted as a top-down algorithm which is starting from the top root node and grows up until the leaf nodes. The objective of these popular algorithms can differ, CART typically uses Gini Impurity, ID3 mainly uses entropy, and since C4.5 is the extension of ID3, it uses normalized information entropy as the objective.

There are several shortcomings of these popular decision tree methods. The primary issue is that they are "myopic" algorithms. This means that, as mentioned above, at every level, they look for the best local decision which may likely cause it to move away from the global optimal. This 1-step look ahead lacks the consideration of the whole picture and can cause to get stuck at the local optima and under-perform in the unseen data. Moreover, these approaches are mainly heuristic, which does not guarantee building globally optimal trees.

Breiman et al.,[2] first sparked the idea of optimal decision trees. Since then, the idea continued to find support, and a new field of research emerged. According to the research, instead of looking only one step ahead each time, one can also look for the all partitions, therefore build a tree with full knowledge. Since building an optimal binary classification tree is an NP-Hard problem [5], there arise many sophisticated approaches for modelling & solving optimal classification trees, so-called OCT. For example, Bertsimas (2017) [6] proposed a novel MIP formulation for learning optimal classification trees, Verver (2019) [7] introduced an approach for binary classification trees. These global optimization models try to fix one of the primary flaws of the local optimize models, such as CART, by learning the

whole tree and trying to achieve optimality. However, there's a trade of between these approaches: CART is quick and easy but doesn't optimize globally and may end up in local-optima and under-perform in train data. On the other hand, global optimization models try to optimize globally but it is slow and lacks scalability.

Although these global optimized models emerged against the shortcomings of myopic models, they make visible the problem of look-ahead. It is well known that there is a pathology problem in tree learning. [8] Global optimized models are, due to their nature, heavily focused on maximizing the accuracy (or minimizing the misclassification error), likely to overfit to training data, however, they fail on the test data. These methods are trying to fit the train data perfectly while minimizing misclassification error so that sometimes they cross the line and are likely to fail in test data.

Although an OCT can learn a tree well, there is a scalability problem which cannot be overlooked. As the instance size increases and tree depth increases, global optimize models are stuck in proving optimality and resulting in higher gaps, or no feasible solution in a reasonable time. These shortcomings question the applicability of the approaches and usually, they require a heuristic for to large, real-life datasets. Most of the researched models introduce binary variables which depend on the row size and feature size. Introducing these variables also increase the complexity of a model exponentially as the instance size starts to increase.

Another major difference of CART is that it employees what one may call a "soft-clustering" approach by using the gini impurity as the objective function. However, OCTs are doing hard-clustering. In detail, hard clustering tries to split the data based on minimum misclassification error (or maximum accuracy) by dividing the largest group of class. On the other hand, soft-clustering can keep multi-labelled classes in a group while still minimizing the gini impurity. Let's say we have 10 data points consisting of 7 Class A and 3 Class B. Hard clustering can only split with 7 to 3 and cannot proceed further, but soft clustering can split data into two group where first group may contains 2 for both Class A and B,

and second group may contains 5 for A, 1 for B, and ends up with lower impurity. Nevertheless, the comparison between these two approaches is not direct. Training accuracy performance is expected to be higher for OCT. Since they optimize for training accuracy, this gives an advantage to OCT but it disappears in the test data. This brings to mind the question to combine soft-clustering with the classical global optimization models, but it requires a non-linear integer optimization and adds more complexity to current approaches.

In this section, we briefly introduce a new formulation for optimal classification trees, which is independent of the number of observations and moves the computational burden to prior calculations. Another bottleneck of the OCTs was heavily introduced binary tracking variables, which are dropped in our formulation. It is looking ahead for 2-step while constructing a two-depth tree, whereas myopic solutions like CART look ahead for 1-step away. We still try to decide which feature to branch upon at every level while minimizing the gini impurity index.

We use 2-steps look ahead modelling and build a rolling look ahead learning for larger depths. We start from the root node and train a two-depth tree, then we fix the first depth of this tree, and go back to the nodes id 2 and 3, which is the parent nodes of the two-depth tree. Then train a two-depth tree for each of them, separately. This enables to learn from the mistakes and give a chance for model to fix these mistakes in the upcoming subtrees. The approach is neither global optimization nor a myopic solution, and it tries to explore the spectrum between these two methodologies, specifically based on 2-steps look-ahead similar to Lastman & Roizman [9]. Iteratively, it constructs a two-depth tree from the parent nodes of the previous trees and optimizes at every level.

This new formulation can enable both hard & soft clustering without any extra work. Parameters can vary from accuracy to gini, which can easily use the objective with the same formulation. Therefore, we can fairly compare the approach with global optimization methods, which uses misclassification in the objective and compare with myopic algorithms like CART, using gini impurity.

This new formulation is scalable with the instance size, have no limitations on growing larger depths. Optimization model does not affected by the instance size, only the pre-calculation part depends on the instance size and feature size of the data because of exploring all the possible combinations via enumeration for a two-depth tree.

CHAPTER II

PREVIOUS WORK

2.1 Literature

The use of mathematical optimization methods for optimal decision trees begins with Bennett (1992) [10]. They proposed a linear programming formulation for determining an optimal linear split for binary class datasets. Günlük et al. [11] proposes an integer programming which aims to find optimal and interpretable classification trees, which is specialized for categorical data and binary class datasets. Verwer (2017) [12] introduced a flexible formulation which is efficient up to 5 depth with small instances. Bennett (1996) [13] proposed a formulation for solving binary classification trees concurrently, represent the existing tree with the disjunctive linear inequalities. Rudin et al. [14], [15], [16] leveraged on optimal decision lists. Hu et al. (2019) [16] proposed an algorithm for optimizing decision tree binary variables. The algorithm introduces analytical limits and leverages the CORELS' library and its computational reuse paradigm in order to limit the search space by designing a bit-vector library. Blanquero et al. (2018) [17] & [18] proposes a continuous optimization formulation and a sparsity concept in randomized trees, which random decisions are made at the tree's internal nodes. The approach can be considered as a randomized optimal version of CART.

Related work with our methodology uses MIP techniques to train globally optimal trees.(Bertsimas & Dunn, 2017)[6]; (Verwer & Zhang, 2019) [7]; (Aghaei et al., 2020) [19]. Dunn et al. (2017) [6] proposed a novel approach using mixed-integer programming (MIP) to learn optimal classification trees by creating variables & constraints for every data point in the training set size. This yields a scalability problem and increases the problem size dramatically. Verwer & Zhang (2019) [7]

proposed a new binary formulation, in order to make formulation largely independent from the training set and using a binary search procedure with big-M for splitting thresholds. *BinOCT* is much faster dropping the tracking variables and outperforms *OCT*. However, because *BinOCT* is limited to creating full binary trees of a particular depth rather than optimally sparse trees, it occasionally adds extra leaves to complete a tree at a given depth. It works for only balanced binary decision trees, it is likely to open an unnecessary leaf just for a complete tree. Aghaei et al. [19] also introduced a strong max-flow based MIP encoding, so-called *FlowOCT*. It is a $big - M$ free formulation and since $big - M$s likely to result in loose LP-relaxations, *FlowOCT* has the ability to produce strong branch-and-bound performance. They further accelerated the solution by introducing new decision variables and apply benders decomposition algorithm, so-called *BendersOCT*.

Decision tree learning exposes a problem called learning pathology, which generates overfitted trees. Myopic solutions like CART can be labelled as no look-ahead or 1-step look-ahead, where in every depth it looks for the best decision in the next step. Global optimize methods can be labelled as infinite-horizon look-ahead since it is learning with full depth. Last & Roizman (2013) [9] indicates that a deeper look may cause overfitting despite increasing the in-sample accuracy. They've found out that most of the time, the highest prediction accuracy is obtained with a two-depth look-ahead where the myopic algorithms are labelled as 1-depth. They conclude that growing the local search slightly with single-level subtrees will be enough.

Dynamic programming is also used frequently for solving optimal classification trees. Garofalakis et al. (2003) [20] proposed a time and memory efficient dynamic programming for optimal subtrees for a predefined size-constrained or accuracy constrained decision trees. However, the approach creates suboptimality while creating larger depths. Nijssen and Fromont (2007, 2010) [21], [22] introduced an algorithm called DL8 for finding decision trees that solves an optimization

problem exactly under a wide range of constraints. This idea of the algorithm can be summarized as the relation between constraints and the itemsets of decision trees. They experimented that DL8 better in out-of-sample accuracy than C4.5. Aglin et al. (2020) [23] proposed a dynamic programming approach using branch-and-bound methodology and leveraged use of caching. The algorithm caches the subtrees during the iterations and saves it for later use. This is useful for speedups since both of the left and right subtrees can be optimized independently. The method outperforms BinOCT and DL8 most of the instances except small ones and fastest to prove optimality and available in Python. [24] Lin et al. (2020) [25] used also dynamic programming search space to address the scalability and poor performance issues for imbalanced data. Dynamic programming search is defined as "exposing a high degree of computational reuse when modelling continuous features". They introduced a framework called GOSDT. It is mostly promising for small and medium featured-size datasets and can scalable well, since it decompose problem into smaller problems and iteratively solve. Demirovic et al. (2020) [26] progress the DL8.5 and introduce constraints to limit the number of total nodes in the tree and improve the efficiency leading a scalable algorithm. The presented method MurTree is relatively limited to small depths (depth 4) to be interpretable and scalable, and overperforms Random Forest for the half of the tested datasets during the experiments with smaller instances.

There are many use cases for different objective functions used in research, including the average testing length Hyafil (1976) [5]. Most of them are about training accuracy, but there can be a combination of training accuracy and model interpretability (Dunn [6] (2017)) & Hu (2019) [16]. Hu (2019) [16] formulates an objective function with a mixture of misclassification error & a sparsity penalty on the number of leaves created by the tree. Lin et al. (2020) [25] also extends the previous research by introducing several linear & non-convex objective functions, such as F-Score, area under the ROC convex hull (AUC_{ch}), area under the ROC convex hull ($pAUC_{ch}$) along with the accuracy, weighted accuracy and

balanced accuracy. Aghaei (2019) [27] used the combination of training accuracy and fairness. They mentioned that these methods are started to be used in social decision-making. It can prejudice and treat individuals inappropriately and unfairly, for a minority group or category whom untreated well and discriminated against in the history. Bennett (1996) [13] introduced several non-linear objective functions to minimize errors. Demirovic (2020) [28] also introduced several non-linear metrics as the objective for optimal classification trees. They proposed a bi-objective methodology which optimize the two rival objective function at the same time for a given set of constraints. To the best of our knowledge, we firstly introduce a gini impurity index in the objective.

For real-world application, Dunn [6] builds interpretable solutions at scale using Optimal Classification Trees as products, creating ML solutions under the name InterpretableAI. [29]

2.2 Contributions

Our contributions involve a simple but powerful new formulation which considers Gini Impurity in the objective function, where we've found that as the depth increases, Gini Impurity is much better than misclassification error. We've proposed a two-depth fixed binary optimal classification tree which can work in multi or binary classes, small or large instances & feature sized or balanced-imbalanced datasets. It explores the spectrum between myopic solutions & globally optimized solutions to avoid pathology and achieve better learning by looking 2 steps ahead. The formulation also drops the main bottleneck of the recent MIP formulations which suffer from big-M constraints and instance size-dependent binary variables and yields a very fast solution even in larger datasets. i.e. around 50000 rows with 135 features, it proves optimality in less than 2 minutes in comparison to BendersOCT which takes more than 30 minutes, reporting 100% optimality gap. It differs from Verwer (2019) [7] that it is not limited to complete binary trees and does not open any unnecessary leaves. The main computational burden of

our model is the non-linearity of gini impurity, which is moved to prior calculation and goes into the model as a parameter.

Since our novel approach constructs for two-depth optimal classification trees, there is a need for larger depths. As the instance size increases, it is likely to require larger depths to learn and capture the trend in the data. That's why we present a rolling look-ahead learning mechanism for larger depths. It solves two-depth fixed optimal decision trees and grows the tree by rolling look-ahead and iteratively fixing the errors on previous depths. Since the two-depth model is fast in training, the algorithm is scalable for growing larger depths or larger instances.

Finally, we reported extensive experiments for 19 benchmark datasets varying in different types. Many of the aforementioned formulations suffer from large instances, which they are not tested more than 10000 instance sizes, or greater than depth 5. We tested our algorithm and compared it with BinOCT, FlowOCT, BendersOCT and CART with all the instances up until depth 8 and 50K rows.

CHAPTER III

MODELLING APPROACH

As we discussed in previous sections, recent studies on optimal classification trees mostly focus on the global optimization of the decision tree. There are a couple of issues with optimizing the decision tree as a whole. One of them is scalability. Optimizing the tree globally depends on the instance size and the feature size. It requires larger depths as the instance size increases to learn and generalize the trend. Another issue is overfitting. Even if an OCT is able to grow a larger depth tree, overfitting pathology emerges. The OCT fits perfectly to the whole training set by optimizing it, typically achieving a high in-sample accuracy value. But it is possible to fail on the test set. By acknowledging these limitations, this study is introducing a new binary classification tree model of which the building blocks are two-depth optimal classification trees.

Our point of view is the same as the aforementioned OCT approaches. We try to decide which feature to branch on in every depth level and at the same time avoid tracking every instance point throughout the decision tree. This means that we are dropping the main bottleneck of the OCT approaches, binary decision variables, for every data point if it exists in that leaf node or not. Yet another approach to the pathology of the OCTs is that instead of using hard clustering - as minimizing the misclassification amount in each node, we try to use soft clustering - minimizing the likelihood of incorrect classification of a data point, which is gini impurity.

3.1 Two-depth models

The key step is to build a perfect two-depth binary tree. A perfect binary tree means that every level contains the maximum number of leaf nodes. Fig. 1 represents the perfect binary two-depth tree, where node 1 is the root node, with

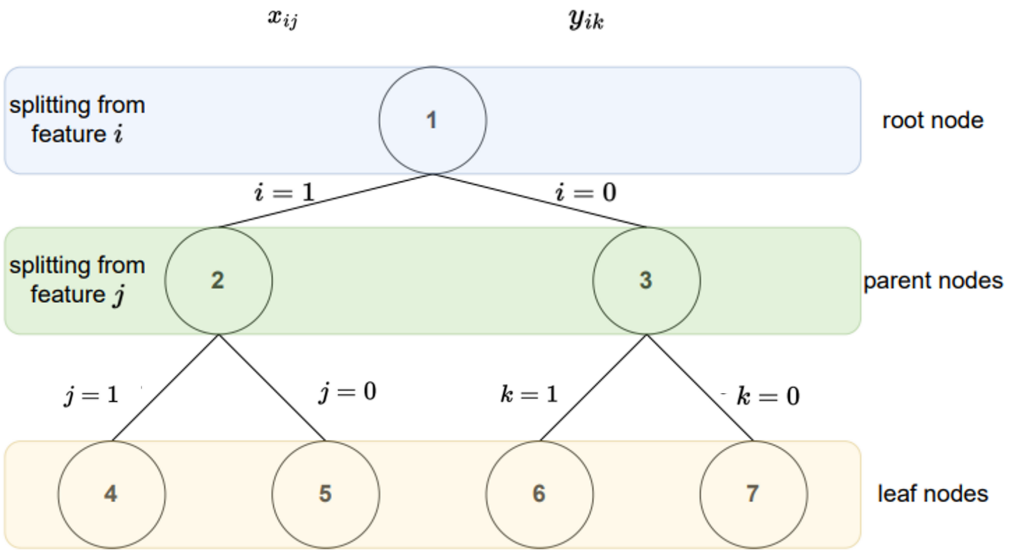

Figure 1: Diagram of two-depth Tree

depth 0. When splitting feature i at the root node, data points follow the path to the left, if feature $i = 1$. (or right if feature $i = 0$). Nodes 2 & 3 are the parent nodes, with depth 1. Finally, leaf nodes, or child nodes, represent the class assignment with node indices $4, 5, 6, 7$ resulting in depth 2. However, the formulation does not force to create a complete tree, if there is no datapoints in a leaf node, it does not force to open it.

The reason why we use two-step models as our building blocks is similar to the logic of the CART algorithm, which is widely used in the practice. CART is a myopic algorithm which tries to identify the best split which results in the minimum loss at a given node, by looking only 1-step ahead. Inspired by the CART algorithm, the proposed model can optimize & learn by looking 2-steps ahead - which is still greedy, but not as much as myopic as CART.

3.2 Problem Formulation

3.2.1 Objective Function

To the best of our knowledge, all the recent optimal classification tree studies are trying to minimize the misclassification errors (or to maximize correctly classified instances). This is one way to build decision trees since the success metric of the

final tree is based on how many of the data points are classified correctly. Using accuracy as the objective function can be described as hard clustering. Although it is beneficial for the training accuracy since optimization is using the training data, there is a chance to overfit. CART-like heuristic approach are known to suffer less from the overfitting problem. One reason for this is the objective function they use that may be interpret as soft-clustering objectives (e.g., gini impurity). Therefore, we also considered using gini impurity as an alternative objective function in our approach.

Gini Impurity is a measure of the likelihood of randomly chosen data being misclassified if it is given a random class label using the class distribution of the dataset [2]. Instead of hard clustering, gini impurity can be described as soft clustering.

Consider a training dataset for a classification task with a set of class $\mathcal{K}$. The probability of a data point labelled as class k in a leaf node denoted as p_k. So the Gini Impurity can be defined as:

$$Gini(dataset) = 1 - \sum_{k \in \mathcal{K}} p_k^2 \tag{1}$$

The Gini Impurity metric is between 0 & $1 - \frac{1}{k}$. The minimum impurity, is 0, meaning that the leaf consists of all the same class samples. The maximum impurity is 0.5 for a case of $\mathcal{K} = 2$, meaning that the leaf consists of an equal number of different classes. This metric is calculated for every leaf node and then weighted. As can be seen from the equation 2, every gini impurity value is multiplied by n_i, which n_i is the number of datapoints in that leaf node i, n is the total training data size.

$$Weighted\ Gini = (1 - \sum_{k \in \mathcal{K}} p_k^2) \cdot \frac{n_i}{n} \tag{2}$$

The CART algorithm supports gini impurity as the objective function. Using the gini impurity metric as an objective function in OCTs can be computationally expensive. Due to the non-linear nature of gini function (1), adding gini as an

objective function transforms OCTs into a nonlinear MIQP model. That's why we designed our approach which involves enumerating gini values of different split combinations prior to running the optimization model. See equation 3 to see how gini impurity is used in the objective function.

As mentioned above, the general metric used to cluster data points in a leaf node is accuracy. We also try to use the accuracy metric in our objective function, in order to see & discuss the effects of:

- what happens if the objective used while building a tree top-down is accuracy vs gini impurity
- to compare with recent studies such as BinOCT, BendersOCT etc.
- the effect of looking 2-steps away (our model) vs looking 1-step away (CART).

3.2.2 Model

Parameters of the model are all calculated prior to model building. Due to complexity of the aforementioned optimal classification trees, as the instance size or the feature size increases model complexity increases exponentially. To overcome this problem, the burden of calculating the misclassification (or gini impurity) coefficients is taken out of the optimization model and handled via enumeration prior to the modelling phase. Gini impurity measure is calculated for all feature i, j combinations, where $i, j \in \mathcal{F}$ for every leaf node of fixed two-depth tree.

- $c_{i,j}^{(1)}$: The weighted gini impurity or the misclassification error of the leaf obtained as a result of the splits $\{x_i = 1, x_j = 1\}$, representing the leaf node id 4 in Figure 1.
- $c_{i,j}^{(2)}$: The weighted gini impurity or the misclassification error of the leaf obtained as a result of the splits $\{x_i = 1, x_j = 0\}$, representing the leaf node id 5 in Figure 1.

- $c_{i,k}^{(3)}$: The weighted gini impurity or the misclassification error of the leaf obtained as a result of the splits $\{y_i = 0, y_k = 1\}$, representing the leaf node id 6 in Figure 1.

- $c_{i,k}^{(4)}$: The weighted gini impurity or the misclassification error of the leaf obtained as a result of the splits $\{y_i = 0, y_k = 0\}$, representing the leaf node id 7 in Figure 1.

Our main objective (3) is to minimize the total gini impurity of the tree, similar to the OCTs, which minimizes the total misclassification error of the tree. x_{ij} & y_{ik} (7) are the binary decision variables, where x_{ij} is 1 if i^{th} feature is the split variable on first level and j^{th} feature is the split variable on second level for *left child*. Similarly, y_{ik} is 1 if i^{th} feature is the split variable on first level and k^{th} feature is the split variable on second level for *right child*. What follow is complete optimization model we propose for the optimal two-depth binary optimal classification trees.

$$\min \sum_{i \in \mathcal{F}} \sum_{j \in \mathcal{F}} \left(c_{i,j}^{(1)} + c_{ij}^{(2)} \right) \cdot x_{ij} + \sum_{i \in \mathcal{F}} \sum_{k \in \mathcal{F}} \left(c_{ik}^{(3)} + c_{ik}^{(4)} \right) \cdot y_{ik} \tag{3}$$

s.t.

$$\sum_{i \in \mathcal{F}} \sum_{j \in \mathcal{F}} x_{ij} = 1 \tag{4}$$

$$\sum_{i \in \mathcal{F}} \sum_{k \in \mathcal{F}} y_{ik} = 1 \tag{5}$$

$$\sum_{j \in \mathcal{F}} x_{ij} = \sum_{k \in \mathcal{F}} y_{ik}, \ \forall i \in \mathcal{F} \tag{6}$$

$$x_{ij}, y_{ik} \in \{0, 1\} \tag{7}$$

Constraint 4 ensures that exactly one pair of features (i, j) are selected for the leaves originating from the left child of the root node. Similarly, Constraint 5

ensures that exactly one pair of features (i, j) are selected for the leaves originating from the right child of the root node. Constraint 6 ensures that the split variables selected for the first level are the same for both the left leaves (4, 5) and the right leaves (6, 7). If a feature $x_i = 1$, then relative datapoints follows through the left side of the part, on the other hand, datapoints which does not satisfy the expression $i = 1$, follows through the right side of the part, as satisfying $i = 0$.

CHAPTER IV

ALGORITHMIC APPROACH

In modern world of analytics, small instances are not quite realistic and do not show the real capabilities of the algorithms or models. When the instance size increases, decision trees tend to grow deeper in order to capture the whole picture. Since our proposed optimization model is limited to two-depths, we need to address the issue of growing larger depth trees or dealing with larger instances. To the best of our knowledge and experiments, in-sample or out-of-sample accuracy values are most likely to increase for a large instance as the tree grows. To overcome this limitation, we proposed a rolling-learning approach which is originated from the CART algorithm.

Rolling horizon approaches can be used in many aspects in predictive analytics. Sethi (1991) [38] used this methodology in forecasting, considering both finite and infinite horizon cases. Wang et al. (2015) [39] tested the approach in predicting truck capacities by creating virtual pricing mechanism. Our proposed algorithm can be defined as a mechanism neither achieving global optimum like OCT nor myopic like CART. It falls between these two methodologies by re-optimizing in every level by looking ahead for 2 depths ahead. This way, we can grow a larger tree by generating lots of small sub-trees and optimising them iteratively.

First of all, we generate and optimize a two-depth model using the training set. Then the algorithm finds the leaf nodes where misclassification occurs. These misclassified nodes are marked. After finding the leaf nodes to grow further, i.e. the marked nodes, we go back to their parent nodes. These parent nodes will be the new root node of sub-trees. e.g., if there is a misclassification in leaf node 4, left side of the tree, then its parent node is 2. So the next process will be building and optimizing a two-depth tree from the parent node 2, which means

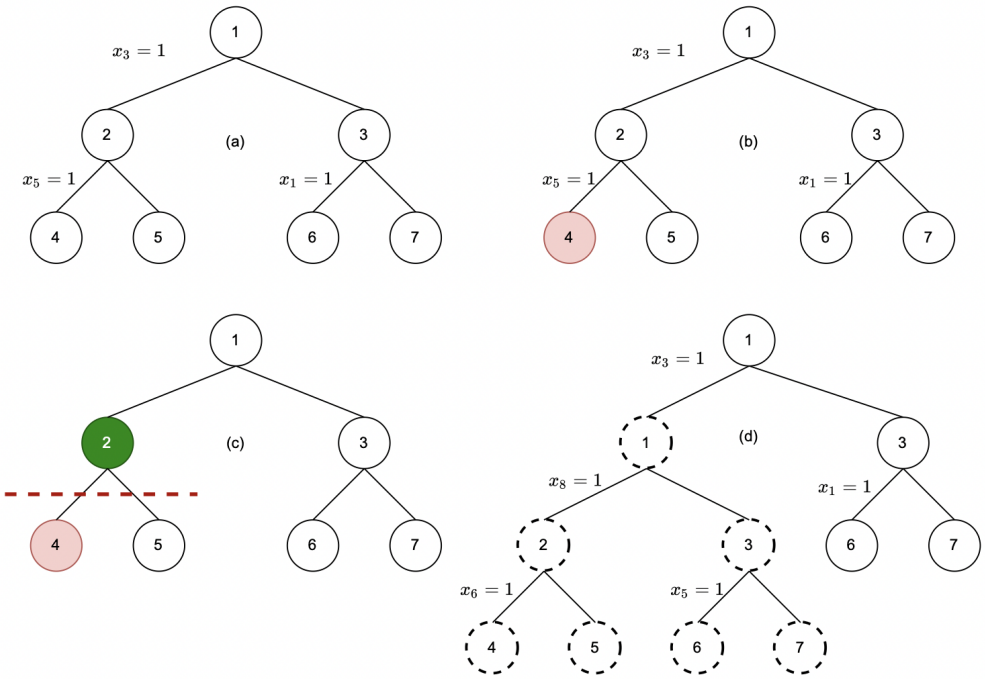

Figure 2: Example of Rolling Look-Ahead Learning

discarding the presolved model's leaf node results. We do the same for all the nodes marked in the previous step. This process goes on until the predefined final depth of the tree is reached or until no more misclassified leaf node is left. Figure 2 illustrates this "rolling" approach. First, we solve a two-depth model as can be seen in (a). The splitting variables are 3rd for the first level, 5th for the left leaf nodes and 1st for the right leaf nodes. Then, in (b), leaf node 4 is labelled as a misclassified node, which means there are more than one class labels for data points in the leaf node. In (c), we terminated the sub-tree where misclassification occurs, cutting the leaf nodes from parent node 2, and starting a new two-depth model from that respective parent node 2, as represented in (d). Note that the splitting variables can change while generating new two-depth tree since this is a new tree with sub-data. The new splitting variable is 8th for the left leaf node of two-depth tree instead of 5th. After this process, the final depth of the tree is 3, and the algorithm repeats the processes $(b), (c), \&(d)$ until the final depth of the the predefined tree reaches to target depth. Furthermore, the algorithm does not necessarily have to go through all the depths. If there is no misclassification in any

of the current leaf nodes depth before the target is reached, then the algorithm terminates itself.

Algorithm 1 Rolling Look-Ahead Learning

```
(1.) Optimize two-depth Model
(2.) Find leaf nodes with misclassification occurs
while depth ≤ target_depth ∨ misclassification exists do
    Find parent nodes of misclassified leaf's, i.e., marked_nodes
    for marked_i ← marked_nodes do
        Optimize two-depth Tree growing from marked_i
        Find misclassified leaf nodes
    end for
end while
```

The major advantage of this model is to learn the misclassified leaf node patterns and try to fix them in next iterations. In Figure 2 - *c*, when the algorithm starts from parent node 2 with a new two-depth model, it does have a chance to classify more accurately in the next depth. Since it re-optimizes from the parent node of previous two-depth model, it has a chance to classify again. It also discards the leaf nodes even if the node is purely classified and consider these data points one more time. Since the CART algorithm looks for the best option for the one-step ahead in every level, our rolling look-ahead learning algorithm leverages by looking 2-steps ahead and optimising. By positioning between global optimal and myopic solution, the two-depth rolling look-ahead leverages the advantages of the CART algorithm and thus avoiding the pathology of the recent optimal classification tree approaches. As discussed above, as the learning increases, probability of over-fitting is also increases. But the rolling look-ahead learning decreases the chances of overfitting.

CART is a very fast and competitive algorithm which has computational complexity $O(\mathcal{P}\cdot\mathcal{N}\cdot\log\mathcal{N})$ where $\mathcal{P}$ is the number of predictors and $\mathcal{N}$ is the number of samples [30]. The dominant complexity part is the sorting of each input in order to find the splitting variable. However, we can discuss the complexity of our methodology in two parts. Firstly, when we consider two-depth model, proposed model is not affected by the instance size, $\mathcal{I}$, main bottleneck of the model is

the precalculation of the gini impurites enumerated for each pair of split variables. Since we calculate gini index for every feature pair, the complexity is $O(\mathcal{F}^2)$ which is still a polynomial complexity. Secondly, the optimization part of the algorithm is so fast even for large instances that it is almost negligible.

One of our flaws is that as the depth of the tree grows, our algorithm must solve a large number of optimization models for each step or misclassified leaf node. While this may seem like a restraint, it actually turns into an advantage rather than a disadvantage. As the depth increases, the data points in the leaf nodes become fewer. This setting enables faster precalculation and run time, which removes the time complexity barrier from our algorithm and allows growing larger depth trees in a reasonable time. In addition, our major improvement in optimal classification trees is that we do not track which data point is located in which leaf node. That tracking methodology increases the complexity of the model exponentially, as the depth increases. Verwer (2019) [7] also dropped the binary variables indicating the leaf-assignment for each instance and formulated the model, $BinOCT$, independent of the instance size. However, it is only for complete binary trees and it opens unnecessary leaf nodes for completeness. Recall that our formulation does not force to open unnecessary leaf nodes. Another positive point is that, in every depth, the tree can grow, by generating new two-depth model and solving, independently. Growing tree towards left and right does not affect the other side. This actually allows trees to run in parallel which yields 2x faster runs. However, the performance improvement via parallel computing & improvements on precalculation was beyond our scope. Further comments about time complexity will be discussed in the 5.5 Results section. We will mention $RLA-G$ for the rolling look-ahead gini model, $RLA-A$ for the rolling look-ahead accuracy model, and RLA for the general method, in the upcoming sections.

CHAPTER V

COMPUTATIONAL EXPERIMENTS

This section tries to explain experiments done throughout the thesis. In section 5.1, all the instances used in the experiments will be discussed. In section 5.2, we will explain how the instances processed before entering into algorithms. Section 5.3 discusses the benchmark models used in these experiments. Section 5.4 describes the comparison of models in terms of different performance metrics. All the discussions and final results reported in detail in Section 5.5.

5.1 Instances

For the experiments, we used publicly available benchmark datasets from UCL repository [37]. Table 2 summarizes the data sets we've used during our study. Furthermore, we've generated different data group labels such as Instance Size Group, Feature Size Group, Over-fitting Group, Class Size Group & Minority Fraction Group.

Table 1: Data Group Labels

Data Group	**Group Label**	**Explanation**
Instance Size Group	Small	instance size is less than 1000
Instance Size Group	Medium	instance size is between 1000 & 10000
Instance Size Group	Large	instance size is greater than 10000
Feature Size Group	Small	feature size is less than 30
Feature Size Group	Medium	feature size is between 30 & 100
Feature Size Group	Large	feature size is greater than 100
Class Size Group	Binary	class size is 2
Class Size Group	Multiple	class size is greater than 2
Overfitting Group	Not Exist	difference between avg. train & test accuracy is less than 10% for CART
Overfitting Group	Exist	difference between avg. train & test accuracy is greater than 10% for CART
Minority Fraction Group	Balanced	minority class fraction is greater than 30%
Minority Fraction Group	Imbalanced	minority class fraction is between 10% & 30%.
Minority Fraction Group	Highly Imbalanced	minority class fraction is less than 10%

The goal is to collect as many datasets as possible from various groups in order

to test the methodology's capabilities. This allows us to see how our algorithm performs when the instance size is high, what happens when there are more than 100 features, and whether there is a difference between binary and multi-class classification. We obtained a total of 19 datasets with various labels.

Table 2: Instance Details

Instance Name	**Class Size**	**Instance Size**	**Feature Size**	**Processed Feature Size**
adult	2	48842	14	135
tae	2	151	5	32
spambase	2	4601	57	136
tic-tac-toe	2	958	9	27
banknote-authentication	2	1372	4	40
wine	3	178	13	130
haberman	2	306	3	25
wdbc	2	569	30	300
diabetes	2	768	8	72
seismic-bumps	2	2584	18	72
monks-1	2	556	6	15
monks-2	2	601	6	15
monks-3	2	554	6	15
titanic	2	891	8	178
kr-vs-kp	2	3196	36	38
car-evaluation	4	1728	6	21
balance-scale	3	625	4	20
nursery	5	12960	8	26
agaricus-lepiota	2	8124	22	112

5.2 Data Preprocessing

Since our methodology only enables binary data classification, the preprocessing is essential. We attempted to find as many categorical data as possible because they are already binarized or simple to binarize. Because categorical data is far more difficult to obtain and retrieve, there is a limited amount of publicly available data. Then, to expand our data library, we binarized continuous or mixed datasets. Before binarizing, the numerical/continuous columns are first categorized depending on their quantile. As quantile divides observations in a sample in the same way, or based on probability, the first step was to quantile a column into ten groups; if the sample size was insufficient, the next step was to quantile in decreasing order until the sample was successfully divided. After that one-hot

encoding [31] is applied for all the categorical columns & for both type of datasets. For the numerical columns less than 7 different entries are not clustered based on quantile, they treated as a discrete class. One-hot method is encoding categorical features as one-hot numeric array. This actually creates a disadvantage since it increases the dimension of a feature by the number of different groups in that feature. Final feature sizes for the datasets can be found in Table 2, as Processed Feature Size.

Data is splitted 90% for the training, 10% for the test. We used Stratified K-Fold Cross Validation [32] instead of sampling, as K being 10. The reason of using K-Fold Cross Validation is that every datapoint is used in test data once. Especially if a dataset is imbalanced, by random sampling, train dataset may not contain any samples from minority class or may contain highly imbalanced. This yields a bias and would likely to fail in testing. This method tries to preserve percentage of samples for every class between folds and creates a more fair environment for testing.

5.3 Compared Methods

There are total of 7 models used in our experiments. These models can group into 3. Traditional acknowledged models, CART, global optimization models used as benchmark - BinOCT, FlowOCT, BendersOCT - and introduced models - RLA-A, RLA-G.

CART is a general expression for *Classification and Regression Trees*. We used the classification decision trees on the basis of Gini Impurity. The Python implementation, *DecisionTreeClassifier* [33], from sklearn library is used for the CART results. All the parameters used as default, except the *max_depth* & *ccp_alpha*. *max_depth* parameter is set from 2 to 8 and *ccp_alpha* is set to 0 since we do not consider pruning in our modelling.

The second group of compared methods are BinOCT, BendersOCT & FlowOCT. BinOCT first introduced by Verwer & Zhang [7] to avoid the complexity problem

caused by introducing tons of binary decision variables in previous OCT implementations. However, it works for only balanced binary decision trees, so that it likely to open an unnecessary leaf just for a complete tree and it will not be a fair comparison. Thus, in this research, the classical OCT formulation which is firstly introduced by Bertsimas (2017) [6] and adapted to binary data by Aghaei et al. (2021) [27] in their Strong optimal classification trees study is used. This model is used as BinOCT & implemented it in Python Programming using Gurobi solver. Another model is the FlowOCT from Aghaei et al. (2021) [34] which they proposed with BendersOCT in their Strong optimal classification trees study. The BendersOCT using benders decomposition to solve the model. We used the open source model which can be found in Github as StrongTree [35]. A total of 1 hour has been established as a time restriction for the entire algorithm for all BinOCT, BendersOCT & FlowOCT, 30 minutes for model run time, and 30 minutes if model deployment is required, as building models can take time.

Final group is the introduced models in this paper. First one is LA-1 / Look-Ahead (1-Step) model. This is similar to a modified CART method that we developed to see what would happen if the CART algorithm used accuracy at each level instead of the Gini impurity index. Instead of splitting based on the Gini index, it splits based on maximum accuracy (or minimum misclassification error). This also helps on examining the affect of look-ahead steps on the methodology. The rest is RLA-G and RLA-A. RLA-G / Rolling Look-Ahead (2-Step) with Gini is recently introduced basis 2-step rolling algorithm with gini impurity index in the objective. On the other hand, RLA-A / Rolling Look-Ahead (2-Step) with Accuracy is another basis 2-step rolling algorithm with accuracy metric in the objective.

5.4 Comparison Metrics

We try to discuss the results from different perspectives. The main comparison metric is accuracy. Classical in-sample accuracy metric was used for training. It

is the percent of the sum of correctly classified data points. This is for how the model optimizes or learned the data in general. Out-of-sample accuracy metric used for the test. This is for how the model behaves with new data. Another flaw of the OCT is the pathology due to its nature. That's why it is likely to overfit test data as the learning process increases. We will also discuss the overfitting margins.

During the tests, trees with a depth of 2 to 8 are created. Larger trees are sometimes required for better learning with larger instances. Classical OCT implementations have issues as the depth increases even for smaller instances. As a result, we've increased the depth and are testing the scalability of all the models we've reported.

Some of the challenges with recent models include that they are unable to close the gap despite producing good outcomes. However, we can only present the optimality gap for depth 2 in our two-depth model. As a result, we'll compare optimality gaps in great detail 2.

Another point of the OCT implementations' aforementioned shortcomings is that when the depth or instance size grows, the model cannot be solved in a reasonable amount of time, and the CPU time grows rapidly. It's worthy to mention that all of the models listed have a maximum run time of one hour. If the previous depth took more than 1 hour or if a feasible solution could not be reported in that time, the following depths will be skipped.

5.5 Results

All the models are implemented in python programming language with a commercialized solver Gurobi 9.1. See Gurobi (2020) [36]. Google Cloud Compute (GCP) is used for the running environment with a virtual machine of 12 CPU x 64 GB RAM. There is a total of 19 datasets, with 10 folds for each dataset, 7 depths (2 to 8) resulting in 1330 data points for all 3 metrics, training accuracy, test accuracy & time. For ease of understanding and tracking, all the models will

be mentioned with their abbreviations and color palette assigned to each model. Further details are discussed in section 5.3. CART (Red) is the classical myopic algorithm. Rolling 2-Steps Look-Ahead Learning with Gini in the objective as RLA-G (Green), same method with Accuracy Metric in the objective is RLA-A (Blue). LA-1 (Purple) is 1-Step is with Accuracy in the objective (same as CART except the objective function). BendersOCT (Pink), FlowOCT, BinOCT are the global optimized methods.

5.5.1 1-Step vs 2-Step Look-Ahead Comparison

In this section, we briefly try to discuss both of the introduced methods, RLA-G & RLA-A among themselves and with the CART algorithm in terms of look-ahead & objective function. RLA-G model uses a two-depth look-ahead policy & CART uses 1-depth look-ahead with Gini Impurity in the objective function.

Below, you can find the comparison charts for the models. The scores are presented in terms of win count, which means that if a model wins for a specific depth, fold & data, it gets a point. If there is a tie between two models, then it counts as a tie in the charts. The data labels as percentages show the percent of the total win count for RLA-G model.

As can be seen from Figure 3, RLA-G beats CART model in every depth, both in terms of training & test accuracy. The difference is more clear in the training accuracy, as expected. However, for depth 2, both of the models are mostly in a tie. This figure clearly shows that looking ahead for 2-steps is better than 1-step when both of the models' objective is Gini Impurity.

When we look at the comparison of LA-1 & RLA-A model, Figure 4, the same trend from the previous figure can be seen again. Looking ahead for 2 steps is more dominant in terms of training accuracy, which is an expected result. However, this trend is mostly preserved in the test accuracy, too. This figure also proves that looking ahead for 2-steps is far better than 1-step when both of the models' objective is accuracy. To summarize, whether the objective is Gini Impurity or accuracy, looking ahead for 2-steps is improving the results than 1-step, both for

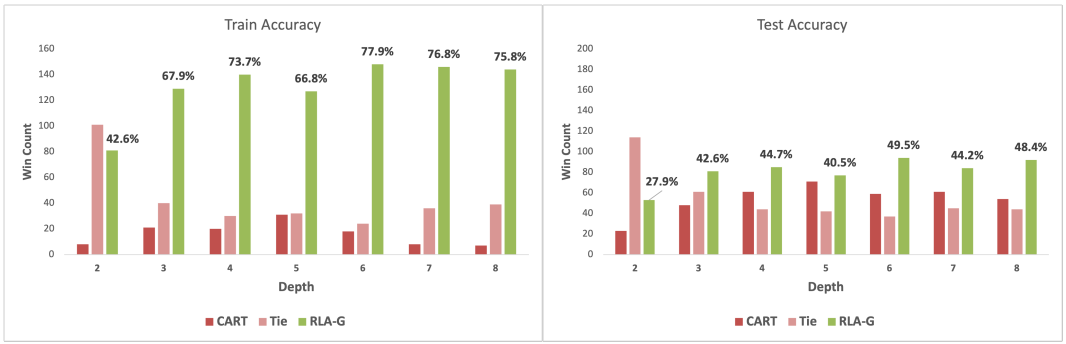

Figure 3: In-Sample & Out-of-Sample Accuracy comparison of CART vs RLA-G Model in terms of Win Count in Depth Breakdown

training and test accuracy values.

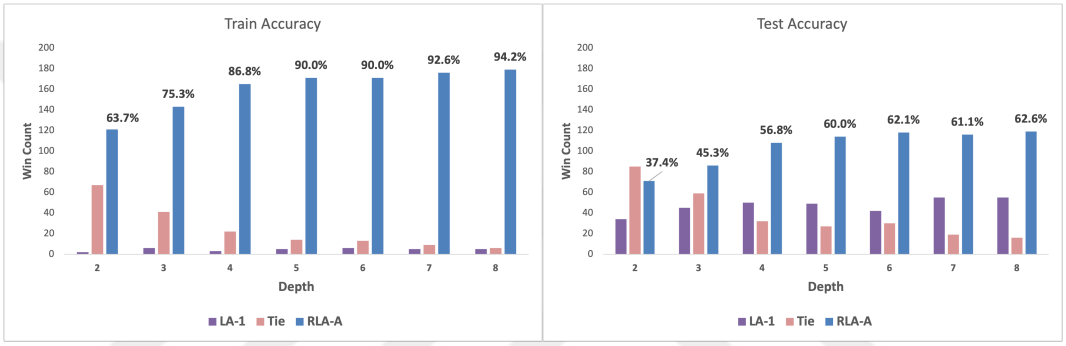

Figure 4: In-Sample & Out-of-Sample Accuracy comparison of LA-1 vs RLA-A Model in terms of Win Count in Depth Breakdown

5.5.2 Objective Function Comparison: Accuracy vs Gini Impurity

For further analysis of the objective function, both of the introduced models, RLA-A & RLA-G, are compared because they are equal in every aspect except the objective function. The aim is to discuss whether the accuracy metric or Gini Impurity is outdoing one another. Figure 5 shows the comparison on Depth Level. RLA-A Model tends to decrease as the depth increases in terms of both training & test accuracy. Conversely, RLA-G model tends to increase as the depth increases for training accuracy, but for test accuracy, there is no clear trend. This is a valuable outcome that Gini Index seems more stable than the accuracy metric as the objective.

It is known that one of the limitations is the potential overfitting of optimal classification trees. Figure 6 shows that as the depth increases, overfitting seems

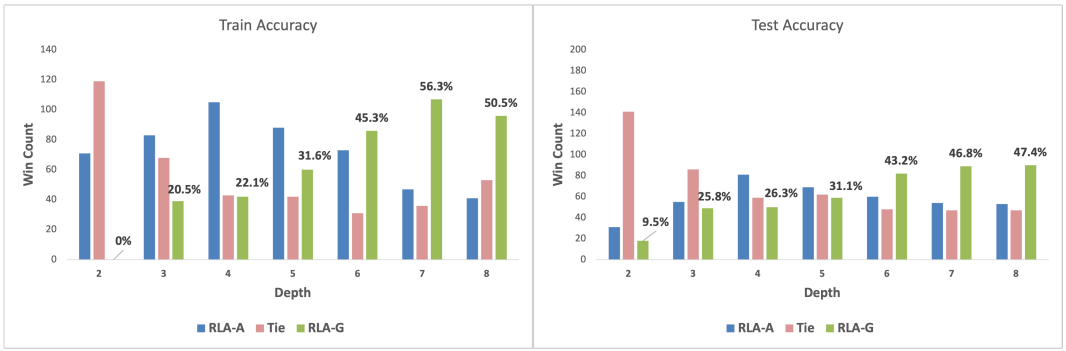

Figure 5: In-Sample & Out-of-Sample Accuracy comparison of RLA-A vs RLA-G Model in terms of Win Count in Depth Breakdown

to be likely for all models and the variation also increases. However, the margins for all are close to each other. Moreover, optimizing using the accuracy metric in the objective function, RLA-A model, is likely to overfit more than using Gini Impurity metric in the objective function, RLA-G Model. This proves that Gini Impurity metric is better than the accuracy metric when dealing with overfitting. Knowing the limitations of OCTs, it is argued that it makes more sense to use the Gini Impurity index as the objective function. Also, notice that since RLA-G and RLA-A fitting to train data by optimizing, it can reach better in training accuracy and overfitting seems more than CART.

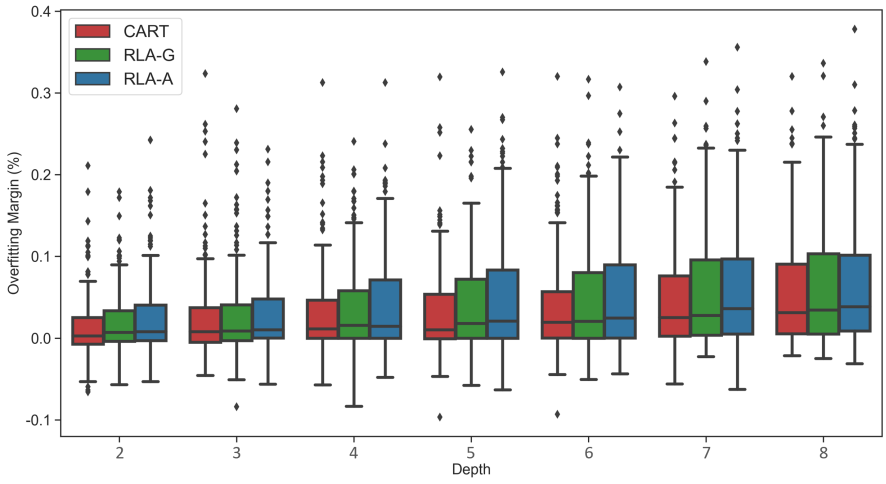

Figure 6: Comparison of Over-fitting Margins for CART, RLA-G & RLA-A Model (%) in Depth Breakdown

Nevertheless, RLA-A model is consistently better when the depth is small.

This brought up a topic. According to our observations, Accuracy Metric is likely to be worse if there is small data to split up in a branch. This usually occurs as the depth increases, as the data to be split in parent nodes decreases. So, we expect a decrease in performance for RLA-A, if the depth increases, even though the instance size is large. But on the contrary, Gini Impurity can split even though the data is small. Figure 7 also proves that, even for a large dataset, RLA-G beats RLA-A as the depth increases.

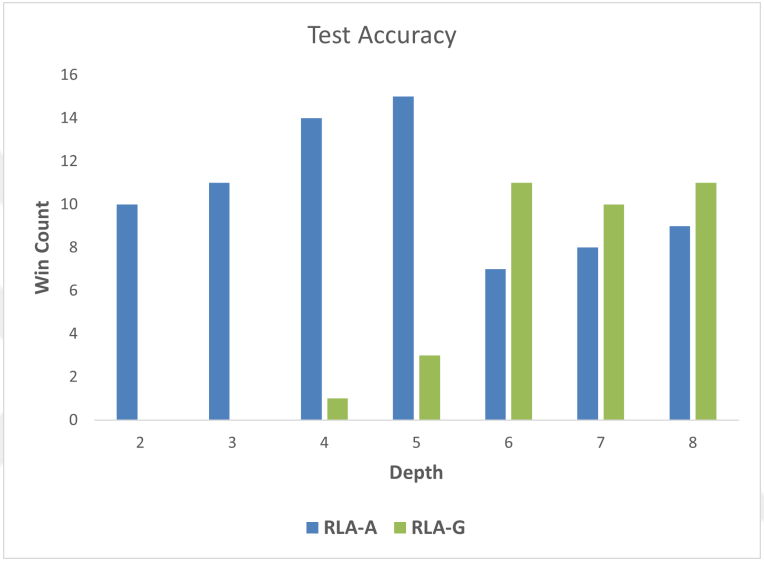

Figure 7: In-Sample & Out-of-Sample Accuracy comparison of RLA-A vs RLA-G Model in terms of Win Count in Depth Breakdown and Large Datasets

In the end, we decided to combine these two methodologies & create a hybrid model, which uses RLA-A model for less than depth 5 and RLA-G model for greater than or equal to 5 depth, called as Hybrid Model in the upcoming parts. Also, we ran a CART model for Test Accuracy metric, winning model for the target class, consisting of every dataset, fold and depth combinations and data labels as features. As can be seen in Figure 8, the main split is from depth 5, where the left part of the tree mainly labelled with RLA-A, right part of the tree mainly labelled with RLA-G.

In comparing CART vs Hybrid model as in Figure 9, Hybrid model dominates in both in-sample and out-of-sample accuracy values for every depth. For in-sample accuracy, Hybrid model is outperforming CART nearly 79% of the runs in

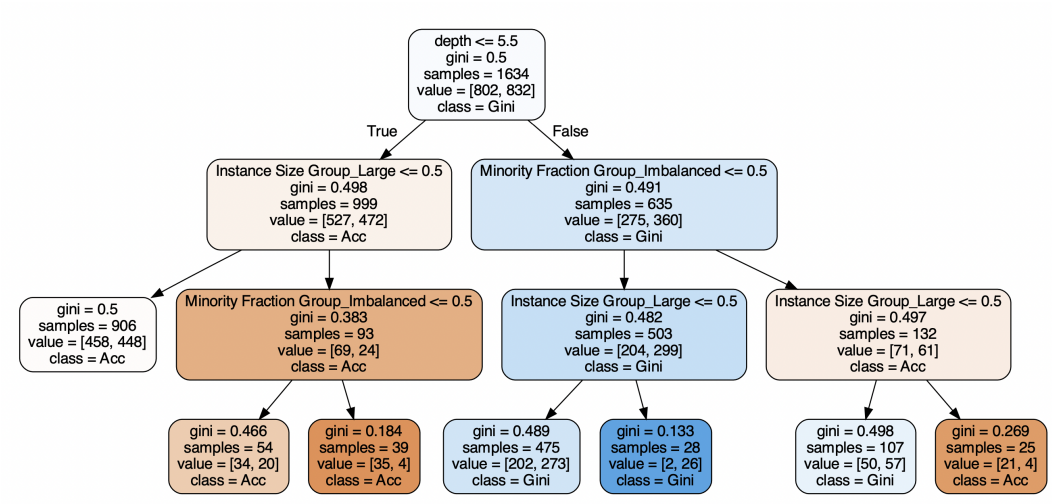

Figure 8: A CART to classify which is better, RLA-G or RLA-A, using the pre-created data group labels

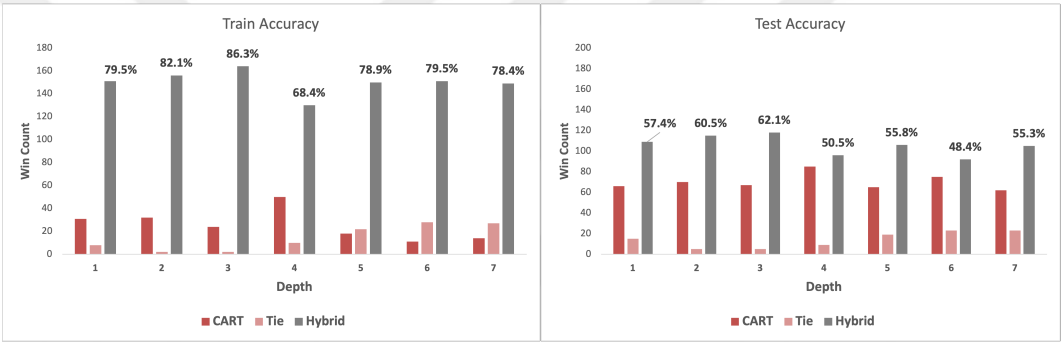

Figure 9: In-Sample & Out-of-Sample Accuracy comparison of CART vs Hybrid Model in terms of Win Count in Depth Breakdown

terms of win count. This trend is still visible in out-of-sample accuracy, 55% of the runs on average, Hybrid model outperforms CART.

5.5.3 Performance Comparison of Methods for Different Instance Features

This dominance is also observed in every group in different labels, such as Instance Size Group, Feature Size Group, Minority Fraction Group, Over-fitting Group and Class Size Group. Details can be found in section 5.1.

Figure 10 shows that for every Instance Size Group, Hybrid Model beats the CART model in terms of in-sample accuracy. This statement is valid for the out-of-sample accuracy except for the *small* instances where CART performed as well as the Hybrid model. This indicates that as the instance size increases, the Hybrid

model fixes the misclassification in every step by looking at the rolling horizon, where the CART model tends to get worse. For the large instance size group, Hybrid model is 4.4, for the medium, it is 3.1 times better than CART. However, for the small, it is 0.9 times worse than CART.

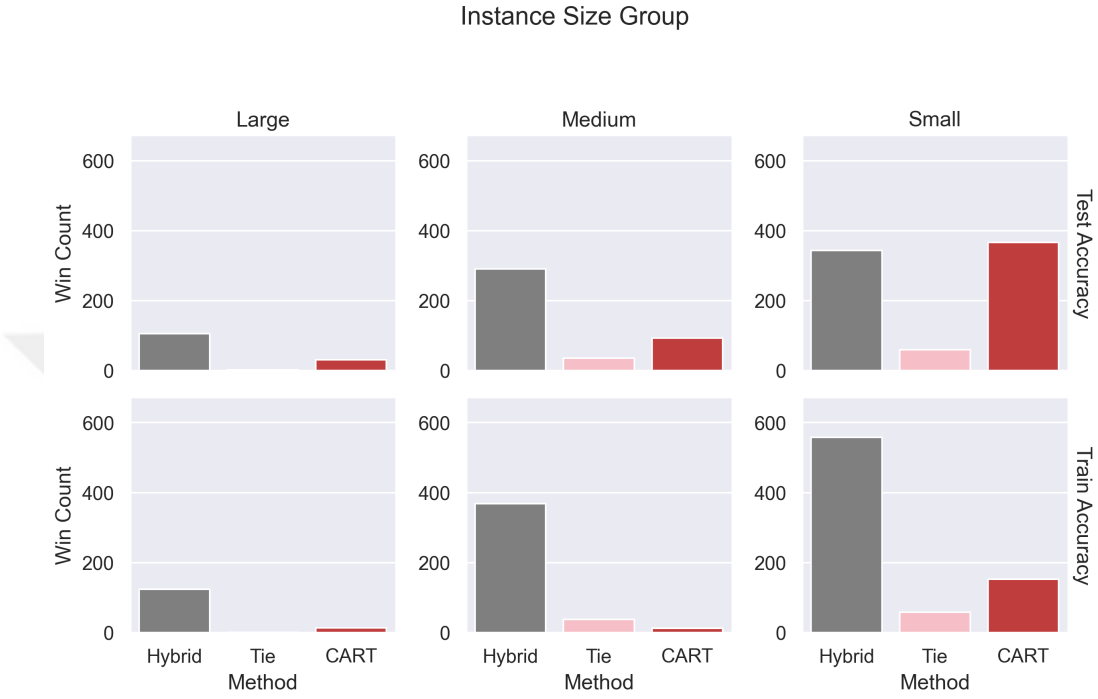

Figure 10: In-Sample & Out-of-Sample Accuracy comparison of CART vs Hybrid Model in terms of Win Count in Instance Size Group Breakdown

For small feature sizes, CART performs better with respect to other groups. In-sample accuracy is far better as expected, and for large & medium feature size groups, Hybrid model is better than CART. For the large feature size group, Hybrid model is 1.5, for the medium, it is 2.4 & for the small, it is 1.2 times better than CART, as can be seen in the Figure 11.

Figure 11: In-Sample & Out-of-Sample Accuracy comparison of CART vs Hybrid Model in terms of Win Count in Feature Size Group Breakdown

Whether the class size is binary or multiple, Hybrid model is the winning model. Although data distribution for binary and multiple cases are not the same, Hybrid model is 1.5 times better CART in binary, and 1.8 times better in multiple for out-of-sample accuracy. Figure 12 indicates that hybrid model performs much better in multiple class sized datasets than it performs in binary class-sized datasets.

Figure 12: In-Sample & Out-of-Sample Accuracy comparison of CART vs Hybrid Model in terms of Win Count in Class Size Group Breakdown

Similar to the class size group, whether overfitting exists or not, Hybrid model is the winning model. Hybrid model is 1.3 times better than CART if overfitting is known to exist, and 1.7 times better if overfitting is known to not exist for out-of-sample accuracy. This trend is also valid for in-sample accuracy, as described in Figure 13.

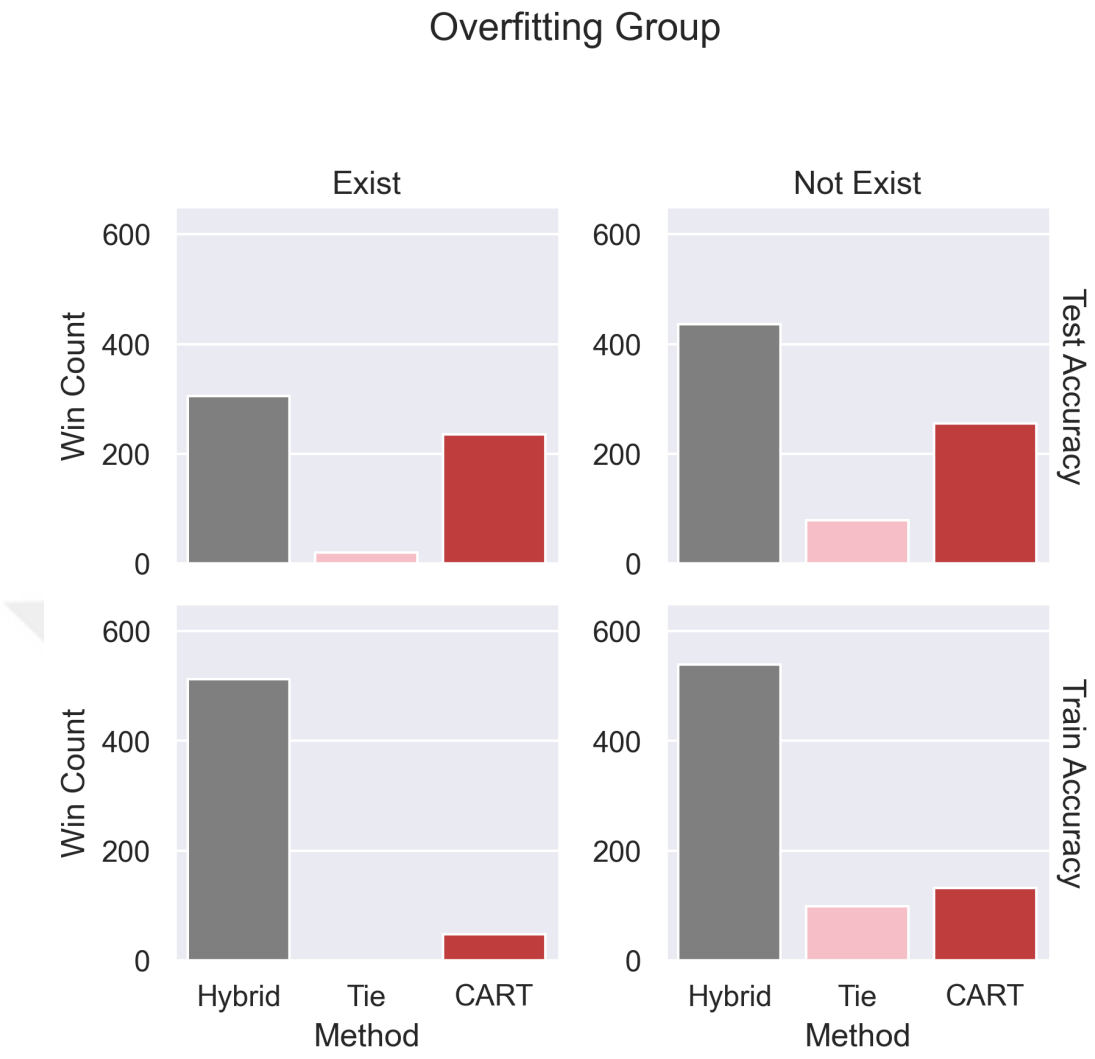

Figure 13: In-Sample & Out-of-Sample Accuracy comparison of CART vs Hybrid Model in terms of Win Count in Over-fitting Group Breakdown

Whether data is balanced, imbalanced or highly imbalanced, the Hybrid model is the beats CART in terms of win count for both in-sample & out-of-sample accuracy. However, this winning percent is more clear in highly imbalanced datasets. The Hybrid model is 1.5 times better than CART if the dataset is balanced, 1.7 times better if the dataset is highly imbalanced and finally, 1.3 times better if the dataset is imbalanced for out-of-sample accuracy. This also can be seen in Figure 14. It establishes that the win count percent for the hybrid model is not varying highly within the minority fraction groups.

Figure 14: In-Sample & Out-of-Sample Accuracy comparison of CART vs Hybrid Model in terms of Win Count in Minority Fraction Group Breakdown

5.5.4 Global Optimization Based Model Comparison

Global optimization-based models are so-called infinite horizon look ahead models which aim to optimize globally. We compared BinOCT, FlowOCT & BendersOCT, which are mentioned in Section 5.3.

FlowOCT is the winning method when compared to BinOCT. From a training accuracy perspective, FlowOCT highly dominates the BinOCT, this domination continues in test accuracy but to a lower degree. This can show that although BinOCT is worse in the training set than FlowOCT, it can produce a good result with respect to it. Figure 15 shows this situation in detail. Since FlowOCT is better, we will compare it with BendersOCT.

When comparing FlowOCT with BendersOCT, BendersOCT is better with respect to it, in the aspect of Test Accuracy. Although the win count percent seems similar, as can be seen in Figure 16, BendersOCT consistently beats FlowOCT, except for depth 6. In training accuracy, both models perform similarly as expected.

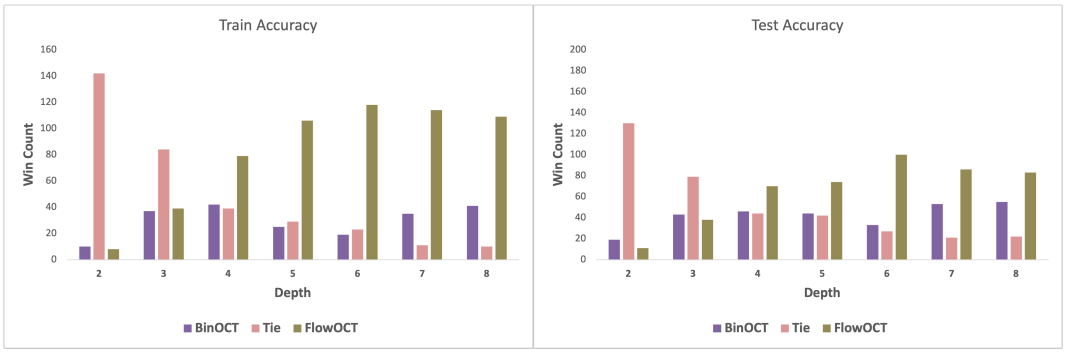

Figure 15: In-Sample & Out-of-Sample Accuracy comparison of BinOCT vs FlowOCT Model in terms of Win Count in Depth Breakdown

Note that as the depth increases or the instance size increases, these models struggle to solve and sometimes cannot produce feasible solutions in a reasonable time. In those situations, we fill forward the accuracy values from previous depths. By virtue of the figure, we will compare the BendersOCT method with our Hybrid model.

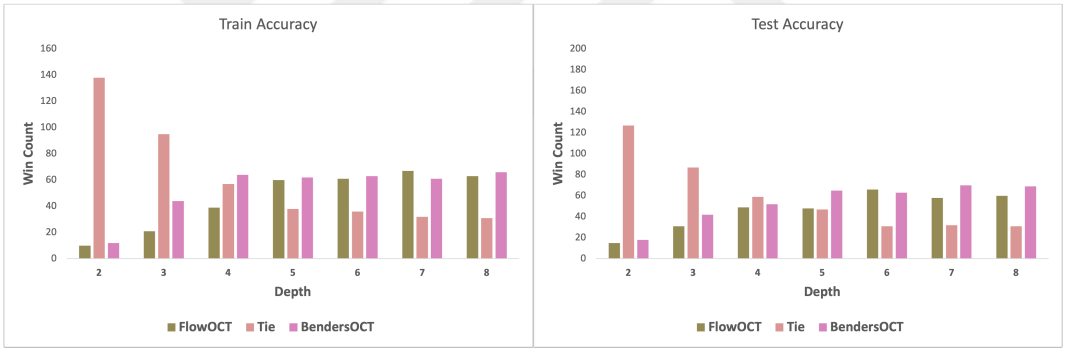

Figure 16: In-Sample & Out-of-Sample Accuracy comparison of FlowOCT vs BendersOCT Model in terms of Win Count in Depth Breakdown

Hybrid model outperforms BendersOCT in terms of win count, both in the perspective of train & test accuracy. The winning percentage is up to near 70% after depth 5 and is more than 60% in in-sample & out-of-sample accuracy, respectively. As can be seen from Figure 17, in depth 2, most of them are in a tie and for depths 4 & 5, Hybrid Model is slightly better than BendersOCT. However, in general, as the depth increases, Hybrid Model tends to outperform BendersOCT for both of the metrics. We compare the hybrid model with global optimization based models they aim to optimize by infinite-horizon look-ahead, on the other

hand, hybrid model 2-step look-ahead. It demonstrate that smaller look-ahead horizon is better than infinite looking. Also, it tries to guarantee optimality in larger depths with respect to hybrid model. However, the time complexity is much higher for global optimization based models than Hybrid model as the instance size increases or as the depth increases.

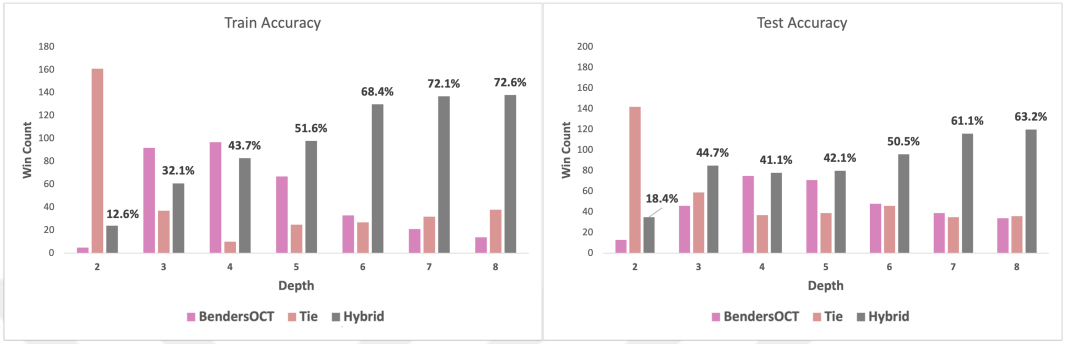

Figure 17: In-Sample & Out-of-Sample Accuracy comparison of BendersOCT vs Hybrid model in terms of Win Count in Depth Breakdown

Besides win count, Hybrid Model is winning for every depth in average test accuracy values. The average accuracy is obtained by taking average of all the dataset and fold combinations. Hybrid model is about 89% when the final depth is 8, on the other hand, BendersOCT is around 80%. There are almost 10 basis points difference between these models, as can be clearly seen in Figure 19. The reason why the gap between models is increasing as the depth increases is that the global optimised model is less likely to produce worse or no results. This also endorsed that the introduced Hybrid Model is better than any other introduced global optimization approach in every aspect.

Additionally, Hybrid Model is also winning for every depth in average training accuracy metric. For lower depths, Hybrid can perform better than BendersOCT despite less margin. But when we consider BendersOCT as the global optimize model and can produce optimal results for depth 3, our model is producing better although it is not globally optimal. However, as the depth increases, difference between the models are up to 10% in favor of Hybrid Model, similar to test accuracy. The reason why the gap between models is increasing as the depth

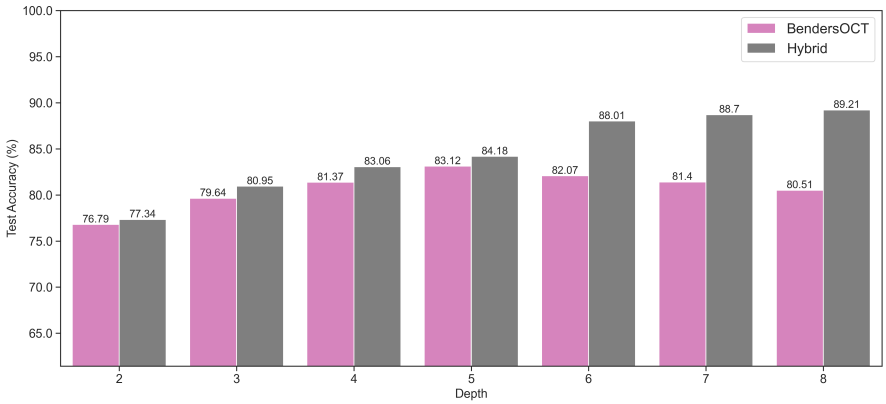

Figure 18: Average Out-of-Sample Accuracy comparison of BendersOCT vs Hybrid model in Depth Breakdown

increases as mentioned above. Figure 18 also summarizes the comparison in detail.

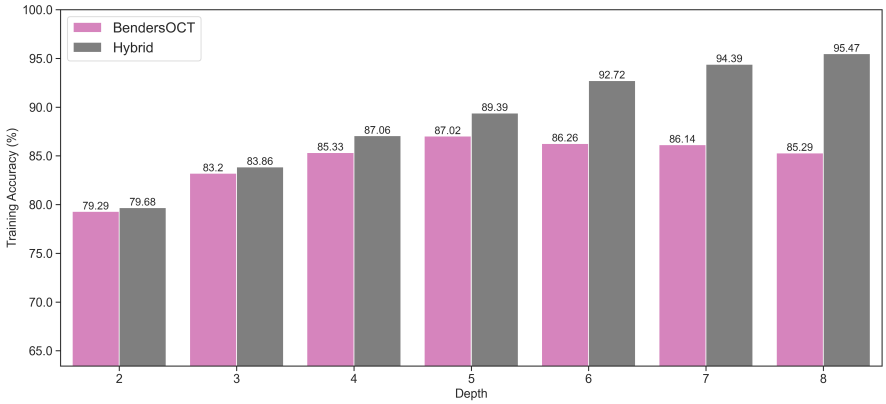

Figure 19: Average In-Sample Accuracy comparison of BendersOCT vs Hybrid model in Depth Breakdown

Table 3 also shows an in-depth dataset breakdown. The values represent (%) of the average out-of-sample accuracy for 10-Fold. The winning model for every dataset & depth pair is marked as bold for easier understanding. Remark that as the depth increases or the instance size increases, global optimised models struggle to produce even feasible solutions in a reasonable time. We fill forward the accuracy values from previous depths or folds. *Adult* & *nursery* datasets are large datasets which we able to test the limits of our algorithms. RLA-A model consistently outperforms for *adult* and Hybrid model consistently outperforms for

nursery dataset, except for the depth 2. For *nursery*, Hybrid model differs from FlowOCT for almost 10% and FlowOCT failed to generate solution after depth 4. In $monks-1$ dataset, all the models, except the BinOCT & CART, performed excellent. For $seismic-bumps$, which is medium-sized data, BinOCT dominates in general, although it is failed to generate solution after depth 4. The solution from depth 4 is used for rest of the depths and it still is the best. On the other hand, BendersOCT only dominates in $monks-3$ dataset. For $kr-vs-kp$ dataset, Hybrid Model consistently outperforms except for depth 4, but it is noteworthy that it is nearly 10% difference according to CART at depth 2. For $monks-2$ dataset, the average difference between RLA-G & RLA-A is the most, RLA-G is better than RLA-A in average for 8.7%.

Table 3: Avg. Out-of-Sample Accuracy for 10-Fold

Data	Depth	Bin OCT	Flow OCT	Benders OCT	CART	RLA-G	RLA-A	Hybrid
adult	2	76.1	76.7	76.7	79.1	79.1	**80.9**	**80.9**
adult	3	76.1	76.7	76.3	82.0	82.0	**82.1**	**82.1**
adult	4	76.1	76.7	76.3	82.2	82.2	**82.7**	**82.7**
adult	5	76.1	76.7	76.3	82.7	82.2	**83.0**	**83.0**
adult	6	76.1	76.7	76.3	82.9	82.8	**83.0**	82.8
adult	7	76.1	76.7	76.3	**82.9**	82.8	**82.9**	82.8
adult	8	76.1	76.7	76.3	82.9	82.9	**83.0**	82.9
agaricus-lepiota	2	69.4	95.8	**96.9**	95.4	**96.9**	**96.9**	**96.9**
agaricus-lepiota	3	69.4	63.3	96.7	98.5	98.8	**99.1**	**99.1**
agaricus-lepiota	4	69.4	63.3	98.4	99.2	**100.0**	**100.0**	**100.0**
agaricus-lepiota	5	69.4	63.3	99.8	99.9	**100.0**	**100.0**	**100.0**
agaricus-lepiota	6	69.4	63.3	91.3	**100.0**	**100.0**	**100.0**	**100.0**
agaricus-lepiota	7	69.4	63.3	96.6	**100.0**	**100.0**	**100.0**	**100.0**
agaricus-lepiota	8	69.4	63.3	98.3	**100.0**	**100.0**	**100.0**	**100.0**
balance-scale	2	63.5	63.5	63.5	**66.4**	63.7	63.5	63.5
balance-scale	3	66.4	**67.6**	**67.6**	65.8	62.4	67.1	67.1
balance-scale	4	69.9	**70.7**	70.3	65.1	64.2	68.2	68.2
balance-scale	5	71.5	72.7	**73.8**	70.9	70.1	70.6	70.6
balance-scale	6	63.9	73.8	**75.7**	73.0	73.1	73.1	73.1
balance-scale	7	60.8	73.8	74.7	**75.8**	72.5	72.3	72.5
balance-scale	8	60.5	71.1	72.0	**75.5**	73.8	72.6	73.8

Table 3 continued from previous page

Data	Depth	Bin OCT	Flow OCT	Benders OCT	CART	RLA-G	RLA-A	Hybrid
banknote-authentication	2	**73.9**	**73.9**	**73.9**	70.9	**73.9**	**73.9**	**73.9**
banknote-authentication	3	80.8	80.6	80.5	78.1	**80.9**	**80.9**	**80.9**
banknote-authentication	4	85.7	**89.3**	89.2	84.8	88.5	88.5	88.5
banknote-authentication	5	72.6	92.3	92.1	87.8	**92.6**	**92.6**	**92.6**
banknote-authentication	6	72.6	91.3	94.0	89.1	93.1	**95.0**	93.1
banknote-authentication	7	72.6	83.3	93.2	92.0	94.8	**95.5**	94.8
banknote-authentication	8	72.6	90.8	91.6	94.7	**96.7**	**96.1**	**96.7**
car-evaluation	2	77.8	**80.3**	77.8	77.8	77.8	77.8	77.8
car-evaluation	3	79.1	78.0	79.4	79.0	**81.2**	80.4	80.4
car-evaluation	4	81.2	**84.4**	81.6	79.6	81.3	81.1	81.1
car-evaluation	5	76.1	82.7	83.5	**86.0**	84.3	85.7	85.7
car-evaluation	6	71.2	72.3	82.9	86.6	88.3	**88.9**	88.3
car-evaluation	7	71.3	69.9	82.5	92.1	**93.9**	92.0	**93.9**
car-evaluation	8	71.3	69.9	80.6	93.7	**95.0**	93.9	**95.0**
diabetes	2	73.2	**73.5**	73.2	**73.5**	**73.5**	**73.5**	**73.5**
diabetes	3	72.2	**73.2**	72.0	72.6	71.7	72.6	72.6
diabetes	4	70.3	73.3	**73.5**	72.1	70.8	72.4	72.4
diabetes	5	72.5	71.6	**73.2**	71.3	70.0	72.6	72.6
diabetes	6	71.3	**73.2**	70.4	70.3	70.3	71.6	70.3
diabetes	7	69.6	65.1	68.9	69.8	70.5	**71.2**	70.5
diabetes	8	69.0	57.0	67.7	69.8	70.2	**70.9**	70.2
haberman	2	68.9	69.3	69.3	**73.5**	70.5	69.0	69.0
haberman	3	**68.9**	68.3	67.6	68.0	66.7	**68.9**	**68.9**
haberman	4	68.6	**70.9**	68.3	69.0	67.0	68.6	68.6
haberman	5	67.6	**68.3**	67.6	67.0	67.0	**68.3**	**68.3**
haberman	6	66.6	**69.6**	67.6	66.4	67.3	67.6	67.3
haberman	7	**69.3**	65.4	68.6	66.3	65.4	66.7	65.4
haberman	8	68.0	**71.0**	67.7	66.7	66.1	69.2	66.1
kr-vs-kp	2	**86.9**	**86.9**	**86.9**	76.8	**86.9**	**86.9**	**86.9**
kr-vs-kp	3	85.1	72.8	93.5	90.4	**93.8**	**93.8**	**93.8**
kr-vs-kp	4	74.1	79.0	**94.4**	94.1	93.2	94.1	94.1
kr-vs-kp	5	75.8	84.8	91.1	94.1	94.1	**95.2**	**95.2**
kr-vs-kp	6	75.8	85.2	79.8	93.6	96.4	**96.1**	**96.4**
kr-vs-kp	7	75.8	67.8	74.2	96.0	**97.9**	97.2	**97.9**

Table 3 continued from previous page

Data	Depth	Bin OCT	Flow OCT	Benders OCT	CART	RLA-G	RLA-A	Hybrid
kr-vs-kp	8	75.8	57.5	75.4	98.4	**98.5**	97.5	**98.5**
monks-1	2	74.3	73.7	74.1	**74.6**	73.6	73.4	73.4
monks-1	3	86.9	86.7	86.3	79.8	87.6	**88.3**	**88.3**
monks-1	4	**100.0**	**100.0**	**100.0**	82.0	96.6	98.2	98.2
monks-1	5	**100.0**	**100.0**	**100.0**	82.3	**100.0**	**100.0**	**100.0**
monks-1	6	92.1	**100.0**	**100.0**	80.9	**100.0**	**100.0**	**100.0**
monks-1	7	89.2	**100.0**	99.6	85.6	**100.0**	**100.0**	**100.0**
monks-1	8	87.1	**100.0**	**100.0**	91.4	**100.0**	**100.0**	**100.0**
monks-2	2	**65.7**	**65.7**	**65.7**	**65.7**	**65.7**	**65.7**	**65.7**
monks-2	3	57.1	56.7	57.2	63.1	**63.9**	62.7	62.7
monks-2	4	54.3	58.6	52.7	**63.2**	58.9	54.9	54.9
monks-2	5	68.7	78.0	71.4	75.2	**75.9**	53.4	53.4
monks-2	6	67.4	92.3	76.6	95.0	**100.0**	75.4	**100.0**
monks-2	7	66.7	89.8	65.4	99.5	**100.0**	93.9	**100.0**
monks-2	8	67.4	87.7	61.6	99.7	**100.0**	97.7	**100.0**
monks-3	2	96.4	96.4	**97.1**	96.4	96.4	96.4	96.4
monks-3	3	98.9	98.9	**99.1**	98.9	98.9	96.4	96.4
monks-3	4	98.9	98.9	**99.1**	98.9	98.9	96.4	96.4
monks-3	5	98.9	98.9	**99.1**	98.9	98.9	96.4	96.4
monks-3	6	95.4	98.7	98.7	**98.9**	98.7	96.1	98.7
monks-3	7	88.4	98.6	**98.7**	**98.7**	97.1	93.2	97.1
monks-3	8	85.9	98.2	**98.6**	98.0	97.5	93.3	97.5
nursery	2	40.6	**77.2**	76.3	76.3	76.3	76.3	76.3
nursery	3	45.2	77.2	78.0	**82.5**	81.8	**82.5**	**82.5**
nursery	4	45.2	84.9	69.9	85.3	87.2	**87.8**	**87.8**
nursery	5	45.2	84.9	68.8	87.6	88.8	**88.9**	**88.9**
nursery	6	45.2	84.9	59.2	89.6	**91.3**	90.9	**91.3**
nursery	7	45.2	84.9	73.4	92.0	**93.1**	92.6	**93.1**
nursery	8	45.2	84.9	72.4	94.0	**95.3**	94.8	**95.3**
seismic-bumps	2	93.1	93.1	93.1	**93.4**	**93.4**	93.2	93.2
seismic-bumps	3	93.0	**93.1**	93.0	93.0	92.8	92.7	92.7
seismic-bumps	4	**93.3**	92.9	92.9	92.9	92.8	92.7	92.7
seismic-bumps	5	**93.3**	92.9	93.1	93.0	92.2	92.4	92.4
seismic-bumps	6	**93.3**	92.9	93.2	92.4	92.0	92.1	92.0
seismic-bumps	7	**93.3**	93.1	93.2	92.0	92.1	91.9	92.1
seismic-bumps	8	**93.3**	93.1	93.2	91.3	91.8	91.9	91.8
spambase	2	83.6	**85.8**	81.7	79.8	84.2	85.6	85.6
spambase	3	61.4	80.0	79.1	87.4	**88.9**	88.6	88.6
spambase	4	62.2	80.1	72.3	**89.5**	89.0	89.4	89.4
spambase	5	62.2	73.4	71.5	89.4	89.5	**90.0**	**90.0**
spambase	6	62.2	73.8	71.9	89.8	90.0	**90.8**	90.0

Table 3 continued from previous page

Data	Depth	Bin OCT	Flow OCT	Benders OCT	CART	RLA-G	RLA-A	Hybrid
spambase	7	62.2	73.8	72.4	90.0	90.3	**90.6**	90.3
spambase	8	62.2	73.8	72.0	90.1	**90.7**	**90.7**	**90.7**
tae	2	79.5	79.5	76.8	79.5	**80.8**	80.1	80.1
tae	3	**82.2**	82.1	**82.2**	77.5	82.1	78.8	78.8
tae	4	80.8	82.1	**85.4**	76.1	84.7	80.1	80.1
tae	5	82.8	81.5	84.7	82.1	**86.0**	80.8	80.8
tae	6	82.1	84.8	**85.4**	82.8	82.8	79.4	82.8
tae	7	80.7	82.8	82.8	**84.1**	**84.1**	80.7	**84.1**
tae	8	83.4	83.4	84.1	82.8	**84.8**	81.4	**84.8**
tic-tac-toe	2	66.7	66.7	**67.0**	66.6	66.1	66.8	66.8
tic-tac-toe	3	72.1	72.8	**73.6**	72.5	72.0	72.8	72.8
tic-tac-toe	4	78.6	79.2	77.7	**82.1**	81.1	81.5	81.5
tic-tac-toe	5	74.6	79.8	82.2	**90.6**	89.5	82.1	82.1
tic-tac-toe	6	69.8	81.5	84.2	**92.8**	90.4	88.3	90.4
tic-tac-toe	7	67.5	83.1	81.7	**94.6**	90.3	89.3	90.3
tic-tac-toe	8	67.2	80.7	73.9	**93.5**	90.6	90.6	90.6
titanic	2	81.1	81.1	81.1	77.2	77.2	**81.2**	**81.2**
titanic	3	82.9	82.4	**83.2**	81.9	**83.2**	81.0	81.0
titanic	4	81.2	82.3	81.3	82.6	81.5	**82.7**	**82.7**
titanic	5	81.6	80.8	81.9	82.5	80.6	**82.2**	**82.2**
titanic	6	73.4	81.0	81.6	**82.0**	78.8	81.5	78.8
titanic	7	73.8	**81.3**	79.0	80.5	80.4	80.2	80.4
titanic	8	73.8	80.4	77.1	79.6	**80.9**	80.4	**80.9**
wdbc	2	79.8	78.7	79.4	**80.7**	78.6	78.8	78.8
wdbc	3	89.8	**90.2**	89.6	88.8	87.2	86.8	86.8
wdbc	4	90.5	92.6	**93.0**	90.0	90.4	90.9	90.9
wdbc	5	73.5	90.9	90.0	**91.8**	90.0	91.1	91.1
wdbc	6	71.5	**91.4**	89.1	91.2	91.1	90.4	91.1
wdbc	7	72.9	59.1	89.4	**91.1**	90.9	90.0	90.9
wdbc	8	72.9	52.0	88.1	**91.9**	90.5	89.7	90.5
wine	2	48.9	47.8	48.3	50.0	**55.1**	49.5	49.5
wine	3	59.6	60.2	58.5	57.8	61.3	**62.4**	**62.4**
wine	4	**72.5**	68.7	69.8	68.0	**72.5**	68.0	68.0
wine	5	79.3	**79.9**	79.3	78.7	79.8	74.3	74.3
wine	6	80.8	**86.6**	81.5	86.0	85.9	83.7	85.9
wine	7	79.2	83.8	76.0	88.8	**89.3**	87.6	**89.3**
wine	8	72.4	87.1	79.3	**89.9**	**89.9**	89.3	**89.9**

5.5.5 Time Complexity Comparison

One of the aforementioned limitations of the OCTs is the time complexity. The main bottleneck was to track every data point throughout the tree which is occurring as the instance size increases or the depth increases. However, our model has less time complexity than the previous models but it is slower when compared with the CART algorithm. The time-consuming part of both RLA-A & RLA-G models is the precalculation section. In general, the RLA-A model seems to take less than RLA-G Model, however, this can be negligible due to Gini Impurity calculation. As the Instance Size Group switches from small to large, the CART, although it is still fast, is 29 times slower than its average time. However, our RLA-G & RLA-A models are around 18 times slower than their average time, as can be seen in Table 4. The reason that the instance size is affecting our time complexity is that as the instance size increases, our algorithm tends to solve more sub trees and calculate Gini Impurity for larger datasets. The modelling part does not affect that much. This result is promising and at the same time shows that as the instance size increases, the time during the algorithm increases at a slower pace. However, there is still room for improving the algorithm as mentioned in Section 4, this was our scope at the time.

Table 4: Time (s) Comparison by Instance Size Group

Instance Size Group	**RLA-G Model**	**CART**	**RLA-A Model**
small	18.857	0.002	14.176
medium	27.476	0.007	22.355
large	353.335	0.058	269.671

One other thing that might affect our algorithm's time complexity is the feature size. Since we calculate Gini Impurity or the accuracy metric for splitting every feature combination, as the feature size increases, our time complexity also increases. From Feature Sizes small to large, RLA-A and RLA-G models took more time.

Table 5: Time (s) Comparison by Feature Size Group

Feature Size Group	**RLA-G Model**	**CART**	**RLA-A Model**
small	5.028	0.003	5.306
medium	8.912	0.003	7.527
large	165.693	0.024	124.886

Table 6 shows that as the depth increases RLA-G & RLA-A Model time complexity is increasing, however, this increase is in linear time. This is expected as the depth increase, although the number of subtrees required to solve is increasing, the data points in that subtrees are getting less. Figure 20 clearly shows the time change with respect to the previous depth, as the time required to solve for each level started to decrease after depth 3.

Table 6: Time (s) Comparison by Depth

Depth	**RLA-G Model**	**CART**	**RLA-A Model**
2	13.279	0.006	9.712
3	26.494	0.007	20.407
4	40.035	0.009	31.018
5	54.311	0.010	42.092
6	69.759	0.011	54.142
7	86.458	0.012	67.118
8	107.171	0.013	81.079

Table 7: Time (s) Comparison by Depth - Hybrid vs BendersOCT

Depth	**Hybrid Model**	**BendersOCT**
2	9.712	357.2
3	20.407	1078.3
4	31.018	1485.9
5	42.092	1559.0
6	69.759	1604.7
7	86.458	1683.7
8	107.171	1643.5

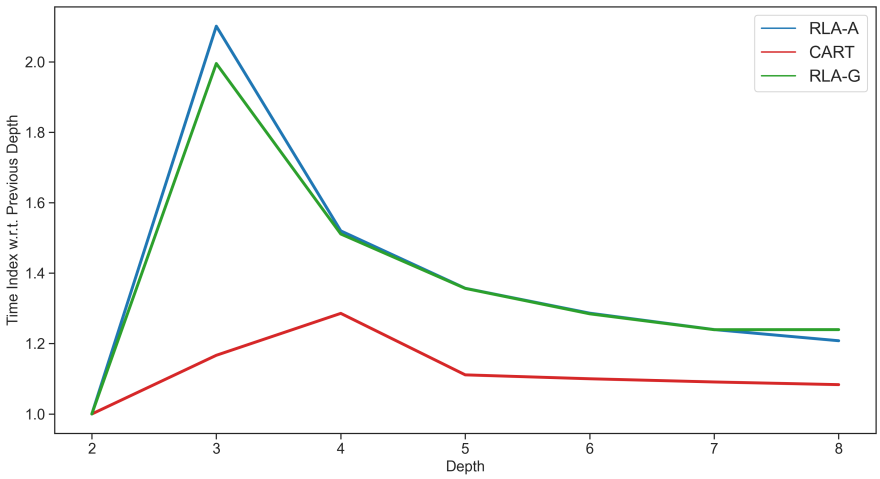

Figure 20: Time Change Index with respect to Previous Depth RunTime in Model Breakdown

Finally, Table 7 shows the average time comparison between Hybrid & BendersOCT. It proves that as the depth increases BendersOCT time complexity is increasing. Although values seem less than 1800 seconds which was our time limitation for the global optimization methods, for small datasets with higher depths or large datasets, BendersOCT runs for the whole given time. It establishes the scalability issues of global optimize OCTs. Usually, BendersOCT proves the optimality in depth 2 for most of the datasets, but on average, it took more than 5 minutes whereas the Hybrid Model took less than 10 seconds.

We observed that as the depth or instance size grows, global optimize methods struggle to provide even feasible solutions, on the other hand, our solution approach can easily scale larger depths. It also outperforms in training accuracy but performing better in test sets is more valuable. BendersOCT varying from 0.2% to 16.8%, on average 6% improvements, from 1% - 11%, on average 2.2% improvement on comparing with myopic algorithms. Finally, it provides a better solution with respect to CART although it is a bit slower than it.

CHAPTER VI

CONCLUSION

Decision trees has been drawing attention for almost 40 years and have retained their popularity thanks to its interpretability, applicability and scalability. In this thesis, we propose a novel formulation and a methodology for building binary optimal classification trees in a more scalable & better performing manner. This is compared to previously introduced globally optimization and already acknowledged myopic solutions. The methodology can be summarized as a rolling look-ahead approach by constructing two-depth optimal binary classification trees while minimizing Gini Impurity or misclassification error, exploring the spectrum between the 1-step look-ahead and infinite-horizon look-ahead, a middle ground between myopic and global optimization methods.

To the best of our knowledge, we firstly used a Gini Impurity index in an objective function of an optimal classification tree. Adding gini impurity creates a soft-clustering environment and performs better on test sets, as myopic heuristics tend to use it. In contrast, global optimization methods use misclassification errors in the objective, which performs well in in-sample accuracy, as expected. But at the same, it exposes the problem of look-ahead pathology and is likely to overfit & perform worse in unseen data. Also, for a fair comparison, we used misclassification error in the objective and compare it with global optimize OCTs, so-called infinite horizon look-ahead approaches. At last, the Hybrid approach is better than BendersOCT varying from 0.2% to 16.8%, on average 6% improvements. Moreover, Hybrid Model can produce great results in every depth of a dataset since global optimization methods cannot even produce a feasible solution in a reasonable time.

We also tested our approach in order to validate it in 19 different benchmark

datasets with different attributes and data group labels. According to observations, for every group from feature size to instance size, overfitting group to balanced datasets, except for some of the small instance sizes, the Hybrid model performs better concerning CART in test accuracy & terms of win count, consistently.

Since all the available datasets are not suitable for binary classification, we've pre-processed the datasets and binarized them. Further details are discussed in the Section 5.1. One of the limitations of our approach is that it does not guarantee optimality in larger depths. It can guarantee only in depth 2, as the model is formulated and fixed specifically for two-depth. However, it is also not a myopic solution like CART and it can provide better results in out-of-sample accuracy concerning CART & other global optimization benchmark models. Moreover, pruning was beyond the scope of this thesis, which requires a bit more research and work but it can be easily activated in our modelling approach. Finally, another limitation of our model can be discussed as time complexity. Although it is much faster with respect to global optimization methods, it is slower than the myopic solutions. For large datasets with maximum 50000 rows, it works less than 4 minutes for larger depths. But on the other hand, CART is performing less than a minute. As mentioned above, there is still room for improvement in the rolling look-ahead part of the algorithm, which constructs and optimizes tens of distinct two-depth trees. These tens of trees can work in parallel in every depth since they are independent of each other. Another computational burden of the approach is in the precalculation part of the optimization which requires enumeration. When the feature size increases, then the complexity of the algorithm also increases. There is also room for improvement in this area since it can be calculated independently for each leaf node.

APPENDIX A

TABLES

Table 8: Avg. In-Sample Accuracy for 10-Fold

Data	Depth	Bin OCT	Flow OCT	Benders OCT	CART	RLA-G	RLA-A	Hybrid
adult	2	76.1	76.6	76.8	79.1	79.1	**80.9**	**80.9**
adult	3	76.1	76.6	76.3	82.1	82.1	**82.2**	**82.2**
adult	4	76.1	76.6	76.3	82.2	82.3	**82.8**	**82.8**
adult	5	76.1	76.6	76.3	82.8	82.6	**83.2**	**83.2**
adult	6	76.1	76.6	76.3	83.1	83.1	**83.5**	83.1
adult	7	76.1	76.6	76.3	83.4	83.5	**83.9**	83.5
adult	8	76.1	76.6	76.3	83.7	83.9	**84.3**	83.9
agaricus-lepiota	2	72.7	96.5	**96.9**	95.4	**96.9**	**96.9**	**96.9**
agaricus-lepiota	3	72.7	62.4	96.8	98.5	99.0	**99.1**	**99.1**
agaricus-lepiota	4	72.7	62.4	98.5	99.4	**100.0**	**100.0**	**100.0**
agaricus-lepiota	5	72.7	62.4	99.8	99.9	**100.0**	**100.0**	**100.0**
agaricus-lepiota	6	72.7	62.4	91.3	**100.0**	**100.0**	**100.0**	**100.0**
agaricus-lepiota	7	72.7	62.4	96.6	**100.0**	**100.0**	**100.0**	**100.0**
agaricus-lepiota	8	72.7	62.4	98.4	**100.0**	**100.0**	**100.0**	**100.0**
balance-scale	2	**68.7**	**68.7**	**68.7**	68.4	**68.7**	**68.7**	**68.7**
balance-scale	3	74.4	**74.6**	**74.6**	69.9	70.5	74.0	74.0
balance-scale	4	77.3	**77.9**	78.0	71.3	73.7	77.5	77.5
balance-scale	5	78.6	81.0	81.2	78.7	80.0	**81.4**	**81.4**
balance-scale	6	67.1	82.4	83.7	83.5	84.8	**84.9**	84.8
balance-scale	7	65.9	80.6	82.6	86.4	**88.8**	88.3	88.8
balance-scale	8	65.6	81.5	80.4	89.1	**91.8**	91.3	**91.8**
banknote-authentication	2	**73.9**	**73.9**	**73.9**	72.3	**73.9**	**73.9**	**73.9**
banknote-authentication	3	**81.7**	**81.7**	**81.7**	78.9	81.4	81.4	81.4
banknote-authentication	4	86.0	**89.5**	**89.5**	84.9	88.8	88.8	88.8
banknote-authentication	5	72.1	93.6	**93.7**	88.2	92.8	92.8	92.8
banknote-authentication	6	72.1	92.2	95.0	89.7	94.5	**95.3**	94.5
banknote-authentication	7	72.1	84.8	94.3	92.9	**96.9**	96.8	**96.9**

Table 8 continued from previous page

Data	Depth	Bin OCT	Flow OCT	Benders OCT	CART	RLA-G	RLA-A	Hybrid
banknote-authentication	8	72.1	91.9	93.1	96.0	**98.1**	98.0	**98.1**
car-evaluation	2	**77.8**	77.5	**77.8**	**77.8**	**77.8**	**77.8**	**77.8**
car-evaluation	3	80.4	80.8	**81.3**	80.7	81.1	81.1	81.1
car-evaluation	4	81.4	**84.3**	84.0	81.7	82.8	83.2	83.2
car-evaluation	5	75.7	84.6	86.1	86.8	85.4	**87.1**	**87.1**
car-evaluation	6	71.4	74.8	85.0	89.0	90.6	**91.1**	90.6
car-evaluation	7	71.6	70.0	84.1	92.6	**95.3**	94.4	**95.3**
car-evaluation	8	71.6	70.0	81.5	95.0	**97.0**	96.4	**97.0**
diabetes	2	**75.0**	**75.0**	**75.0**	74.7	74.9	**75.0**	**75.0**
diabetes	3	76.5	76.4	76.6	75.3	76.1	**76.8**	**76.8**
diabetes	4	77.5	76.5	76.5	75.9	77.6	**79.0**	**79.0**
diabetes	5	74.9	79.5	78.6	77.2	79.8	**81.5**	**81.5**
diabetes	6	72.3	77.4	74.8	78.5	81.8	**84.2**	81.8
diabetes	7	70.4	71.3	72.5	80.3	84.3	**86.9**	84.3
diabetes	8	69.7	57.7	72.2	82.1	86.2	**89.5**	86.2
haberman	2	**76.2**	**76.2**	**76.2**	73.5	74.5	**76.2**	**76.2**
haberman	3	**78.3**	**78.3**	**78.3**	74.9	76.0	78.2	78.2
haberman	4	**80.2**	80.0	80.1	77.9	78.3	**80.2**	**80.2**
haberman	5	82.2	82.3	**82.4**	79.8	80.6	82.1	82.1
haberman	6	83.0	84.4	**84.9**	82.2	83.4	84.0	83.4
haberman	7	79.1	**85.8**	85.7	83.8	85.4	85.6	85.4
haberman	8	77.5	**87.3**	85.7	84.9	86.8	87.1	86.8
kr-vs-kp	2	**86.9**	**86.9**	**86.9**	77.6	**86.9**	**86.9**	**86.9**
kr-vs-kp	3	84.2	74.0	93.5	90.4	**93.8**	93.8	**93.8**
kr-vs-kp	4	74.1	79.8	**94.6**	94.1	94.0	94.4	94.4
kr-vs-kp	5	76.1	85.4	91.9	94.1	94.7	**95.2**	**95.2**
kr-vs-kp	6	76.1	84.9	80.3	94.3	**96.4**	96.0	**96.4**
kr-vs-kp	7	76.1	69.0	75.6	96.7	**98.5**	97.7	**98.5**
kr-vs-kp	8	76.1	57.1	74.5	98.6	**99.1**	98.2	**99.1**
monks-1	2	**78.0**	**78.0**	**78.0**	74.6	**78.0**	**78.0**	**78.0**
monks-1	3	**89.8**	**89.8**	**89.8**	81.6	88.1	89.3	89.3
monks-1	4	**100.0**	**100.0**	**100.0**	84.6	96.6	97.6	97.6
monks-1	5	**100.0**	**100.0**	**100.0**	86.3	**100.0**	**100.0**	**100.0**
monks-1	6	92.6	**100.0**	**100.0**	87.3	**100.0**	**100.0**	**100.0**
monks-1	7	89.6	**100.0**	**100.0**	91.3	**100.0**	**100.0**	**100.0**
monks-1	8	88.9	**100.0**	**100.0**	95.7	**100.0**	**100.0**	**100.0**
monks-2	2	**65.7**	**65.7**	**65.7**	**65.7**	**65.7**	**65.7**	**65.7**
monks-2	3	**67.7**	67.6	**67.7**	66.2	66.5	66.0	66.0
monks-2	4	70.1	70.1	**70.2**	68.5	69.4	68.7	68.7
monks-2	5	76.7	82.2	79.3	78.8	**84.1**	73.4	73.4

Table 8 continued from previous page

Data	Depth	Bin OCT	Flow OCT	Benders OCT	CART	RLA-G	RLA-A	Hybrid
monks-2	6	71.3	95.3	86.3	96.5	**100.0**	84.7	**100.0**
monks-2	7	67.7	94.4	76.0	99.3	**100.0**	97.9	**100.0**
monks-2	8	69.3	91.8	73.3	99.7	**100.0**	99.7	**100.0**
monks-3	2	96.4	96.4	**96.9**	96.4	96.4	96.4	96.4
monks-3	3	**98.9**	**98.9**	**98.9**	**98.9**	**98.9**	96.4	96.4
monks-3	4	**98.9**	**98.9**	**98.9**	**98.9**	**98.9**	96.4	96.4
monks-3	5	**98.9**	**98.9**	**98.9**	**98.9**	**98.9**	96.4	96.4
monks-3	6	95.8	**98.9**	**98.9**	**98.9**	**98.9**	96.4	**98.9**
monks-3	7	88.9	**99.0**	98.9	98.9	**99.0**	96.7	**99.0**
monks-3	8	86.3	98.9	98.9	98.9	**99.0**	97.6	**99.0**
nursery	2	40.6	76.2	**76.4**	**76.4**	**76.4**	**76.4**	**76.4**
nursery	3	45.2	76.2	78.0	**82.5**	82.4	**82.5**	**82.5**
nursery	4	45.2	83.4	70.0	85.3	87.2	**87.8**	**87.8**
nursery	5	45.2	83.4	68.9	87.6	89.0	**89.1**	**89.1**
nursery	6	45.2	83.4	59.2	89.7	**91.7**	91.1	**91.7**
nursery	7	45.2	83.4	73.0	92.2	**93.6**	92.9	**93.6**
nursery	8	45.2	83.4	71.9	94.4	**95.9**	95.1	**95.9**
seismic-bumps	2	**93.5**	**93.5**	**93.5**	93.4	93.4	**93.5**	**93.5**
seismic-bumps	3	93.6	93.6	93.6	93.5	**93.7**	**93.7**	**93.7**
seismic-bumps	4	93.6	93.8	93.7	**93.8**	94.2	94.0	94.0
seismic-bumps	5	93.5	93.9	93.7	94.2	**94.8**	94.4	94.4
seismic-bumps	6	93.5	93.8	93.6	94.7	**95.5**	95.0	**95.5**
seismic-bumps	7	93.5	93.9	93.5	95.3	**96.1**	95.5	**96.1**
seismic-bumps	8	93.5	93.9	93.5	95.9	**96.7**	96.1	**96.7**
spambase	2	83.9	**86.1**	82.3	80.1	85.4	**86.1**	**86.1**
spambase	3	61.5	80.5	80.2	87.4	**89.1**	**89.1**	**89.1**
spambase	4	62.2	81.7	72.6	89.7	90.1	**90.4**	**90.4**
spambase	5	60.9	72.9	71.5	90.3	90.8	**91.7**	**91.7**
spambase	6	60.9	73.8	71.9	91.4	91.9	**92.8**	91.9
spambase	7	60.9	73.8	72.8	92.4	93.3	**93.7**	93.3
spambase	8	60.9	73.8	72.3	93.4	**94.7**	94.5	**94.7**
tae	2	**84.5**	**84.5**	**84.5**	83.5	83.5	**84.5**	**84.5**
tae	3	**89.0**	**89.0**	**89.0**	86.5	88.4	87.7	87.7
tae	4	93.2	93.5	**93.6**	88.8	92.3	90.7	90.7
tae	5	96.1	**96.3**	96.2	92.7	94.3	92.7	92.7
tae	6	97.0	**97.5**	97.1	95.0	95.9	93.8	95.9
tae	7	96.3	97.5	**97.6**	96.2	97.1	95.2	97.1
tae	8	94.5	**97.6**	**97.6**	97.3	**97.6**	96.7	**97.6**
tic-tac-toe	2	**70.9**	**70.9**	**70.9**	70.7	70.6	**70.9**	**70.9**
tic-tac-toe	3	77.2	77.5	77.3	75.5	76.2	**77.4**	**77.4**
tic-tac-toe	4	83.0	83.7	83.3	83.9	84.8	**84.9**	**84.9**

Table 8 continued from previous page

Data	Depth	Bin OCT	Flow OCT	Benders OCT	CART	RLA-G	RLA-A	Hybrid
tic-tac-toe	5	77.2	86.1	86.2	91.1	**92.3**	89.6	89.6
tic-tac-toe	6	71.1	84.2	87.5	95.0	**96.8**	94.8	**96.8**
tic-tac-toe	7	69.7	85.1	84.3	97.8	**98.9**	98.3	**98.9**
tic-tac-toe	8	69.2	82.9	77.4	98.9	**99.8**	99.3	**99.8**
titanic	2	81.1	81.1	81.1	78.9	78.9	**81.2**	**81.2**
titanic	3	83.4	83.4	**83.6**	82.3	83.4	82.2	82.2
titanic	4	84.1	84.3	**84.7**	84.3	84.4	84.3	84.3
titanic	5	84.1	85.4	85.0	85.2	**86.1**	85.9	85.9
titanic	6	76.8	85.4	84.9	86.3	**87.5**	87.3	**87.5**
titanic	7	76.7	85.3	82.3	87.6	**89.4**	88.6	**89.4**
titanic	8	76.7	86.2	81.2	88.7	**90.7**	89.8	**90.7**
wdbc	2	**83.1**	**83.1**	**83.1**	82.9	**83.1**	**83.1**	**83.1**
wdbc	3	92.1	**92.2**	**92.2**	90.9	90.9	91.0	91.0
wdbc	4	92.8	95.2	**96.9**	93.1	93.7	94.0	94.0
wdbc	5	76.9	93.6	96.0	94.8	**96.3**	96.1	96.1
wdbc	6	76.6	94.4	94.7	96.5	**97.9**	97.7	**97.9**
wdbc	7	75.9	59.1	93.5	97.6	**98.9**	98.6	**98.9**
wdbc	8	75.9	52.0	93.1	98.4	**99.6**	99.2	**99.6**
wine	2	**62.0**	**62.0**	**62.0**	59.3	59.6	**62.0**	**62.0**
wine	3	**71.6**	**71.6**	**71.6**	69.8	70.8	71.5	71.5
wine	4	80.0	**80.1**	**80.1**	78.5	77.8	79.3	79.3
wine	5	85.6	**87.8**	**87.8**	85.5	85.3	85.8	85.8
wine	6	87.2	93.5	**93.8**	91.5	90.9	91.4	90.9
wine	7	87.2	**97.1**	**97.1**	94.8	94.8	95.2	94.8
wine	8	82.2	96.9	**99.1**	97.9	97.1	97.4	97.1

REFERENCES

[1] L. Breiman, "Random forests," *Machine learning*, vol. 45, no. 1, pp. 5–32, 2001.

[2] L. Breiman, J. Friedman, R. Olshen, and C. Stone, "Classification and regression trees," 1984.

[3] J. R. Quinlan, *C4.5: Programs for machine learning*. The Morgan Kaufmann series in machine learning, San Mateo, California: Morgan Kaufmann Publishers, 1993.

[4] J. R. Quinlan, "Induction of decision trees," *Machine learning*, vol. 1, no. 1, pp. 81–106, 1986.

[5] H. Laurent and R. L. Rivest, "Constructing optimal binary decision trees is np-complete," *Information processing letters*, vol. 5, no. 1, pp. 15–17, 1976.

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

[7] S. Verwer and Y. Zhang, "Learning optimal classification trees using a binary linear program formulation," vol. 33, no. 01, pp. 1625–1632, 2019.

[8] S. Murthy and S. Salzberg, "Lookahead and pathology in decision tree induction," in *IJCAI*, pp. 1025–1033, Citeseer, 1995.

[9] M. Last and M. Roizman, "Avoiding the look-ahead pathology of decision tree learning," *International journal of intelligent systems*, vol. 28, no. 10, pp. 974–987, 2013.

[10] K. P. Bennett, "Decision tree construction via linear programming," tech. rep., University of Wisconsin-Madison Department of Computer Sciences, 1992.

[11] M. Menickelly, O. Günlük, J. Kalagnanam, and K. Scheinberg, "Optimal generalized decision trees via integer programming," *CoRR, abs/1612.03225*, 2016.

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

[13] K. P. Bennett and J. A. Blue, "Optimal decision trees," tech. rep., 1996.

[14] N. Larus-Stone, E. Angelino, D. Alabi, M. Seltzer, V. Kaxiras, A. Saligrama, and C. Rudin, "Systems optimizations for learning certifiably optimal rule lists," in *SysML Conference*, p. 16, 2018.

[15] E. Angelino, N. Larus-Stone, D. Alabi, M. Seltzer, and C. Rudin, "Learning certifiably optimal rule lists for categorical data," *arXiv preprint arXiv:1704.01701*, 2017.

[16] X. Hu, C. Rudin, and M. Seltzer, "Optimal sparse decision trees," *Advances in Neural Information Processing Systems*, vol. 32, 2019.

[17] R. Blanquero, E. Carrizosa, C. Molero-Río, and D. R. Morales, "Sparsity in optimal randomized classification trees," *European Journal of Operational Research*, vol. 284, no. 1, pp. 255–272, 2020.

[18] R. Blanquero, E. Carrizosa, C. Molero-Río, and D. R. Morales, "Optimal randomized classification trees," *Computers & Operations Research*, vol. 132, p. 105281, 2021.

[19] S. Aghaei, A. Gomez, and P. Vayanos, "Learning optimal classification trees: Strong max-flow formulations," *arXiv preprint arXiv:2002.09142*, 2020.

[20] M. Garofalakis, D. Hyun, R. Rastogi, and K. Shim, "Building decision trees with constraints," *Data Mining and Knowledge Discovery*, vol. 7, no. 2, pp. 187–214, 2003.

[21] S. Nijssen and E. Fromont, "Mining optimal decision trees from itemset lattices," in *Proceedings of the 13th ACM SIGKDD international conference on Knowledge discovery and data mining*, pp. 530–539, 2007.

[22] S. Nijssen and E. Fromont, "Optimal constraint-based decision tree induction from itemset lattices," *Data Mining and Knowledge Discovery*, vol. 21, no. 1, pp. 9–51, 2010.

[23] G. Aglin, S. Nijssen, and P. Schaus, "Learning optimal decision trees using caching branch-and-bound search," in *Proceedings of the AAAI Conference on Artificial Intelligence*, vol. 34, pp. 3146–3153, 2020.

[24] G. Aglin, S. Nijssen, and P. Schaus, "Pydl8. 5: a library for learning optimal decision trees," in *Proceedings of the Twenty-Ninth International Conference on International Joint Conferences on Artificial Intelligence*, pp. 5222–5224, 2021.

[25] J. Lin, C. Zhong, D. Hu, C. Rudin, and M. Seltzer, "Generalized and scalable optimal sparse decision trees," pp. 6150–6160, 2020.

[26] E. Demirović, A. Lukina, E. Hebrard, J. Chan, J. Bailey, C. Leckie, K. Ramamohanarao, and P. J. Stuckey, "Murtree: optimal classification trees via dynamic programming and search," *arXiv preprint arXiv:2007.12652*, 2020.

[27] S. Aghaei, M. J. Azizi, and P. Vayanos, "Learning optimal and fair decision trees for non-discriminative decision-making," vol. 33, no. 01, pp. 1418–1426, 2019.

[28] E. Demirović and P. J. Stuckey, "Optimal decision trees for nonlinear metrics," vol. 35, no. 5, pp. 3733–3741, 2021.

[29] "Optimal decision trees Interpretable AI." https://www.interpretable.ai/products/optimal-trees/(Accessed July 21, 2022).

[30] T. Hastie, R. Tibshirani, and J. Friedman, *The Elements of Statistical Learning*. Springer Series in Statistics, New York, NY: Springer New York, 2009.

[31] "OneHotEncoder." https://scikit-learn/stable/modules/generated/sklearn.preprocessing.OneHotEncoder.html (Accessed July 21, 2022).

[32] "StratifiedKFold." https://scikit-learn/stable/modules/generated/sklearn.model_selection.StratifiedKFold.html (Accessed July 21, 2022).

[33] "DecisionTreeClassifier." https://scikit-learn/stable/modules/generated/sklearn.tree.DecisionTreeClassifier.html (Accessed July 21, 2022).

[34] S. Aghaei, A. Gómez, and P. Vayanos, "Strong optimal classification trees," *arXiv preprint arXiv:2103.15965*, 2021.

[35] "StrongTree." https://github.com/pashew94/StrongTree (Accessed July 21, 2022).

[36] "Gurobi 9.1: The Best-of-Breed Mathematical." https://www.gurobi.com/news/gurobi-9-1-the-best-of-breed-mathematical-optimization-solver-just-got-better/ (Accessed July 21, 2022).

[37] "UCI Machine Learning Repository." https://archive.ics.uci.edu/ml/index.php (Accessed July 21, 2022).

[38] S. Sethi and G. Sorger, "A theory of rolling horizon decision making," *Annals of Operations Research*, vol. 29, no. 1, pp. 387–415, 1991.

[39] C. Wang, H. C. Lau, and Y. F. Lim, "A rolling horizon auction mechanism and virtual pricing of shipping capacity for urban consolidation centers," in *International Conference on Computational Logistics*, pp. 422–436, Springer, 2015.

VITA

Zeynel Batuhan Organ graduated from TED Bursa High School in 2014 and earned his B.Sc. in Industrial Engineering from Özyeğin University in August 2018. He has been pursuing his M.Sc. degree in Industrial Engineering at Özyeğin University since September 2019, under the supervision of Asst. Prof. Enis Kayış and Asst. Prof. Tagi Hanalioğlu. He has been working at Invent Analytics as a data scientist since June 2018. His interests are optimal classification trees, big data applications and time series forecasting.