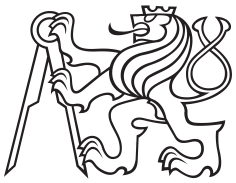
Master Thesis



Czech Technical University in Prague

F3

Faculty of Electrical Engineering
Department of Computer Science

Mixed-integer Programming in Machine Learning: Decision Trees and Neural Networks

Bc. Jiří Němeček

Supervisor: Mgr. Jakub Mareček, Ph.D.
Field of study: Open Informatics
Subfield: Artificial Intelligence
May 2023

Acknowledgements

Firstly, I would like to thank my supervisor Mgr. Jakub Mareček, Ph.D., for his guidance and pragmatic approach. I also thank doc. Ing. Tomáš Pevný, Ph.D., for his insights regarding the experimentation.

I acknowledge the benefits of being able to use the RCI Cluster. My ability to experiment was greatly enhanced because of it and the people behind it.

Lastly, I sincerely thank everyone who supports me and those who do not stand in the way.

This work has received funding from the European Union's Horizon Europe research and innovation programme under grant agreement No. 101084642.

The RCI cluster has been supported by the OP RDE funded project 380 CZ.02.1.01/0.0/0.0/16_019/0000765 "Research Center for Informatics".

Declaration

I declare that the presented work was developed independently, and I have listed all sources of information used within it in accordance with the methodical instructions for observing the ethical principles in the preparation of university theses.

In Prague, 23. May 2023

.....

Prohlašuji, že jsem předloženou práci vypracoval samostatně a že jsem uvedl veškeré použité informační zdroje v souladu s Metodickým pokynem o dodržování etických principů při přípravě vysokoškolských závěrečných prací.

V Praze, 23. 5. 2023

.....

Abstract

In Machine learning (ML), the question of explainability and fairness is becoming ever more critical. New concepts of desired properties of ML models are being introduced and often require looking for other optimization methods. Mixed-integer programming (MIP) is one such optimization method. We overview the wide use of MIP in ML and explore two topics in depth.

We mainly contribute with a MIP formulation of a classification tree, focusing on a fair accuracy distribution over its leaves. We call the formulation Fair Classification Tree (FCT). With it, we tackle a currently overlooked problem. By adding further models to the leaves of the FCTs, we show that the extended models have comparable accuracy to state-of-the-art models while significantly improving the explainability of the model.

Secondly, we propose a possible way to assess the quality of a model's counterfactual explanations using a function and offer its use in model training in search of balance in explainability and accuracy at once. Counterfactual explanations are a post-hoc explanation technique widely used in fields ranging from finance to healthcare.

Keywords: mixed-integer programming, machine learning, fairness, decision trees, counterfactual explanations

Supervisor: Mgr. Jakub Mareček, Ph.D.

Abstrakt

Ve strojovém učení se otázka vysvětlitelnosti a férovosti stává stále více důležitou. Nové koncepty požadovaných vlastností modelů strojového učení jsou představovány a často vyžadují jiné metody pro optimalizaci. Jednou z takových metod je smíšené celočíselné programování (MIP). Vytváříme přehled širokého použití MIP ve strojovém učení a rozvádíme dvě témata do hloubky.

Primárně představujeme MIP formulaci rozhodovacího stromu, zaměřenou na férové rozdělení přesnosti na jeho listech. Nazýváme tuto formulaci Fair Classification Tree (FCT). Pomocí ní se zaměřujeme na dosud přehlížený problém. Přidáním dalších modelů do listů FCT ukazujeme, že rozšířené modely mají přesnost srovnatelnou se současně nejlepšími modely, zatímco značně zlepšují vysvětlitelnost modelu.

Dále představujeme funkci pro hodnocení kvality kontrafaktuálních vysvětlení modelu, kterou nabízíme k použití při trénování modelu pro hledání rovnováhy mezi vysvětlitelností a přesností modelu. Kontrafaktuální vysvětlení jsou post-hoc vysvětlovací technikou široce používanou v oblastech od financí po zdravotnictví.

Klíčová slova: smíšené celočíselné programování, strojové učení, férovost, rozhodovací stromy, kontrafaktuální vysvětlení

Překlad názvu: Smíšené celočíselné programování ve strojovém učení

Contents

Figures

Tables

Chapter 1

Introduction

Artificial Intelligence (AI) is a topic that seems to be getting ever more important over time. In recent months, with the rise of large language models, practically everyone is getting into contact with the capabilities of AI [Hu, 2023]. As the broad society learns to work with AI models, the models must gain users' trust. The requirements for the capabilities of models are expanding from pure quantitative performance to a more complex optimization goal. It often no longer suffices to show that model is accurate. In many cases, society demands fair treatment, transparency, and accountability.

With great power comes great responsibility. As the interest in AI is rising, so does the interest in Explainable Artificial Intelligence (XAI). XAI is the concept of providing the user of an AI model with not only a prediction but also an explanation. This explanation might be regarding a single prediction or even broader concepts, such as explaining the model's weaknesses and warning the user if a prediction might be wrong. The purpose of this is to improve understanding and to increase the trustworthiness of AI models [DARPA, 2016; Saeed and Omlin, 2021]. For some areas, such as healthcare and security, the existence of high-quality XAI methods is essential to achieve wide adoption of AI in those areas [Adadi and Berrada, 2018].

One of the primary goals of XAI is delivering good explanations without sacrificing too much accuracy [Adadi and Berrada, 2018]. That is commonly believed to be non-trivial. However, Rudin [2019] claims this perceived trade-off is a myth. Nevertheless, the prevalence of black-box models is undisputed.

A different term, close to explainability, is interpretability. The distinction between their meanings varies in existing works. Saeed and Omlin [2021] review some of them and provide an overarching definition:

> "*Explainability* provides insights to a targeted audience to fulfill a need, whereas *interpretability* is the degree to which the provided insights can make sense for the targeted audience's domain knowledge." [Saeed and Omlin, 2021, page 4]

In other words, explainability is the ability of a model to explain its decision or some of its attributes or features. On the other hand, interpretability is the ability of users of the model (or other interested people) to understand the explanations and conclude something from them.

A similar distinction is between local and global explanations. While local explanations help us understand a decision of a model, global explanations help us understand the full model, enable us to analyze it, and possibly spot some inconsistencies or flaws.

The issues of explainability and interpretability of AI models are the motivating themes of this work.

1.1 Why Mixed-integer programming (MIP)?

MIP and especially Mixed-Integer Linear Programming, which we will consider mostly, is a very expressive framework of mathematical problem specification. Mixed-integer linear program is essentially a linear program where at least one of the variables takes only integer values. General MIP formulation belongs to the class of NP-complete problems. [Shahrabi Farahani and Lagergren, 2013]

On the one hand, this means that MIP formulations are significantly more challenging to solve compared to Linear programming. On the other hand, this enables us to specify any NP-hard problem using MIP and then use highly optimized general solvers, like CPLEX [Cplex, 2009] or Gurobi [Gurobi Optimization, LLC, 2023] to obtain the solutions. This capability is the cornerstone of many applications.

1.2 Goals of the thesis

This thesis aims to do three things. Firstly, we wish to provide a quick overview of the current use of MIP in AI. Secondly, provide the reader with some propositions for optimization functions regarding counterfactuals. And lastly, as the central part of the thesis, we introduce a new variant of Hybrid tree [Zhou and Chen, 2002] with a focus on the balance of explainability and accuracy.

1.2.1 Use of MIP methods in Machine learning (ML)

Initially, we will do a quick overview of various uses of MIP in ML. For a more in-depth look, we will refer the reader to further sources, not to repeat what has been said but to provide an overlooking view. This part will put focus on the uses of MIP for explainability and interpretability, describing counterfactual explanations in detail. However, it will not be focusing exclusively on them only.

1.2.2 Optimizing explanations' validity in Classification trees

The most significant part of the thesis will touch on the problem of finding optimal classification trees. We will propose a new MIP formulation that optimizes a different goal from the most common one, focusing on the validity of explanations that the tree provides. It will be done by maximizing the leaf accuracy of the least accurate leaf.

1.2.3 Counterfactual generation using MIP

The second practical topic will concern the generation of counterfactual explanations of the decisions of Artifical Neural networks (NNs). We will introduce a couple of potential utility functions, based on a set of counterfactuals, aiming to train models with local explanations in mind. This could aid models, that are not interpretable but can provide counterfactual explanations.

Declaration note. The model regarding the classification trees and global explanations created in Chapter 3 of the thesis was submitted for review as a scientific article titled *Improving the Validity of Decision Trees as Explanations*. The dates of submission collided with the submission of the thesis. It is partly similar to Chapter 3, especially with regard to the experiments. The description here is more detailed.

Chapter 2

Current use of MIP in ML

From the discussion of Glover [1986], we can see that because of its ability to express NP-hard problems, MIP played a role in the development and use of AI methods. We will concern ourselves strictly with Machine learning (ML) models. That is by using the definition by Zhou [2021]:

> "Machine learning is the technique that improves system performance by learning from experience via computational methods." [Zhou, 2021, page 2]

This means we are interested in the algorithms that use data to learn concepts rather than solving a single complex task with simple, smaller input, such as planning [Vossen et al., 1999]. We also are not interested in how ML improves the performance of MIP solvers, although it is a vast area of research that shows promising results [Khalil et al., 2016; Tang et al., 2020; Zhang et al., 2023].

2.1 Introduction of MIP to ML

One of the first ML problems to be ever explicitly formulated as a MIP is clustering [Vinod, 1969]. The task of clustering is typical because it contains

two important ingredients. It can be directly formulated mathematically, and there is no known effective and optimal algorithm. This, together with the option to formulate a custom objective function, makes it a good candidate for MIP. Already Kusiak [1984] proposed 5 different formulations of clustering. A commonly used MIP approach to general clustering was introduced by Grötschel and Wakabayashi [1989].

At that time, MIP solvers were not very practical for the applications to ML methods, which led to some skepticism regarding the usefulness of MIP in this area. It took many years and a lot of development on the hardware and software side of MIP solvers to achieve the state it is now. To paint a clearer picture, Bertsimas and Dunn computed the speedup factor of MIP solvers of *800 billion* between 1991 and 2015. That includes both software (1.4 million by itself) and hardware improvements [Bertsimas and Dunn, 2017].

This delay is further exemplified by the fact that the prestigious journal Annals of Statistics accepted the first use of MIP for ML in 2014 [Bertsimas and Mazumder, 2014]. In that work, Bertsimas and Mazumder show a formulation achieving high-quality solutions on the Least Quantile of Squares problem, even proving optimality for smaller instances in a reasonable time.

Since then, Rahul Mazumder (and Dimitris Bertsimas) have pioneered the research of MIP methods in ML. To describe a few more of their influential works, in 2016, they proposed a MIP approach to subset selection, where they tremendously improved on the state-of-the-art LASSO method [Bertsimas et al., 2016]. In 2017, Mazumder and Radchenko introduced a method for minimizing the number of non-zero coefficients of linear models. It was expressed as MIP formulation that outperformed other popular methods, even on non-trivial sizes. Later, Bandi et al. [2019] introduced a formulation for training Gaussian Mixture Models that outperforms the Expectation–maximization algorithm (EM) on multiple datasets. These works contributed significantly to the advent of MIP in ML techniques.

2.2 Current state of the art

We wish to briefly describe the current state of the use of MIP for ML. Since we do not aim for a comprehensive survey, the main focus will be on Decision trees and Artifical Neural networks (NNs). We look into these closely because further chapters reference and build upon the works presented. Lastly, we briefly overview some other applications to various fields of ML.

2.2.1 Decision trees

Creating an optimal decision tree is a problem from the NP-complete class [Hyafil and Rivest, 1976]. And since commonly used heuristic algorithms opt for speed rather than optimality, there is a place for frameworks like MIP to shine. There are multiple various formulations to choose from. There is work by Menickelly et al. [2016] that targets classification trees for data with strictly categorical features. Then there are more general classification tree formulations, such as the work of Verwer and Zhang [2017] or Optimal Classification Tree (OCT) by Bertsimas and Dunn [2017] that use standard real-valued thresholds. The OCT model will be central in Chapter 3. Additionally, the work of Verwer et al. [2017] formulates regression trees using MIP. These works also point out that one does not need to optimize just the standard misclassification error (or mean-squared in case of regression). They can incorporate more nuanced criteria into the optimization goal.

More recently, new formulations have improved the performance of MIP solvers in this area, for example, using Binary Programming [Verwer and Zhang, 2019], or formulations utilizing the max-flow problem [Aghaei et al., 2020, 2022]. Jo et al. [2023] create a decision tree with interpretability and fairness in mind.

There are many more formulations of Decision trees [e.g. Alston et al., 2022; Vos and Verwer, 2022; Blanco et al., 2022; Alès et al., 2022]. To briefly describe a few selected, D'Onofrio et al. [2023], inspired by Support vector machines (SVMs), add margins on decisions to the optimization. And Florio et al. [2022] expand from training decision trees to training decision diagrams.

Decision trees are an active research topic, especially when using MIP methods. It is due to their excellent interpretability, which is gaining importance in the current times.

2.2.2 Neural networks

Artifical Neural networks (NNs) are what comes to mind when ML is mentioned. Much research has come into perfecting the optimization of NNs, and attempts to use MIP in this regard are no exception [Simon and Takefuji, 1988; Fischetti and Jo, 2018; Thorbjarnarson and Yorke-Smith, 2021; Sildir and Aydin, 2022]. Some formulations also contain the capability to optimize the structure of the NN [Dua, 2010].

For example, Ferber et al. [2020] incorporate a MIP formulation into a layer of a Neural network.

Mostly, however, MIP is used as an additional tool rather than a tool for training the NNs directly. In areas where MIP finds its use, such as scheduling problems, the coalition of the speed of NNs and the global capabilities of MIP is being explored [Chen et al., 2023]. Or, for example, Zhu et al. [2020] create an approximation with the NN and use MIP to mend the approximate solution to a valid one.

Another way of using MIP in tandem with NNs is the formulation of trained NN as an integer program. This can be used to design new chemical compounds by having a NN and a desired property and use MIP to search for input vectors [Akutsu and Nagamochi, 2019]. A similar task is a search for adversarial examples, which is an important topic for the security of ML applications [Tjeng et al., 2017].

Because of the broad applicability of MIP on trained NNs, much research focuses on improving the formulation of the NN computation. E.g., Anderson et al. [2020] propose multiple formulation improvements to increase the speed of optimization, and König et al. [2022] use various MIP solver configurations for a similar purpose.

The last area of the use of MIP for NNs to mention is the quantization. Quantization is a process of moving a NN model from floating-point numbers to 8 (or less) bit integers, thus decreasing the size of the model and power consumption while increasing the speed of inference, thanks to the use of simpler computation. There are attempts to improve the performance of quantization with MIP, Hubara et al. [2020] suggest a formulation to optimize the number of bits to quantize to.

2.2.3 Further uses

There are many areas of ML. Providing a complete and extensive overview is not the aim of this chapter. To conclude the dense summary of the use of MIP in ML, we list a few more areas with MIP applications:

- *Rule lists* are very well-explainable models that can be optimized for various goals in mind using MIP [Rudin and Ertekin, 2018].

- *Linear models* are one of the simplest ML models. Even though there are usually better ways of training linear models, one can represent them using MIP as exemplified by Verwer et al. [2017], where they optimize for a more complex goal but use linear models re-formularized as MIP.
- To train robust *Support vector machine (SVM)*, Kurtz [2022] uses MIP to generate adversarial examples to improve the robustness of the resulting model.
- For general *Ensemble models*, Bastos et al. [2022] proposed a MIP formulation oriented on pruning a set of trained models to simplify the inference and possibly to improve the predictive capabilities of the ensemble.
- In *Boosting*, Kurtz [2022] propose an algorithm using a MIP formulation that outperforms classical AdaBoost on challenging datasets.
- The *Clustering* formulation of Grötschel and Wakabayashi [1989] has since been improved by Miyauchi et al. [2018], who reduced the time and memory requirements. Today, MIP is used for clustering in areas ranging from customer segmentation [Sağlam et al., 2006] to biology [Han et al., 2019]. MIP also enables us to solve more advanced clustering problems, such as (relaxed) correlation clustering [Figueiredo and Moura, 2013; Queiroga et al., 2021].
- *Bayesian Networks* are also challenging to optimize, which means there are multiple MIP applications to their optimization [e.g. Cussens, 2011; Bartlett and Cussens, 2015; Küçükyavuz et al., 2020].
- Similarly to Bayesian networks, *Markov decision processes* are being optimized using MIP [Albert, 2022].
- *Semi-supervised learning*, a wide area of methods, where at least a semi-supervised variant of SVM can be optimized using MIP [Demiriz and Bennett, 2001].
- *Feature selection* is an integral part of Machine Learning processes. While it can be expressed using MIP, it is often non-tractable to directly optimize [Bertolazzi et al., 2016]. Regardless, Maldonado et al. [2014] show that their formulation consistently outperforms other techniques.
- In *Dimensionality reduction*, the sparse principal component analysis aims to select at most k components to be non-zero. The requirement for some components to be zero, compared to standard Principal Component Analysis (PCA), enhances the interpretability of the reduction. The sparse PCA is NP-hard, which makes it a good fit for MIP [Dey et al., 2018].
- MIP can also introduce new models altogether. An example is the idea of *hyper-boxes* instead of hyperplanes and their use for multiclass classification [Üney and Türkay, 2006].

Seeing all of the provided examples, one can see that MIP is widely used in ML techniques. Many applications were formulated in the 20^{th} century. And as solvers are improving their performance, the formulations are becoming tractable to use for bigger and bigger instances. [Bertsimas and Dunn, 2017]

2.3 MIP in ML explainability

Regarding the Explainability of ML, MIP plays a significant role. For one, it can provide global optimality to well-interpretable models, such as Decision trees [Bertsimas and Dunn, 2017] or rule lists [Rudin and Ertekin, 2018]. It can also improve the explainability by performing optimized sparse PCA and reducing the number of relevant dimensions of a model [Dey et al., 2018], or find high-quality sparse linear models [Mazumder and Radchenko, 2017].

The above applications provide a kind of *global explanation*. Global explanations are called global in the sense that they explain the general (global) behavior of the model. On the opposite side are *local explanations*, which explain a single decision made by the model. An example would be a decision tree where decisions are based on a single variable compared to a convolutional NN returning a single channel bitmap, showing what pixels of the input image influenced the decision the most. Clearly, local explanations are less valuable. For example, Rudin [2019] calls for more focus on interpretability, which could be considered the same as global explainability.

Yet, some state-of-the-art models are so complex that explaining them globally via model extraction methods is not performed for accuracy reasons. For such models, one turns to local explanations that do not require a loss in accuracy [Bastani et al., 2019]. Commonly used local explanations are counterfactual explanations. And MIP can generate various high-quality counterfactual explanations [Chuang et al., 2023].

2.3.1 Counterfactual explanations

Counterfactual explanations are post-hoc explanations. They suggest what should change on the input to get the desired output. An example might be a lender denying a loan and providing a counterfactual explanation, such as "If your income was 1,000$ higher, your application would be accepted." These

are very valuable pieces of information to the users of ML systems. [Guidotti, 2022]

They are similar to adversarial examples and could be used in this way to find problems with the systems in place. We often want the counterfactual explanations to satisfy different desiderata. Guidotti [2022] comprised them in the following list:

- *Validity.* The counterfactual change should change the class accordingly.
- *Minimality.* It should not be possible to choose fewer changes than the counterfactual suggests to achieve the same class change.
- *Similarity.* The counterfactual change should not change the input in a significant way.
- *Plausibility.* The counterfactual change should keep the new input similar to other points in the dataset. For example, in the loan example, it should not say to the customer that they should be 250 years old.
- *Discriminative Power.* This means that the counterfactual should help to understand where the boundary between two classes is.
- *Actionability.* The provided changes should be possible to perform. For example, it should not suggest having different gender, race, etc.
- *Causality.* The counterfactuals should consistently keep the relationships between input features. Guidotti [2022] gives an example that the loan length and interest rate should be changed accordingly together, as is typical.
- *Diveristy.* If generating multiple counterfactuals, they should cover as many changes as possible with minimal overlap.

MIP formulations

There are many ways to obtain a counterfactual explanation, but only some use MIP [Guidotti, 2022]. We can use MIP to generate counterfactuals for NNs [Mohammadi et al., 2021], oblique decision trees (trees using hyperplanes for decisions) [Carreira-Perpiñán and Hada, 2021] and even ensembles [Parmentier and Vidal, 2021].

This comes as no surprise, since all these models can be formulated using MIP, and what remains is to select the right goal and encoding of the input

vector. Selecting the best goal is a well-researched topic. Ustun et al. [2019] introduced actionability-focused formulation, Russell [2019] focused on the diversity of the counterfactuals and the DACE model by Kanamori et al. [2020] improved their similarity by incorporating the training data distribution into the process. In their later work, Kanamori et al. [2021] introduce a new concept of ordering the changes to better suit the needs of the user.

Chapter 3

Fair explanations using classification trees

In this chapter, we dive into the area of global explanations. We will optimize a classification tree with concern for fairness regarding the provided explanations with the tree structure. This will be done directly by adding relevant constraints to the MIP formulation.

Classification trees are commonly considered to be well-interpretable. [e.g. Laber et al., 2023] That is because of their inherent structure, which is very logical and understandable. Trees can be broken down into simple logical formulae, one per leaf. Feldman [2000] showed that such formulae are understandable up to the size of around 5-9 literals, meaning trees must be constrained in depth to be understandable. Limited depth means higher classification error. And since the tree is trained to minimize the total misclassification error, it can mean that some leaves have accuracy close to random selection. This is because more homogeneous parts of the data are cropped away while the more challenging areas of the sample space remain in a single leaf. In those leaves, accuracy is very low. An example of this can be seen in Figure 3.1. The Classification and Regression Trees algorithm (CART) tree has a leaf with 57.1% accuracy.

We argue that if a sample is classified by a leaf, where the decision is supported by very low accuracy, the explanation is not fair with regard to other explanations supported by better accuracy. It makes the explanation less valid, if not potentially harmful. A user might incorrectly put a lot of importance on an explanation supported by low accuracy. When a leaf has low accuracy on training data, its decision can be flipped by introducing just a few samples of a different class.

Conventional algorithms, like CART [Breiman et al., 1984], often create trees with some leaves with low accuracy. This is usually not an issue because they optimize for the total accuracy of the model, and such low-accuracy leaves are averaged out. We claim that algorithms like CART, by performing splits of the highest decrease of gini impurity (or entropy), leave the most difficult parts of the tree for last. This, in the end, creates a node with low accuracy containing the "hard" samples. Notice, for example, the CART tree in Figure 3.1. It has one leaf with significantly worse accuracy compared to other leaves. The explanations provided by such a leaf are not very informative.

To address this issue, we aim to find a tree with the highest *leaf accuracy*. By leaf accuracy, we mean the least accuracy in a single leaf across all tree leaves. The MIP formulation of a tree with this goal in mind is what we call Fair Classification Tree (FCT). After creating this model, the total accuracy is improved by adding well-performing ML models to each leaf. This extension in the leaves creates the full *hybrid tree* (using the naming convention of Zhou and Chen [2002]). We call a model's leaf accuracy the accuracy of the least accurate leaf of the low-depth explainable tree. And we use the term hybrid-tree accuracy to refer to the total accuracy of the entire model with the extending models (the hybrid tree).

3.1 MIP formulation of FCT

Finding an optimal classification tree is known to be an NP-complete problem [Hyafil and Rivest, 1976]. As mentioned earlier, that is not an issue for MIP solvers since NP-hard problems can be transformed into MIP formulations.

We start with the OCT model introduced by Bertsimas and Dunn [2017] and extend it into a model that focuses on best interpretability. This means that each interpretation is supported by the highest possible accuracy. We will call that model FCT.

3.1.1 OCT formulation

Let us first explain the MIP formulation of OCT according to Bertsimas and Dunn [2017] in this section.

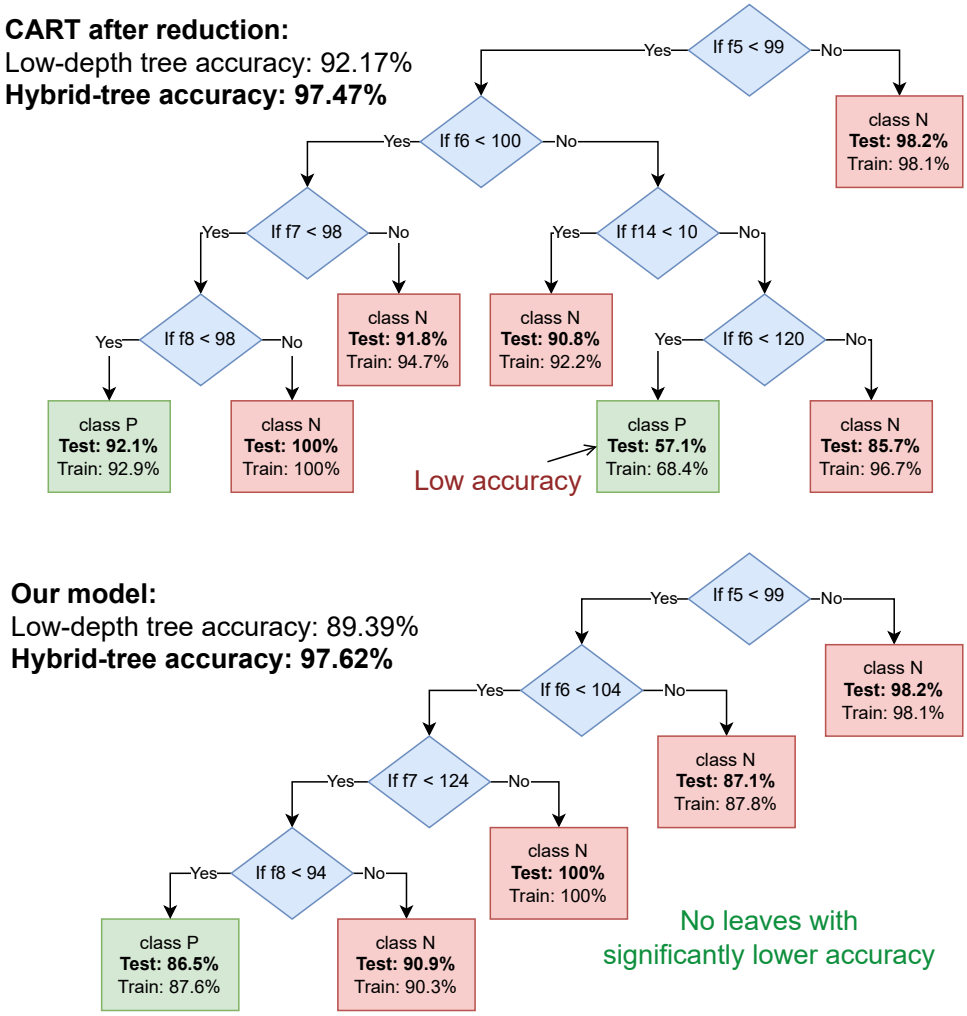


Figure 3.1: Comparison of a tree created by our method (FCT) and a tree on the same data, created by CART.

We consider a K classification problem, with n samples with p features each. We will refer to i-th sample as $\boldsymbol{x}_i = (x_{i1}, \ldots, x_{ip})$. Each $\boldsymbol{x}_i$ has a corresponding class y_i, which takes one value out of the K classes.

Basic structural variables

A classification tree consists of branching nodes and of leaf nodes. We call these sets of nodes $\mathcal{T}_B$ and $\mathcal{T}_L$, respectively. In branching nodes, we will model decision-making. With interpretability in mind, we consider so-called axis-aligned trees. That means every branching node t compares only the value of a single feature $x_{:j}$ and a threshold b_t. This is a more restrictive kind

of decision-making compared to oblique trees, where decision making based on a linear combination of all features.

This will be achieved using a set of binary variables a_{tj} for all branch nodes t and all features j. We restrict them in a way that allows a single feature to be activated for each node:

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

Together with selecting a variable, we optimize the threshold. This is represented by a set of variables b_t that take continuous value from the interval $[0, 1]$ because we consider our data to be normalized to this range.

We now have classification trees, where a branch node decision can now be conveniently written as $\boldsymbol{x}_i^\top \boldsymbol{a}_t \lesseqgtr b_t$. We further presume (in alignment with Bertsimas and Dunn [2017]) that in case of equality, the sample is assigned to the right branch. This means, that a sample x_i arrives to the left child of t if $\boldsymbol{x}_i^\top \boldsymbol{a}_t < b_t$ and to the right child if $\boldsymbol{x}_i^\top \boldsymbol{a}_t \geq b_t$.

Since strict inequalities cannot be used in MIP, the authors use the minimal difference between different consecutive values of each feature as a small epsilon that replaces the need for a strict inequality. It is precisely defined as follows:

$$\epsilon_j = \min \left\{ x_j^{(i+1)} - x_j^{(i)} \middle| x_j^{(i+1)} \neq x_j^{(i)}, \forall i \in \{1, \ldots, n-1\} \right\}$$

where $x_j^{(i)}$ is the i-th largest value in the j-th feature.

We also need to represent the assignment of a sample i to a leaf node t. This is done using a set of binary variables z_{it}. It holds that $z_{it} = 1 \iff$ x_i is assigned to a leaf t.

Putting all of these things together, we can see the two most important constraints which will ensure the correct function of the branching:

$$\boldsymbol{a}_m^\top \boldsymbol{x}_i \geq b_m - (1 - z_{it}) \tag{3.1}$$

$$\boldsymbol{a}_m^\top (\boldsymbol{x}_i + \boldsymbol{\epsilon}) \leq b_m + (1 + \epsilon_{\max})(1 - z_{it}) \tag{3.2}$$

where $i \in \{1, \ldots, n\}, t \in \mathcal{T}_L$ and $m \in \mathcal{T}_B$ is the node in which we decide to go right (3.1) or left (3.2). We will call these sets of ancestors of a node t with the node t to their left and right $A_L(t)$ and $A_R(t)$, respectively. These sets are disjoint, and their union is the set of all ancestors of a leaf t. $\epsilon_{\max}$ is

the highest value of ϵ_j over all j and serves as the best big-M bound. $\boldsymbol{\epsilon}$ is a p-dimensional vector consisting of ϵ_j.

We must further ensure that samples are assigned to exactly one leaf. This is done by the following constraint:

$$\sum_{t \in \mathcal{T}_L} z_{it} = 1$$

Extra structural variables

We want to allow the solver to ignore some nodes. Though we represent the complete binary tree, we do not require to get a complete tree as a result. Thus, we add binary variables to signal if a specific node is used or not.

For branching nodes, these will be variables d_t. We enforce their property by constraining $\boldsymbol{a}_t$ and b_t to be equal to zero if $d_t = 0$. This should lead to all samples being sent to the right subtree according to Bertsimas and Dunn [2017], but it introduced a flaw in the formulation, as will be discussed later in Section 3.1.1.

$$\sum_{j=1}^{p} a_{jt} = d_t, \quad \forall t \in \mathcal{T}_B \tag{3.3}$$

$$b_t \leq d_t, \quad \forall t \in \mathcal{T}_B \tag{3.4}$$

We also ensure that child nodes of an unused node are also unused with the following constraint:

$$d_t \leq d_{p(t)}, \quad \forall t \in \mathcal{T}_B \backslash \{\text{root}\} \tag{3.5}$$

where $p(t)$ is the parent node of node t. The variable d_t also enables us to use the number of nodes as an optimization parameter.

Regarding a similar variable for leaf nodes, we introduce binary variables l_t that reflect the emptiness of a leaf t.

$$l_t = \begin{cases} 1 & \text{if } \sum_{i=1}^{n} z_{it} > 0 \\ 0 & \text{if } \sum_{i=1}^{n} z_{it} = 0 \end{cases}$$

This is done using the z_{it} variables in the following set of constraints:

$$z_{it} \leq l_t, \quad \forall i \in \{1, \ldots, n\}, \quad \forall t \in \mathcal{T}_L \tag{3.6}$$

We now use this variable l_t to introduce a limit on the minimal amount of samples assigned to each leaf.

$$\sum_{i=1}^{n} z_{it} \geq N_{\min} l_t, \quad \forall t \in \mathcal{T}_L \tag{3.7}$$

where $N_{\min}$ is the minimal amount of samples assigned to a leaf.

To enable the computation of the misclassification error, we must know the classification of each sample. First, we aggregate the number of samples in a leaf t. Bertsimas and Dunn name this N_t and write the following constraints:

$$N_t = \sum_{i=1}^{n} z_{it}, \quad \forall t \in \mathcal{T}_L \tag{3.8}$$

Then we aggregate the number of samples of a certain class k in a leaf t. Bertsimas and Dunn call this variable N_{kt}.

$$N_{kt} = \sum_{i=1}^{n} Y_{ik} z_{it}, \quad \forall k \in \{1, \ldots, K\}, \quad \forall t \in \mathcal{T}_L \tag{3.9}$$

where Y_{ik} is equal to 1 if $y_i = k$. Equivalently:

$$Y_{ik} = \begin{cases} 1 & \text{if } y_i = k \\ 0 & \text{if } y_i \neq k \end{cases} \quad \forall i \in \{1, \ldots, n\}, \quad \forall k \in \{1, \ldots, K\} \tag{3.10}$$

The formulation of the constraint (3.9) and the definition of Y (3.10) are different from the original formulation by Bertsimas and Dunn [2017] from which we otherwise did not diverge. This is just a simplification of the original. Both are mathematically equivalent.

Classification variables

Lastly, we introduce a variable representing the class assigned to a leaf. This is done by a set of binary variables c_{kt}, which can be interpreted as true if leaf t is assigned a class k. We ensure that only one class is assigned and that no class is assigned to an empty leaf.

$$\sum_{k=1}^{K} c_{kt} = l_t, \quad \forall t \in \mathcal{T}_L \tag{3.11}$$

Finally, we obtain information about the number of misclassified samples or misclassification loss. Bertsimas and Dunn name the new variable L_t. We need it to take the value of the minimal difference between the total number of samples in a leaf N_t and the number of samples of any class. The class satisfying the minimal difference will be the selected class k for which $c_{kt} = 1$. This is achieved using these constraints:

$$L_t \geq N_t - N_{kt} - n(1 - c_{kt}), \qquad \forall k \in \{1, \ldots, K\}, \quad \forall t \in \mathcal{T}_L \tag{3.12}$$

$$L_t \leq N_t - N_{kt} - nc_{kt}, \qquad \forall k \in \{1, \ldots, K\}, \quad \forall t \in \mathcal{T}_L \tag{3.13}$$

$$L_t \geq 0, \qquad \forall t \in \mathcal{T}_L \tag{3.14}$$

where n is the size of the training set and serves as a sufficiently high big-M constant. The constraints (3.14) are important for empty leaves. Otherwise, we might get arbitrarily low loss using a single empty leaf.

We conclude this introduction of OCT model by formulating the objective function:

$$\min \frac{1}{\hat{L}} \sum_{t \in \mathcal{T}_L} L_t + \alpha \sum_{t \in \mathcal{T}_B} d_t \tag{3.15}$$

Where $\hat{L}$ is the loss obtained by a simple majority classifier, essentially n minus the number of samples of the most represented class. It is used to "normalize" the first summand of the optimized function. This makes the coefficient α independent of dataset size. [Bertsimas and Dunn, 2017]

Formulation flaw

Until now, the described formulation was the work of Bertsimas and Dunn [2017] with only one constraint slightly changed for better understanding. However, after implementing this formulation, we uncovered a flaw in the formulation. It is trivially solved with 0% error by $d_m = 0, \forall m \in \mathcal{T}_B$.

Consider the following. A sample x_i is assigned to a leaf t. This means, that $z_{it} = 1$, and thus branch decision constraints (3.1) and (3.2) simplify to $\boldsymbol{a}_m^\mathsf{T} \boldsymbol{x}_i \geq b_m$ and $\boldsymbol{a}_m^\mathsf{T}(\boldsymbol{x}_i + \boldsymbol{\epsilon}) \leq b_m$ respectively, for any ancestor m for which it holds that $d_m = 0$. There is nothing in the way of trivially satisfying both of these with $\boldsymbol{a}_m = \mathbf{0}$ and $b_m = 0$. Solver thus sets all d_m to 0 to enable all $\boldsymbol{a}_m = \mathbf{0}$ and proceeds by assigning all samples of each class to a single leaf, achieving seemingly 100% accuracy.

A proposed solution to this issue is to change the decision constraint from

(3.2) to

$$\begin{aligned} \boldsymbol{a}_m^\top(\boldsymbol{x}_i + \bar{\boldsymbol{\epsilon}}) + \epsilon_{\min} \leq b_m + (1 + \epsilon_{\max})(1 - z_{it}), \\ \forall i \in \{1, \ldots, n\}, \quad \forall t \in \mathcal{T}_L, \quad \forall m \in A_L(t) \end{aligned} \tag{3.16}$$

where $\epsilon_{\min}$ is the lowest value of ϵ_j and $\bar{\epsilon}_j = \epsilon_j - \epsilon_{\min}$. $\bar{\boldsymbol{\epsilon}}$ is essentially the same vector as $\boldsymbol{\epsilon}$ but with all values decreased by $\epsilon_{\min}$.

This constraint is not satisfiable if $d_m = 0$ and any leaf t such that $m \in A_L(t)$ is assigned any data sample ($z_{it} = 1$). This holds because all $\epsilon_j > 0$ and thus also $\epsilon_{\min} > 0$. This results in the grouping of samples in the rightmost leaf of a subtree with m as a root. This is what Bertsimas and Dunn [2017] claimed should happen.

This change in the formulation will not influence the results otherwise, since if $d_m = 1$, there is $a_{mj} = 1$ for exactly one j, and removed and added $\epsilon_{\min}$ will cancel out.

Although this is a valid correction of the issue, we were not interested in optimizing the number of decision tree nodes, so we went for a more straightforward solution to this issue and discarded the d_t variable altogether in further extending formulations. In all places where d_t was present, we swapped it for 1. By this, we force all decision nodes to branch. To include non-complete trees, we prune them after MIP optimization.

Complete model

Now that we have described all parts of the OCT formulation, the complete formulation is in Figure (3.2).

In our tests using the OCT formulation, we did not use the number of branch nodes. We used the model with the original constraint (3.2) for the decision bound to the left child and replaced the variable d_t with 1 and $\alpha = 0$ for the purposes of optimization.

3.1.2 Straightforward leaf accuracy formulation

The model described in the previous section is the basis for our work. The initial implementation of our desideratum consisted of a simple hard constraint

$$
\begin{aligned}
\min\ & \frac{1}{\hat{L}} \sum_{t \in \mathcal{T}_L} L_t + \alpha \sum_{t \in \mathcal{T}_B} d_t \\
\text{s. t. } & L_t \geq N_t - N_{kt} - n(1 - c_{kt}) && \forall k \in \{1, \dots, K\}, \quad \forall t \in \mathcal{T}_L \\
& L_t \leq N_t - N_{kt} - n c_{kt} && \forall k \in \{1, \dots, K\}, \quad \forall t \in \mathcal{T}_L \\
& L_t \geq 0 && \forall t \in \mathcal{T}_L \\
& N_{kt} = \sum_{i=1}^{n} Y_{ik} z_{it} && \forall k \in \{1, \dots, K\}, \quad \forall t \in \mathcal{T}_L \\
& N_t = \sum_{i=1}^{n} z_{it} && \forall t \in \mathcal{T}_L \\
& l_t = \sum_{k=1}^{K} c_{kt} && \forall t \in \mathcal{T}_L \\
& \boldsymbol{a}_m^\mathsf{T} \boldsymbol{x}_i \geq b_m - (1 - z_{it}) && \forall i \in \{1, \dots, n\}, \forall t \in \mathcal{T}_L, \forall m \in A_R(t) \\
& \boldsymbol{a}_m^\mathsf{T} (\boldsymbol{x}_i + \bar{\boldsymbol{\epsilon}}) + \epsilon_{\min} \leq \\
& \qquad b_m + (1 + \epsilon_{\max})(1 - z_{it}) && \forall i \in \{1, \dots, n\}, \forall t \in \mathcal{T}_L, \forall m \in A_L(t) \\
& \sum_{t \in \mathcal{T}_L} z_{it} = 1 && \forall i \in \{1, \dots, n\} \\
& z_{it} \leq l_t && \forall i \in \{1, \dots, n\}, \quad \forall t \in \mathcal{T}_L \\
& \sum_{i=1}^{n} z_{it} \geq N_{\min} l_t && \forall t \in \mathcal{T}_L \\
& \sum_{j=1}^{p} a_{jt} = d_t && \forall t \in \mathcal{T}_B \\
& 0 \leq b_t \leq d_t && \forall t \in \mathcal{T}_B \\
& d_t \leq d_{p(t)} && \forall t \in \mathcal{T}_B \backslash \{\text{root}\} \\
& z_{it}, l_t \in \{0, 1\} && \forall i \in \{1, \dots, n\}, \quad \forall t \in \mathcal{T}_L \\
& a_{jt}, d_t \in \{0, 1\} && \forall j \in \{1, \dots, p\}, \quad \forall t \in \mathcal{T}_B \\
& c_{kt} \in \{0, 1\} && \forall k \in \{1, \dots, K\}, \quad \forall t \in \mathcal{T}_L
\end{aligned}
$$

Figure 3.2: The OCT formulation. For a detailed explanation, see the explanation in Section 3.1.1 or the original paper itself [Bertsimas and Dunn, 2017]. There is a minor correction of the original in the red constraint. It is changed according to the formulation issue discussed in Section 3.1.1.

on leaf accuracy.

$$L_t \leq N_t(1 - A^h), \quad \forall t \in \mathcal{T}_L \tag{3.17}$$

where A^h is the desired accuracy in leaves.

The model with added constraints (3.17) is infeasible for some A^h. Take, for example, a tree of depth 1 and a training set with a single feature, where $x = [1, 2, 3]$ and $y = [1, 0, 1]$. For $A^h > \frac{2}{3}$, the model is infeasible.

Since we do not really need to minimize the misclassification error, we also tested a version where we omit the optimization goal and focus solely on feasibility.

This constrained model was used in a kind of halving algorithm or bisection. We search for the maximal A^h for which the model is feasible. Trivially, we know that for $A^h = \frac{1}{K}$, the model will always be satisfiable because we can always assign the most common class in each leaf. Similarly, we know that the upper bound on accuracy is, by definition, 1. We then halve the interval, looking for the boundary between feasible and infeasible models. See Algorithm 1 for exact pseudocode.

Algorithm 1: Halving algorithm

```
Input: K                                        // Number of classes
Input: p                          // Desired precision of the result
Output: A^{h*}                      // Best achievable leaf accuracy
L ← 1/K
U ← 1
while U − L > p do
    M ← (L+U)/2
    if MIP formulation with A^h = M is feasible then
        L ← M
    else
        U ← M
return L
```

Needless to say, this is not the best way of using MIP. The solver takes a long time even to decide on feasibility. In practice, this meant we gave the solver around one hour to decide whether a model was feasible and assumed infeasibility otherwise.

Even like this, the algorithm takes several hours to get to an accurate enough estimate. Assume we want to be precise up to a tenth of a percent, equivalent to an accuracy of 0.001. Assuming we perform binary classification,

it takes 9 runs of the solver to obtain the result. That means 9 hours if we use an objective function and a bit less if we don't. The way to compute in general the number of iterations is the following:

$$n_{\text{runs}} = \left\lceil \log\left(\frac{1-\frac{1}{K}}{p}\right)\right\rceil \tag{3.18}$$

The algorithm's time complexity means shortening the time given to a single MIP formulation run, which meant worse results. That was naturally our next focus.

Soft accuracy formulation

One interesting way of improving the potential of the solver to find good solutions was the proposal to set up a softer constraint on leaf accuracy. It consisted of relaxed demand on accuracy in leaves with fewer assigned training samples. We set a threshold H^{thresh} as the number of samples in a leaf for which to use the standard accuracy (we refer to it as *hard accuracy*). When there are fewer samples than H^{thresh}, soft accuracy is used instead. Hard accuracy A_t^h in a leaf t is computed as follows:

$$A_t^h = \frac{N_{kt}}{N_t}, \quad k \in \{1,\ldots,K\} \text{ s.t. } c_{kt} = 1$$

And the soft accuracy A_t^s is computed like this:

$$A_t^s = \frac{N_{kt}}{H^{\text{thresh}}}, \quad k \in \{1,\ldots,K\} \text{ s.t. } c_{kt} = 1$$

Considering the A^h as a parameter of the model, the soft accuracy formulation is achieved by comparing the number of misclassified samples L_t to the maximal number of misclassified samples $L_{\text{thresh}} = H^{\text{thresh}}(1 - A^h)$.

Now let us introduce the MIP formulation. Firstly, we need a binary variable h_t that holds true if hard accuracy is used in a node t.

$$h_t = \begin{cases} 1 & \text{if } N_t \geq H^{\text{thresh}} \\ 0 & \text{if } N_t < H^{\text{thresh}} \end{cases}$$

We force this behavior using the constraints:

$$h_t \leq \frac{N_t}{H^{\text{thresh}}} \qquad \forall t \in \mathcal{T}_L \tag{3.19}$$

$$h_t \geq \frac{N_t - H^{\text{thresh}} + 1}{n} \qquad \forall t \in \mathcal{T}_L \tag{3.20}$$

The constraints (3.19) sets the h_t to 0 when the number of samples in the leaf is less than a threshold, and the constraints (3.20) pushes it to 1 for values above the threshold, while normalizing by n to keep the value below 1. The $+1$ in the numerator ensures that for $N_t = H^{\text{thresh}}$ the value h_t is set to 1.

Having the h_t variable set, we can continue by replacing the simple constraint on accuracy (3.17) with the following constraints:

$$L_t \leq H^{\text{thresh}}(1 - A^h) + nh_t \qquad \forall t \in \mathcal{T}_L \qquad (3.21)$$

$$L_t \leq N_t(1 - A^h) + n(1 - h_t) \qquad \forall t \in \mathcal{T}_L \qquad (3.22)$$

where n is used as a big-M constant. We can see the resemblance to the original formulation (3.17) that now splits into two that are made tighter or looser, based on the value of h_t.

This was a semi-successful approach as it improved the quality of the solutions. Still, it did not solve the issue of time complexity and general oddity of the bisection approach.

3.1.3 Optimizable formulation

The clear next step was to create a formulation that could optimize leaf accuracy in the MIP solver.

This is non-trivial because accuracy is defined as the number of samples with the correct class divided by the number of all samples. Both of these values are represented using variables in our formulation, so if we were to set accuracy as a variable to optimize directly, we would not be able to linearize the equation. In OCT formulation, the total number of samples is a parameter, so it is easier to state the optimization goal.

To obtain accuracy in a different way, we introduce concepts of accuracy potential and accuracy contribution. Accuracy potential s_{it} is the accuracy that one sample x_i can contribute to a leaf t. Looking at the value for a single leaf t, it is equal to 0 for samples not assigned to the leaf t and has the same value as all other samples assigned to the leaf t. It also sums to 1 for every node t. This is the potential amount of accuracy it can contribute to the leaf accuracy of t.

Accuracy contribution S_{it} is the accuracy a sample x_i contributes to the

accuracy of leaf t. It equals s_{it} if the leaf t classifies x_i correctly. In terms of the formulation, if $c_{kt} = 1$ for $k = y_i$.

To describe the MIP formulation of accuracy potential s_{it}, we first state the requirements:

1. must be between 0 and 1 inclusive
2. must sum to 1 for each leaf t to represent a 100% leaf accuracy
3. must be 0 for samples unassigned to node t
4. must be equal for all samples assigned to the node t

We satisfy the first three requirements simply with the following set of constraints:

$$0 \leq s_{it} \leq 1 \qquad \forall i \in \{1, \ldots, n\}, \quad \forall t \in \mathcal{T}_L \tag{3.23}$$

$$\sum_{i=1}^{n} s_{it} = l_t \qquad \forall t \in \mathcal{T}_L \tag{3.24}$$

$$s_{it} \leq z_{it} \qquad \forall i \in \{1, \ldots, n\}, \quad \forall t \in \mathcal{T}_L \tag{3.25}$$

We sum them to l_t instead of 1 to ensure the possibility of having empty leaves. Since empty leaves have no assigned samples, all s_{it} for such leaves t would be equal to zero, and their sum could never be equal to 1.

The last requirement is the tricky one. We started with the following straightforward formulation:

$$\begin{aligned} s_{it} &\leq s_{jt} + (2 - z_{it} - z_{jt}) \quad \forall i, j \in \{1, \ldots, n\}, \quad \forall t \in \mathcal{T}_L \\ s_{it} &\geq s_{jt} + (z_{it} + z_{jt} - 2) \quad \forall i, j \in \{1, \ldots, n\}, \quad \forall t \in \mathcal{T}_L \end{aligned} \tag{3.26}$$

Which suffers from the fact that it contains $O(n^2)$ constraints, which increases the total complexity from linear to quadratic with respect to the size of training data.

To keep the size complexity linear for FCT, just as is the case with OCT, we introduce a reference potential r_t for each leaf t. This reference will serve as the common value to which all samples assigned will relate their values. We replace constraints (3.26) with this formulation:

$$\begin{aligned} r_t &\leq s_{it} + (1 - z_{it}), \quad \forall i \in \{1, \ldots, n\}, \quad \forall t \in \mathcal{T}_L \\ r_t &\geq s_{it} + (z_{it} - 1), \quad \forall i \in \{1, \ldots, n\}, \quad \forall t \in \mathcal{T}_L \end{aligned} \tag{3.27}$$

Which is linear in size depending on n. We finally satisfied all four requirements of the accuracy potential effectively.

As a next step, we define accuracy contribution S_{it} as accuracy that a sample contributes to the leaf accuracy t. Its value will be either equal to s_{it} or 0 depending on whether the sample is of the class assigned to the leaf or not. The constraints to ensure that are the following:

$$0 \leq S_{it} \leq s_{it} \qquad \forall i \in \{1, \ldots, n\}, \quad \forall t \in \mathcal{T}_L \tag{3.28}$$

$$S_{it} \leq \sum_{k=1}^{K} Y_{ik} c_{kt} \qquad \forall i \in \{1, \ldots, n\}, \quad \forall t \in \mathcal{T}_L \tag{3.29}$$

$$S_{it} \geq s_{it} + \sum_{k=1}^{K} Y_{ik} c_{kt} - 1 \qquad \forall i \in \{1, \ldots, n\}, \quad \forall t \in \mathcal{T}_L \tag{3.30}$$

where the $\sum_{k=1}^{K} Y_{ik} c_{kt}$ will always evaluate to $c_{y_i t}$ which is the binary value signalling whether node t is assigned the class of the sample $\boldsymbol{x}_i$, that is y_i. If so, constraint (3.29) will allow for higher than 0 values, and constraint (3.30) becomes a tight lower bound on S_{it}.

What remains is to sum the accuracy contributions and obtain the minimal leaf accuracy of all leaves. We call the variable Q. The set of constraints ensuring that we have the minimum is the following:

$$Q \leq \sum_{i=1}^{n} S_{it} + (1 - l_t), \quad \forall t \in \mathcal{T}_L \tag{3.31}$$

where the $(1 - l_t)$ summand corrects the issue of empty leaves having accuracy 0 in the formulation by setting this constraint to 1 in order for them to not influence the result. Since all samples must be assigned to some leaf, it cannot happen that all leaves are empty, and the perceived leaf accuracy would go to 1.

We do not need to give a lower bounding constraint to Q because it will be the sole part of our maximization objective:

$$\max \; Q \tag{3.32}$$

This ensures that Q will be equal to the maximal possible value, which is the leaf accuracy value, thanks to the constraints (3.31).

Soft accuracy constraints

Since we used soft constraints in the previous formulation variant, we implemented them here as well. We use the same constraints (3.19) and (3.20) for variable h_t. And for the soft accuracy itself, we replace the desired accuracy A^h with our variable Q in the constraints (3.21):

$$L_t \leq H^{\text{thresh}}(1 - Q) + nh_t, \quad \forall t \in \mathcal{T}_L \tag{3.33}$$

the change is highlighted in blue. These constraints remain linear since H^{thresh} is a parameter of the formulation.

We further add another conditional to the constraints (3.31) on leaf accuracy Q to make them trivially satisfied for leaves that use the soft accuracy:

$$Q \leq \sum_{i=1}^{n} S_{it} + (1 - l_t) + (1 - h_t), \quad \forall t \in \mathcal{T}_L \tag{3.34}$$

the changes are again highlighted in blue.

This addition to the formulation was also used for some testing, as will be presented in Section 3.4. Otherwise, our formulation reached its final form, as there is not much more to be done to improve it. We optimize directly the criterion we want in one go, and the formulation has not asymptotically increased in size, compared to OCT.

3.1.4 Final FCT formulation

The version of the formulation that has shown the best results is the variant that directly maximizes the leaf accuracy. The variant using soft accuracy has had worse results compared to the simpler version. More on these findings in Section 3.4.

We present a complete formulation of the FCT model in Figure 3.3. It is the most successful variant proposed. The implementation of tree functioning is taken from the OCT formulation by Bertsimas and Dunn [2017]. Those parts are in black. We do not use the d_t variable, so the formulations representing the branching nodes are the same as in the OCT model. And where d_t was used originally, it is replaced by 1. The changes are in blue.

	Symbol	Explanation	Size
Params	Y_{ik}	Equal 1 for true class of a sample	$n \times K$
	$\boldsymbol{x}_i$	Input samples	$n \times p$
	$\boldsymbol{\epsilon}$	Minimal change in feature values	p
	$\epsilon_{\max}$	Maximal value of $\boldsymbol{\epsilon}$	1
	$N_{\min}$	Minimum of samples in a leaf	1
	$\mathcal{T}_L$	Set of leaf nodes	2^d
	$\mathcal{T}_B$	Set of decision (branching) nodes	$2^d - 1$
	$A_L(t)$	Ancestors of leaf t that decide left	$\leq d - 1$
	$A_R(t)$	Ancestors of leaf t that decide right	$\leq d - 1$
Variables	Q	Tree's leaf accuracy	1
	s_{it}	Accuracy potential of $\boldsymbol{x}_i$ in leaf t	$n \times 2^d$
	S_{it}	Accuracy contribution of $\boldsymbol{x}_i$ in leaf t	$n \times 2^d$
	r_t	Reference accuracy for $s_{:t}$	2^d
	z_{it}	Assignment of $\boldsymbol{x}_i$ to leaf t	$n \times 2^d$
	l_t	Non-emptiness of leaf t	2^d
	c_{kt}	Assignment of class k to leaf t	$K \times 2^d$
	a_{jt}	1 if deciding on feature j in node t	$p \times \lvert\mathcal{T}_B\rvert$
	b_t	Decision threshold in node t	$2^d - 1$

Table 3.1: Description of MIP symbols used in the FCT formulation in Figure 3.3. Parameter n refers to the number of samples, K is the number of classes, p is the number of features, and d is the depth of the tree.

Additionally removed from the OCT formulation were variables (and their respective constraints) regarding the misclassification loss L_t, number of samples in a leaf N_t, and the number of samples of a particular class in a leaf N_{kt}.

In their place, we added variables and constraints regarding accuracy potential s_{it}, accuracy contribution S_{it}, and Q, the leaf accuracy itself. All completely new constraints and formulations are in purple.

3.2 Creating the Hybrid trees

Having optimized the tree using MIP, we will continue by doing two things. The first one is the reduction of the tree. It is a simple transformation of the tree to an equivalent form in terms of total accuracy. It usually removes some leaves since the MIP solving process creates a complete binary tree, and some are often redundant.

$$
\begin{aligned}
&\max\ Q \\
&\text{s. t. } Q \le \sum_{i=1}^{n} S_{it} + (1 - l_t) && \forall t \in \mathcal{T}_L \\
&\quad s_{it} \le z_{it} && \forall i \in \{1,\dots,n\}, \quad \forall t \in \mathcal{T}_L \\
&\quad r_t \le s_{it} + (1 - z_{it}) && \forall i \in \{1,\dots,n\}, \quad \forall t \in \mathcal{T}_L \\
&\quad r_t \ge s_{it} + (z_{it} - 1) && \forall i \in \{1,\dots,n\}, \quad \forall t \in \mathcal{T}_L \\
&\quad S_{it} \le s_{it} && \forall i \in \{1,\dots,n\}, \quad \forall t \in \mathcal{T}_L \\
&\quad S_{it} \le \sum_{k=1}^{K} Y_{ik} c_{kt} && \forall i \in \{1,\dots,n\}, \quad \forall t \in \mathcal{T}_L \\
&\quad S_{it} \ge s_{it} + \sum_{k=1}^{K} Y_{ik} c_{kt} - 1 && \forall i \in \{1,\dots,n\}, \quad \forall t \in \mathcal{T}_L \\
&\quad l_t = \sum_{i=1}^{n} s_{it} && \forall t \in \mathcal{T}_L \\
&\quad l_t = \sum_{k=1}^{K} c_{kt} && \forall t \in \mathcal{T}_L \\
&\quad \boldsymbol{a}_m^{\mathsf{T}} \boldsymbol{x}_i \ge b_m - (1 - z_{it}) && \forall i \in \{1,\dots,n\}, \forall t \in \mathcal{T}_L, \forall m \in A_R(t) \\
&\quad \boldsymbol{a}_m^{\mathsf{T}} (\boldsymbol{x}_i + \boldsymbol{\epsilon}) \le \\
&\qquad b_m + (1 + \epsilon_{\max})(1 - z_{it}) && \forall i \in \{1,\dots,n\}, \forall t \in \mathcal{T}_L, \forall m \in A_L(t) \\
&\quad \sum_{t \in \mathcal{T}_L} z_{it} = 1 && \forall i \in \{1,\dots,n\} \\
&\quad z_{it} \le l_t && \forall i \in \{1,\dots,n\}, \quad \forall t \in \mathcal{T}_L \\
&\quad \sum_{i=1}^{n} z_{it} \ge N_{\min} l_t && \forall t \in \mathcal{T}_L \\
&\quad \sum_{j=1}^{p} a_{jt} = 1 && \forall t \in \mathcal{T}_B \\
&\quad 0 \le b_t \le 1 && \forall t \in \mathcal{T}_B \\
&\quad z_{it}, l_t \in \{0,1\} && \forall i \in \{1,\dots,n\}, \quad \forall t \in \mathcal{T}_L \\
&\quad a_{jt} \in \{0,1\} && \forall j \in \{1,\dots,p\}, \quad \forall t \in \mathcal{T}_B \\
&\quad c_{kt} \in \{0,1\} && \forall k \in \{1,\dots,K\}, \quad \forall t \in \mathcal{T}_L \\
&\quad 0 \le Q, r_t, S_{it}, s_{it} \le 1 && \forall i \in \{1,\dots,n\}, \quad \forall t \in \mathcal{T}_L
\end{aligned}
$$

Figure 3.3: Full FCT formulation. What each constraint represents has been explained extensively in Section 3.1. In black are the original constraints of OCT by Bertsimas and Dunn [2017], with minor edits in blue. In purple are our additions to the formulation that complete the FCT. For explanations of the symbols, refer to Table 3.1 or descriptions above.

The second step is then the extension. We take data assigned to every leaf and train new models on those samples, creating the hybrid tree. This improves the model's accuracy to levels comparable to using accuracy-oriented models. The slight difference in accuracy is lost to improve the fairness of the explanations provided by the tree.

3.2.1 Tree reduction

The reduction of the tree is done by iterative removal of leaves. We combine all sibling leaves assigned the same class and remove all empty leaves. This is done recursively until no further leaves can be removed.

This process decreases the number of splits in the tree and increases the number of samples in leaves, giving more capability to learn good extending models. While not reducing the overall accuracy of the tree, it can also improve the leaf accuracy since we are interested only in the minimum accuracy.

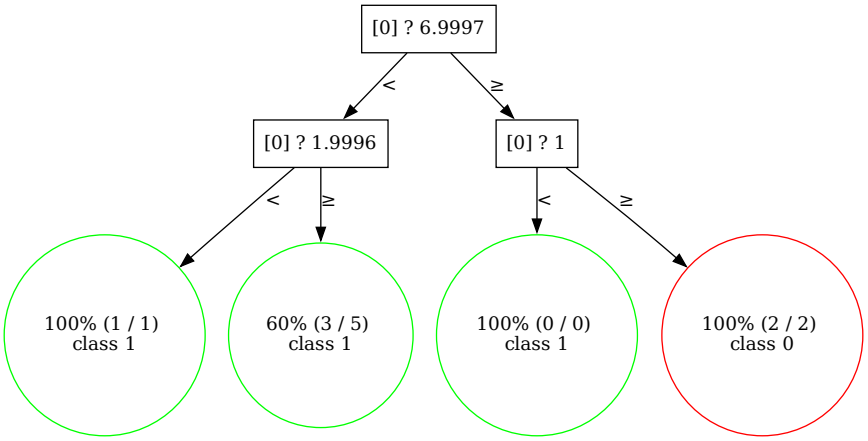
To illustrate this step, see the example in Figure 3.4.

3.2.2 Leaves extension

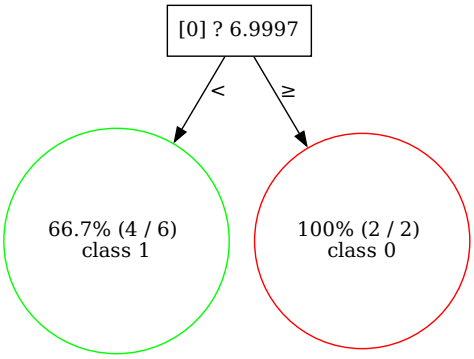
After the FCT is reduced to the simplest yet equivalent form, it is time to extend it. We go through all leaves and use the training data assigned to each leaf to train a further model.

For our purposes, we used the XGBoost [Chen and Guestrin, 2016] model that has been shown to perform the best on mid-sized datasets [Grinsztajn et al., 2022].

To optimize the XGBoost models, we do a Bayesian search for 50 iterations to search for the most suitable hyperparameters of the model. The used distributions are in Appendix C.2. Every configuration is trained on 3-fold cross-validation on the corresponding data. The best hyperparameters from the search are then used to train the leaf-extending model on all assigned training data. If a leaf has 100% accuracy, we cannot train anything and keep the class assigned to the leaf. If a leaf does not have enough samples of one



(a) : Tree before the reduction.



(b) : The same tree after the reduction

Figure 3.4: Example of Tree reduction. This is an example tree, trained using CART on a dummy dataset of 8 samples. The tree before reduction is extended to be a complete tree of depth 2. The percentages in each leaf represent accuracy on the set of 8 samples. We can see both types of reductions. Notice that the reduction increased the leaf accuracy of the tree by 6.7 percentage points and simplified the learned explanations.

class to perform the cross-validation, such a leaf is finished by an extending tree of depth 5.

3.3 Experiments

Finally, we take a look at the experiments. The majority of FCT models were trained with a depth limit equal to 4. The trees were then reduced and extended by XGBoost to create the final Hybrid tree structure and to be more comparable to the accuracy optimizing methods.

For MIP solving, we used the Gurobi optimizer [Gurobi Optimization, LLC, 2023], and for XGBoost training, we used the scikit-learn interface of the model's library [Chen and Guestrin, 2016]. For the hyperparameter search of the XGBoost models in leaves, we used the scikit-optimize interface [Head et al., 2021] of the Bayesian search with 3-fold cross-validation. The search was performed for 50 iterations, and after the best hyperparameters were found, they were used to train a model on all training data assigned to the leaf.

We compare to CART algorithm, which is a widely used heuristic algorithm. We used scikit-learn [Pedregosa et al., 2011] implementation. To optimize its hyperparameters, we again used scikit-optimize interface of the Bayesian search. We perform 100 iterations using 5-fold cross-validation. Specific details of distributions are listed in Appendix C.2.

We also compare our method to OCT [Bertsimas and Dunn, 2017] for which we used our implementation, provided with the implementation of FCT.

FCT hyperparameters. We use one main configuration of hyperparameters and then compare it to configurations with minor modifications to express the influence of the hyperparameters. The main configuration of hyperparameters of FCT is the following:

- The maximal depth of the tree is 4.
- The minimal number of samples in a leaf is 50.

- The Gurobi solver is parametrized to focus on heuristic improvements. The Heuristics parameter is equal to 0.8, and the MIPFocus parameter is 1. This means the solver prefers looking for new feasible solutions more than on proving the optimality.

3.3.1 Training modes

Even though the FCT formulation is complete, it still allows for different paths leading to the optimum. Because the entire formulation takes a lot of time to solve optimally for trees of depth greater than 2, we tested three training modes. Other than direct optimization of the MIP formulation from scratch (will be referred to as *Direct*), we propose two different configurations.

One uses a solution from CART algorithm as a starting point of the optimization process. This is done using scikit-learn [Pedregosa et al., 2011] implementation, without any hyperparameter tuning. The hyperparameters are either default or the same as will be for the FCT formulation. Those are a limit on depth (d) and a minimal amount of samples in a single leaf ($N_{\min}$). This was the main method of optimization; we refer to it as *Warmstarted*.

The third variant works iteratively by increasing the depth of the tree and warmstarting the models with the solutions of the shallower trees. We first optimize the tree of depth 1, then use the solution as a starting point of the tree with depth 2, and so on, until we reach the final depth. It will be referred to as *Gradual* since the tree is gradually increasing its depth.

3.3.2 Datasets used

For experiments, we used a benchmark of tabular mid-size datasets created by Grinsztajn et al. [2022]. The datasets contained in the benchmark are in four sets, based on two criteria: Whether the target is classification or regression and whether the dataset contains at least one categorical feature or only numerical ones. Since the FCT model handles only classification, we also take into account only two of the four sets that have classification as a target. All visualizations will thus distinguish two types of datasets, named numerical and categorical for short. Note that the categorical datasets contain at least one categorical feature. Not all features must be categorical. Indeed, there are datasets that, even without the categorical features, satisfied

the conditions set by the authors of the benchmark and are thus both in categorical and numerical sets.

All runs were performed on 10 random splits of each dataset to training and testing sets. The training set is 80% of the entire dataset, bounded to at most 10 000 samples to satisfy "mid-sizedness" as was the intention and practice of the benchmark authors. This way, we preserve comparability to Grinsztajn et al. [2022]. The test set consists of the remaining 20% of the dataset. The same splitting process is performed with 10 different random seeds for the split.

3.4 Results

We start by showing results on a single dataset that has a lot of importance to the XAI community. The COMPAS dataset contains data from a system designed to help judges make better decisions. The system creators were accused of racial bias [Julia Angwin et al., 2016]. The version used here (and in the benchmark) is a simplified variant that contains basic information about the defendant, and the target feature is a binary variable evaluating true if the defendant re-offended in the next two years.

See Figure 3.5 for comparing our results to CART aggregated over multiple runs. We notice a drop in model accuracy when comparing FCT (cca 0.65) to CART (cca 0.67), which is understandable, given the fact that model accuracy is not the objective of FCT formulation. On the other hand, we see an increase of more than 10 percentage points in leaf accuracy, substantially improving the validity of the explanations. The presented CART trees are well optimized. Both types of trees were extended (and also reduced first), so their results can be compared.

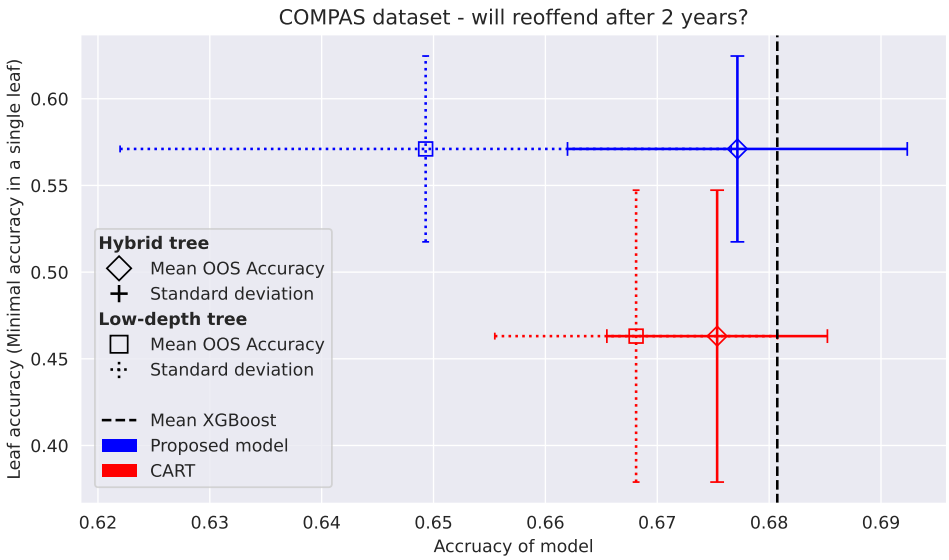


Figure 3.5: Aggregated results for 10 different data splits of the *compas-two-years* dataset. We observe the performance of 2 kinds of trees with extensions. We see a lower performance in total accuracy of FCT (the Proposed model) compared to CART method, but better leaf accuracy for FCT, for which it was the optimized variable. Upon extending the models, the model performance increases for both CART and FCT, with FCT having a slight edge.

3.4.1 Comparison to other methods

Figure 3.6 shows the aggregated performance of FCT (Proposed) model, the warmstarted variant of OCT and optimized CART. We see that our model outperforms not only CART but also OCT which served as the basis for our model. On average, our model improves the minimal leaf accuracy by 11.16 percentage points compared to CART.

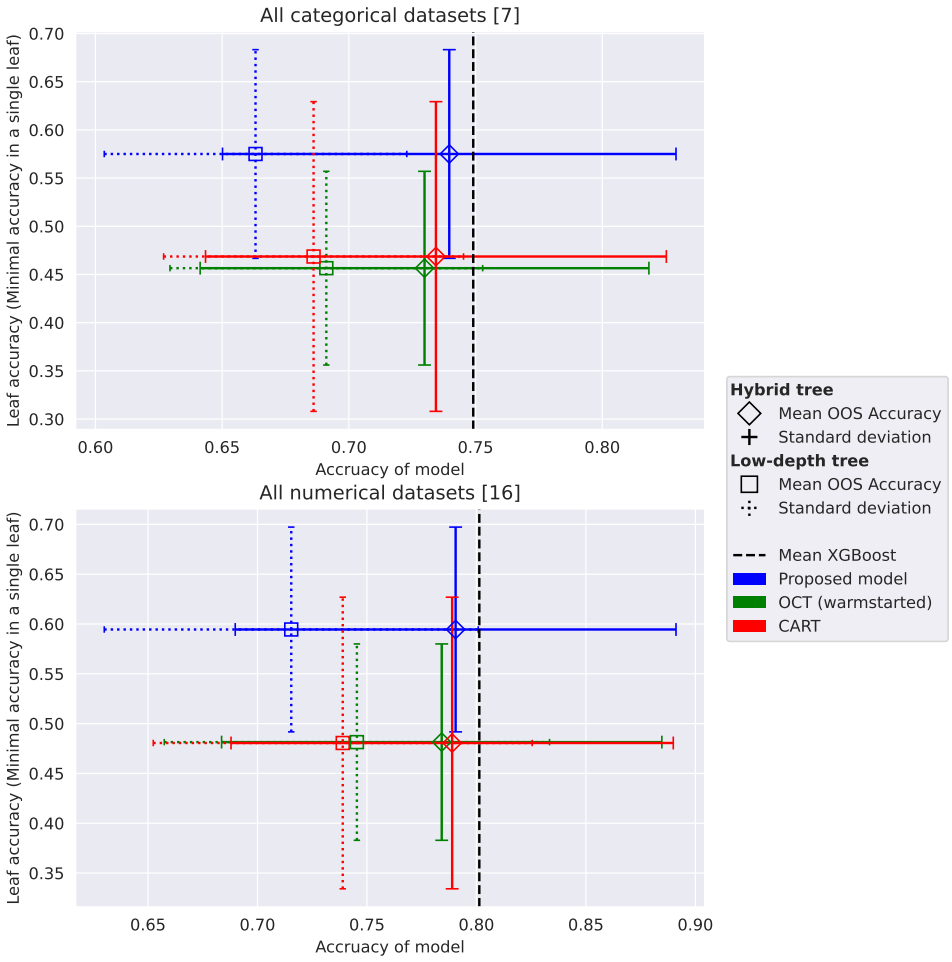


Figure 3.6: Results on the testing sets. We see a significant increase in leaf accuracy when using FCT formulation. A slight increase in hybrid tree accuracy can also be noticed, although it has low statistical significance. For more detailed results, refer to the comparison in Table 3.2.

	Minimal	Mean (± std)	Maximal
Leaf Accuracy - CART			
categorical	−0.0247	0.1064 ± 0.0729	0.1905
numerical	−0.0553	0.1139 ± 0.0630	0.2116
Model Accuracy - CART			
categorical	−0.0027	0.0053 ± 0.0084	0.0182
numerical	−0.0253	0.0016 ± 0.0086	0.0111
Model Accuracy - XGBoost			
categorical	−0.0228	-0.0095 ± 0.0064	−0.0036
numerical	−0.0276	-0.0108 ± 0.0076	0.0005

Table 3.2: Improvements of mean accuracy on datasets between FCT and CART or XGBoost. Data is computed by subtracting the mean accuracy of CART or XGBoost, respectively, from the mean accuracy of our model on each dataset. In the first two rows, we compare the leaf accuracy of our model to CART. In the middle two rows, we compare the model accuracy of the hybrid FCT trees with hybrid CART trees. In the last two rows, we compare our hybrid tree model to the mean XGBoost model trained on the same dataset. Detailed numeric results are in Appendix C.3.

3.4.2 Dependance on dataset difficulty

With Figure 3.7 we explore the behavior for datasets of varying difficulty. We define three categories of difficulty based on the mean accuracy of the XGBoost model obtained by the benchmark authors. The thresholds of difficulty categories are 0.7 and 0.8 for datasets containing some categorical features and 0.75 and 0.85 for datasets with only numerical features.

This was done to see how well the proposed formulation behaves based on some intrinsic complexities of a dataset. We notice a more stable increase in leaf accuracy, positively correlated with the increase in hybrid tree accuracy of FCT. Compared to CART, our model seems to perform better on categorical data.

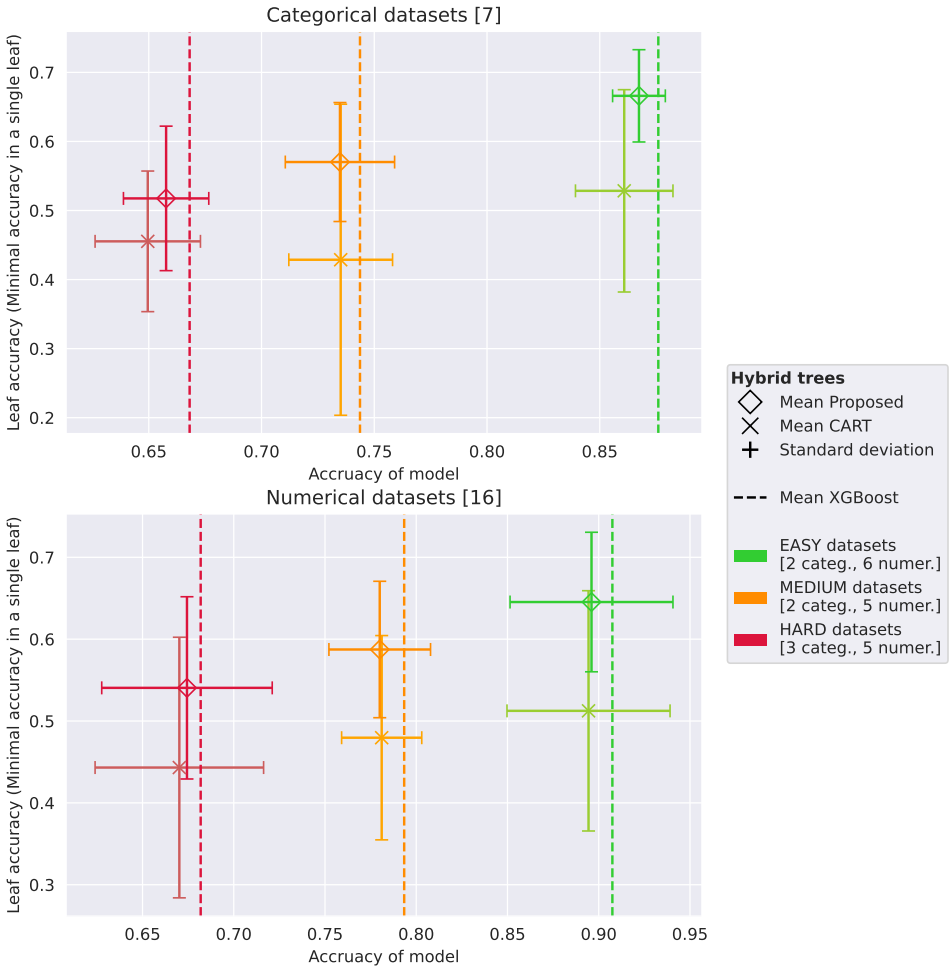


Figure 3.7: Results on testing data on all datasets. In square brackets are the numbers of datasets belonging to each difficulty set. This plot shows that our method, when extended in leaves, does not significantly decrease overall performance compared to pure XGBoost while sometimes improving upon accuracy obtainable by extended CART. It also shows that this capability is invariant to the difficulty of the dataset.

3.4.3 Required resources

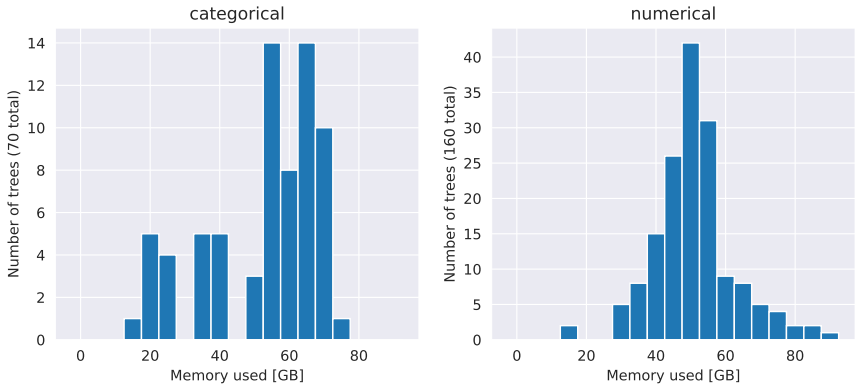
We performed all experiments on an internal cluster with sufficient amounts of memory. Each run of the MIP solver has been limited to 8 hours on 8 cores of AMD Epyc 7543, totaling 64 core-hours per split of a dataset. The extension part takes, on average, around 3 core-hours per split. This totals around 15,500 core-hours for the entire classification part of the benchmark of Grinsztajn et al. [2022] and one configuration of hyperparameters.

The entire optimization of CART of depth 4 with the extensions of the leaves took around 500 core-hours for the entire classification part of the benchmark.

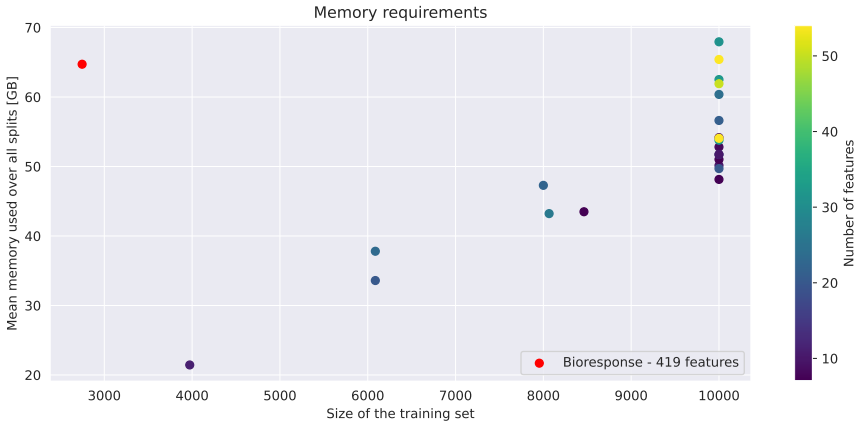
Memory requirements

Overall, the memory requirements of the datasets for our 8-hour runs were between 15 and 95 GB. On average, all datasets required at most 70 GB of working memory. Figure 3.8 shows the memory requirements of our formulation in more detail. The extension phase of the process is negligible in this regard, as it requires only about 1.5 GB of working memory for all the datasets. Training and extending the CART models also required less than 2 GB of working memory.

The amount of memory required by the MIP solver is dependent on the size of the data in the number of training samples, as well as the number of features. Figure 3.8b shows this linear dependence of memory requirements on the size of the training set. Based on the coloring of the nodes, we also see the dependence on the number of features, especially in the case of the Bioresponse dataset.



(a) : Histogram of memory requirements of MIP solver for all dataset splits.



(b) : Mean memory requirements on datasets. Dots are colored according to the number of features. Dataset Bioresponse is excluded from the color mapping due to having a significantly higher number of features. Training sets were clipped to a maximum of 10,000 points.

Figure 3.8: Memory requirements mostly do not exceed 70 GB. The memory requirements increase slightly when more time is given to the solver and significantly increase when bigger training sets are considered. We can also see some correlation between the number of features and memory requirements when looking at same-size datasets.

3.4.4 Helper methods

In this section, we wish to explore the methods introduced to improve the performance of the model. We will be comparing the use of Soft accuracy and Tree reduction.

Soft accuracy

The notion of leaf accuracy is very unstable. The value is easily affected by a single inaccurate leaf. This observation led to the idea of soft accuracy, where we are more benevolent with less-populated leaves. For details, see page 23.

Though it tackles well the issue of less populated leaves with high inaccuracy, it does not provide improvements when considered during the optimization. In Figure 3.10 one can see that if we consider the leaf accuracy on testing data to be performed without the soft accuracy switch, the accuracy drops well below the range of CART.

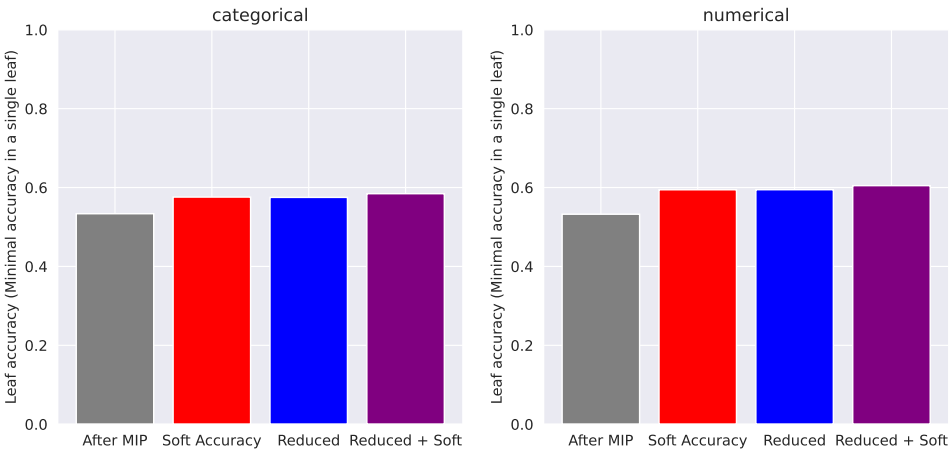


Figure 3.9: Comparison of the effect of proposed methods on leaf accuracy. Each bar represents a mean value of leaf accuracy on testing data. The grey bar shows the leaf accuracy right after the MIP solver ends. The red and blue are after we compute the soft accuracy or after we reduce the tree, respectively. The purple bar represents both methods combined. We see that soft accuracy helps the leaf accuracy about the same amount as the reduction of the tree. Indeed, both reduce the effect of inaccurate, less-populated leaves. Their combination does not yield significant further improvement.

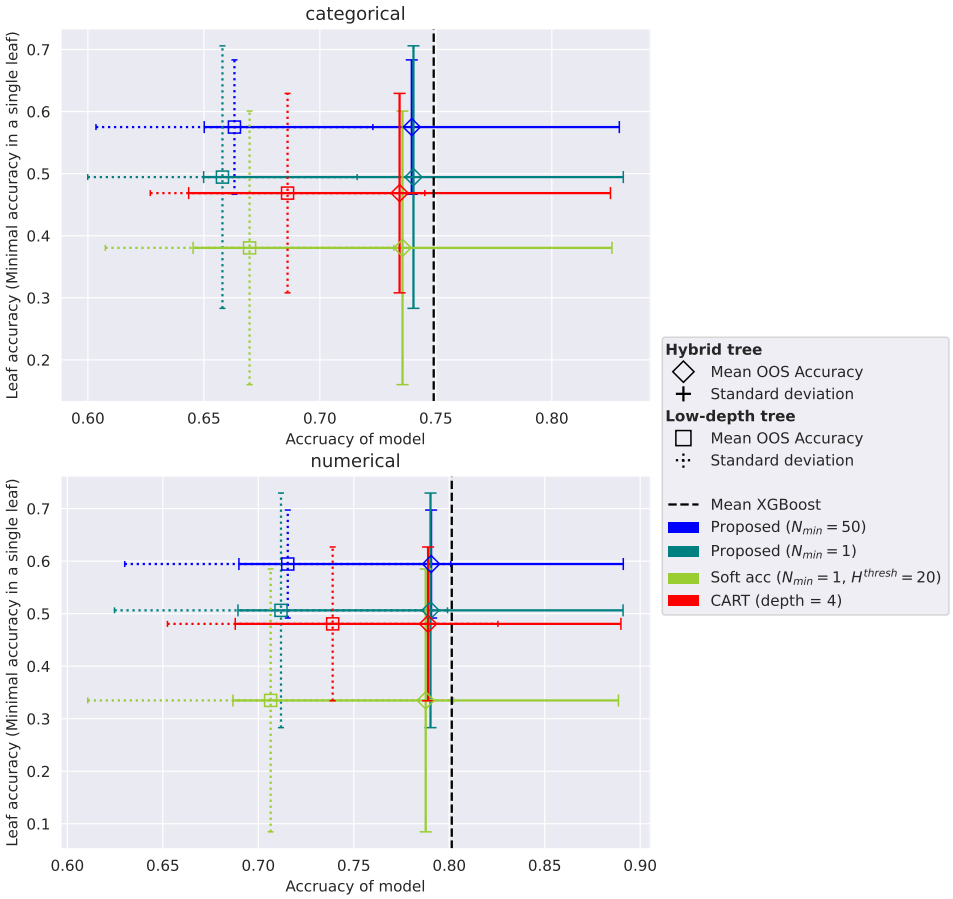
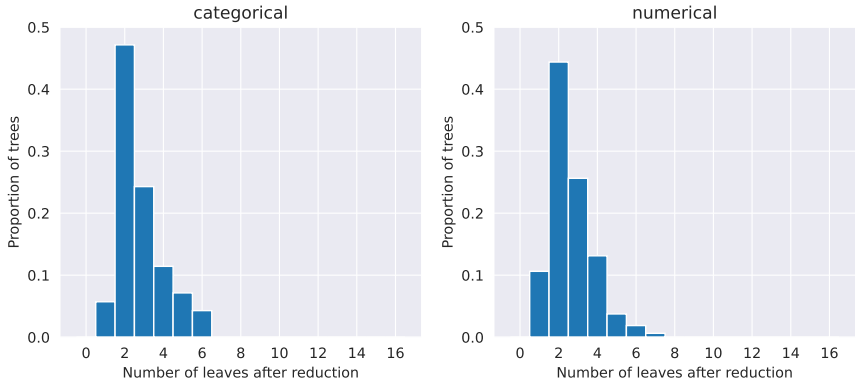


Figure 3.10: Performance using soft accuracy with threshold $H^{\text{thresh}} = 20$. We compare 3 configurations of FCT here. The blue is our selected configuration. Green has relaxed constraints on the minimum of samples in a leaf. Yellow is the same but trained using soft constraints. It performs the worse by far.

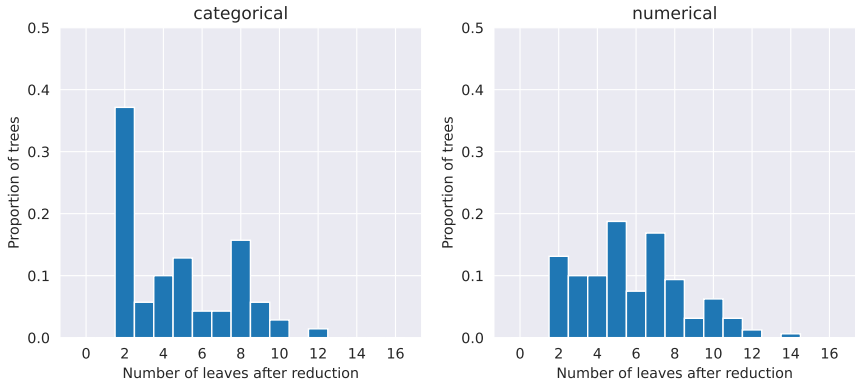
Tree reduction

Tree reduction is a good way to obtain more robust leaf accuracy. By reducing the tree without hurting the accuracy, we achieve the same improvement as by creating an ad-hoc concept of soft accuracy. See Figure 3.9

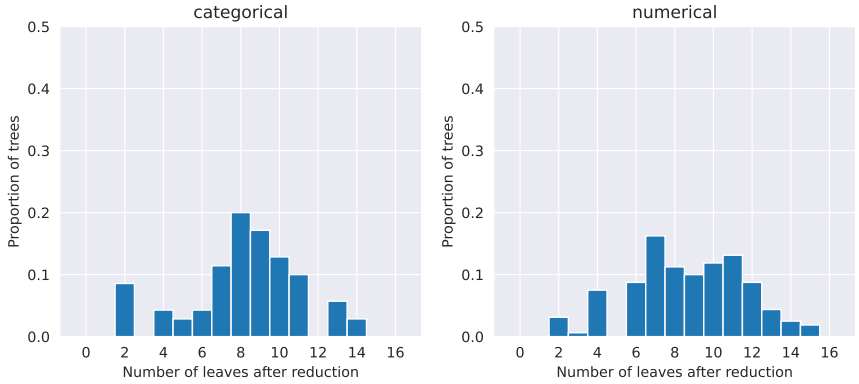
In Figure 3.11, we compare the complexity of the created trees by comparing the distributions of the number of leaves (or potential explanations) provided by the method. The CART model averages around 8 leaves after reduction. Our proposed model's distribution is close to that of CART models. When solving the MIP formulation directly, the distribution is severely shifted toward very small trees. Our proposed method uses a default CART solution to warmstart the search.



(a) : Histogram of the number of leaves of the reduced trees optimized directly using the proposed formulation. The trees are heavily pruned.



(b) : Histogram of the number of leaves of the reduced FCT trees created with the proposed formulation, warmstarted using a simple CART solution. The trees are smaller than well-optimized CART but retain some complexity. This was the chosen method.



(c) : Histogram of the number of leaves of the reduced trees created by CART with optimized hyperparameters.

Figure 3.11: Comparison of the numbers of leaves in trees after the reduction procedure.

3.4.5 Other optimization strategies

To show that the warmstarting optimization method is truly the best, we show the comparison in Figure 3.12. All methods have equivalent conditions and hyperparameters, including the time to optimize. To refresh the reader, we provide a quick description of

- Direct refers to the straightforward use of the MIP formulation.
- Warmstarted uses a simple CART solution (created using default hyperparameters) as a starting point of the solving process.
- Gradual refers to a special process where we use a shallower tree to train a tree with a depth higher by 1 until we reach the desired depth.
- Halving is the primitive method of using a slightly extended OCT formulation and performing the halving algorithm. See Algorithm 1.

All of the three approaches were run with the same resources. This meant that even the Gradual approach took 8 hours in total. The time was distributed so that the available time for the optimization process doubled with each increase in depth. This means 32 minutes for the first run, 64 minutes for the tree of depth 2, 128 for depth 3, and 4 hours 16 minutes for the final tree with depth 4.

For the Halving approach, we use the estimate of the number of runs needed (see Equation (3.18)) and divide the amount of time to n_{runs} equal parts.

Two interesting things manifest. Firstly, the halving algorithm shows respectable performance. One would likely not expect such performance from a first-draft variant of the solution.

Secondly, the gradual approach shows promise. It has higher hybrid-tree accuracy by another 0.2 percentage points on average and has lower leaf accuracy by only about 1.2 percentage points compared to the Warmstarted approach (see Table 3.3 for the numerical comparison).

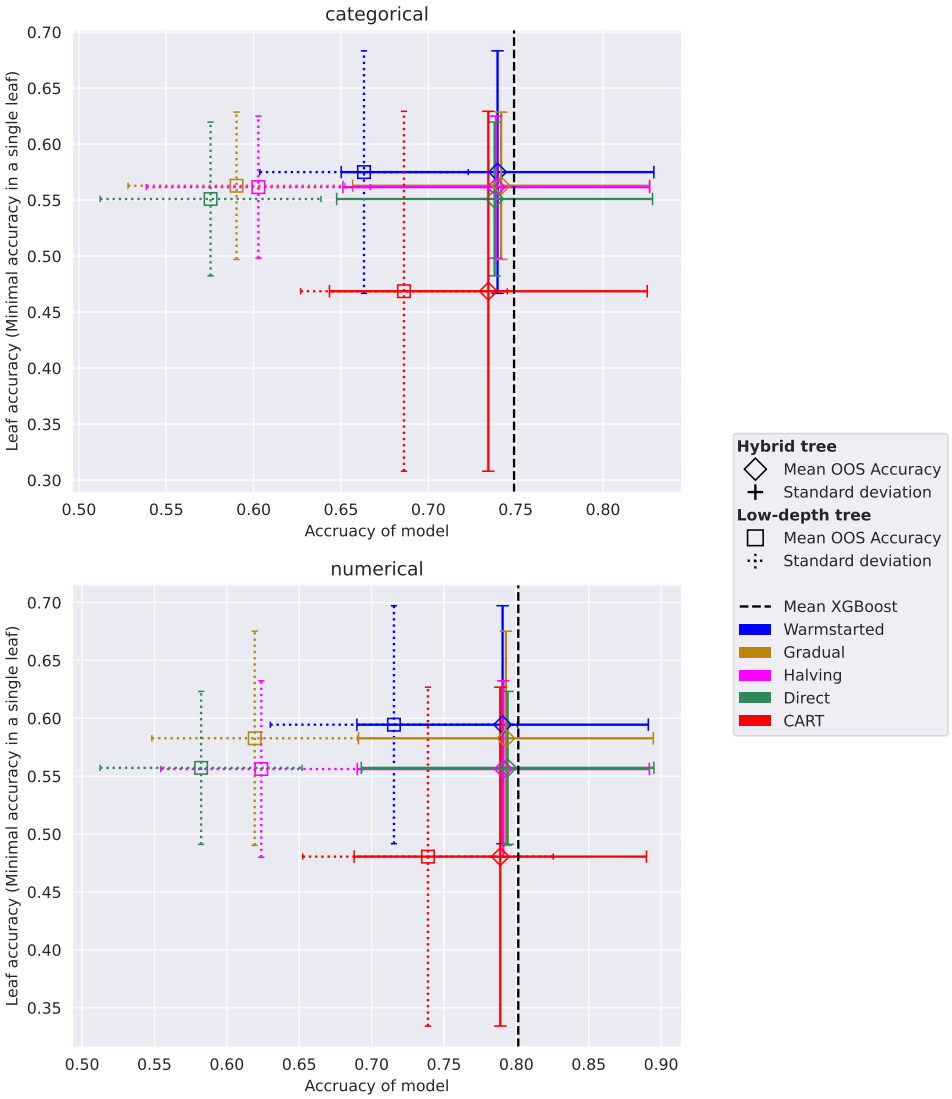


Figure 3.12: Comparison of the various strategies to the optimization given the same resources and conditions. Warmstarted refers to starting the optimization process with a CART solution. The gradual approach is the approach of increasing the depth of a tree and starting each new depth with the solution of the previous shallower tree. Direct means a simple, straightforward optimization of the formulation, as it is stated, without any hints. Halving is the implementation of OCT and a hard constraint on the leaf accuracy, that changes as the Algorithm 1 progresses. All four approaches were run with the same resources. For a closer investigation of the Gradual approach, see Table 3.3.

	Data type	Min	Mean (± std)	Max
Leaf Accuracy	categorical	−0.1094	0.0122 ± 0.0753	0.1130
	numerical	−0.0867	0.0117 ± 0.0624	0.1154
Model Accuracy	categorical	−0.0219	-0.0021 ± 0.0094	0.0083
	numerical	−0.0103	-0.0023 ± 0.0056	0.0076

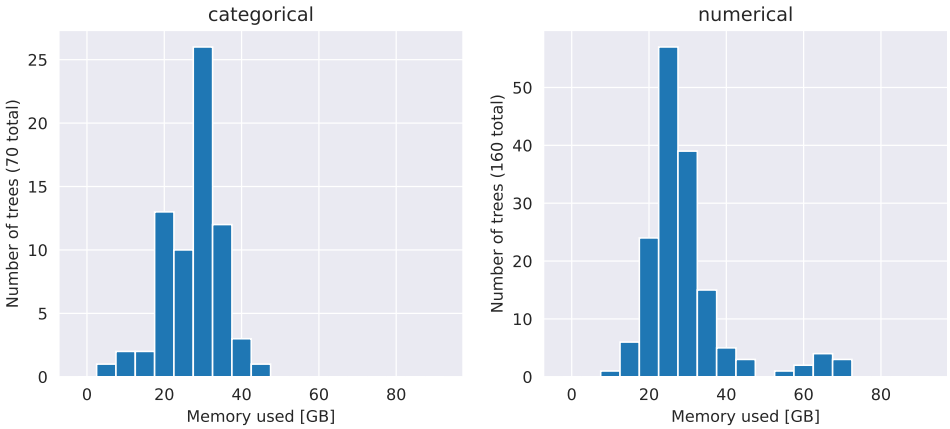
Table 3.3: Comparison of Gradual and Warmstarted approach. Positive numbers show an advantage in the mean accuracy of the Warmstarted approach. Gradual refers to the approach when the depth of the tree is gradually increased during the optimization process.

3.4.6 Shorter time

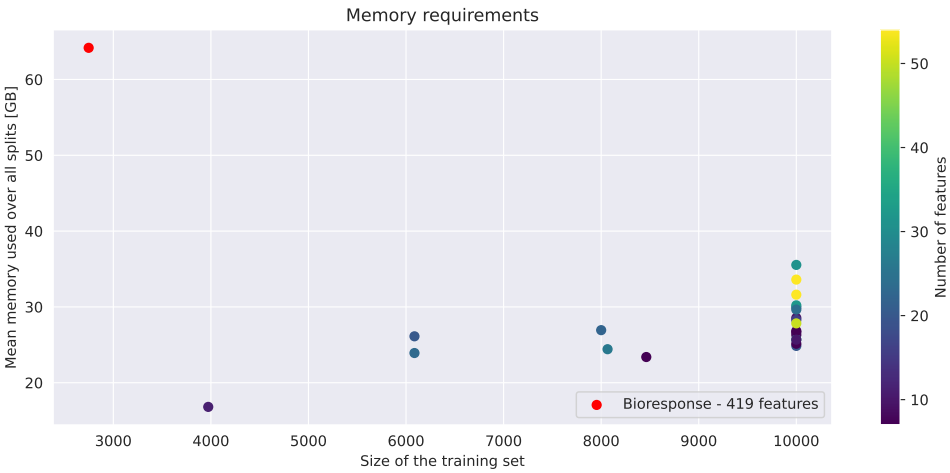
When considering a shorter time for optimization, we can lower the memory requirements from up to 95 GB to levels attainable by current personal computers. When optimizing our MIP model for one hour, the required memory is below 50 GB for all datasets except Bioresponse, which has one order of magnitude more features than the rest of the datasets included in the benchmark.

The mean memory requirement is below 30 GB of working memory (compared to 50 GB for the 8-hour run). See Figure 3.13 for details.

Figure 3.14 shows that even with this limited time, compared to CART, we can achieve significant improvement in leaf accuracy and similar accuracy of hybrid trees.



(a) : Histogram of memory requirements of MIP solver for all dataset splits.



(b) : Mean memory requirements on datasets. Dots are colored according to the number of features. Dataset Bioresponse is excluded from the color mapping due to having a significantly higher number of features. Training sets were clipped to a maximum of 10,000 points.

Figure 3.13: Comparison to a version of the FCT (Proposed) model that the Gurobi solver optimized for only one hour. Compared to the main configuration, which ran for 8 hours, we notice a significant decrease in memory requirements for most datasets, up to tens of gigabytes. An outlier dataset Bioresponse with cca 10 times more features sees a smaller decrease of about 2 GB.

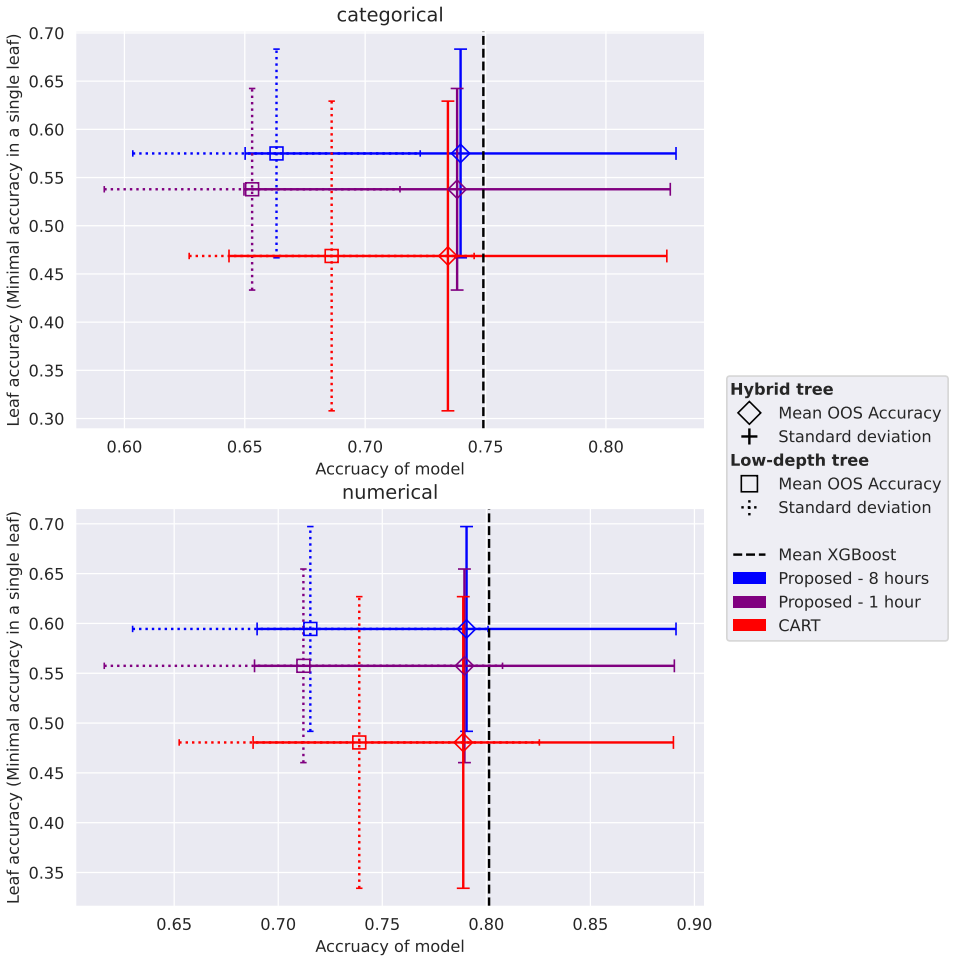


Figure 3.14: Comparison of the performance of the FCT model after 1 and 8 hours of optimization.

3.4.7 MIP solve process

Although the MIP solver works towards global optimality, the road there is lengthy. Figure 3.15b shows the progress of the MIP gaps during the 8-hour optimization averaged over all datasets. For a detailed look, see Figure 3.16. We see that the solution is still improving, albeit rather slowly, after 8 hours. Given that none of the runs improved the objective bound from 1 noticeably, the narrowing of the MIP gap is achieved only through finding a better feasible solution. The objective bound always starts at 1 because of the constraints on the variable Q.

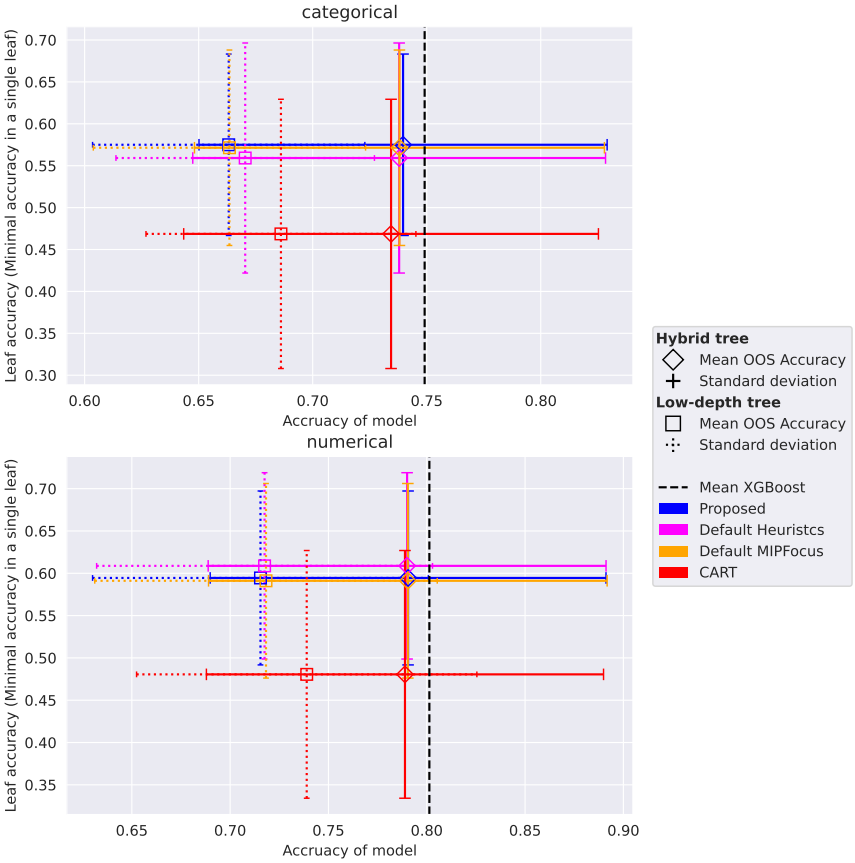
This lack of improvement of the objective bound might have been affected by our Gurobi parameter settings which focused on finding feasible solutions and heuristic search. However, in tests with default parameters, the best bound did not improve either.

Default parameters of Gurobi solver

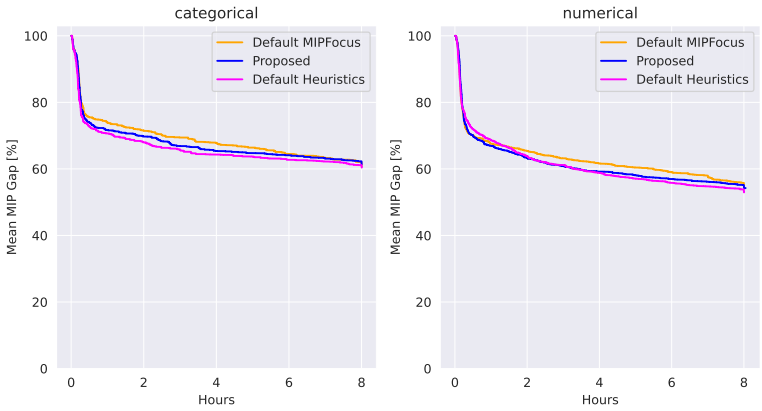
To measure the performance improvement of our choice of parameters of the solver, we ran a test with the default value of the MIPFocus parameter and a test with the default value of the Heuristics parameter.

The results (see Figure 3.15a and Tables 3.4, 3.5) show no significant improvements regarding the MIPFocus parameter. However, with the default value of the Heuristics parameter, we observe an improvement in performance on numerical datasets and a decrease in performance on categorical datasets. The absolute differences are about the same, so we opted for the variant with similar performances on both categorical and numerical datasets, that being the proposed variant focusing on heuristics. The proposed configuration also shows a more stable increase in accuracy w.r.t. the performance of CART models. The solver performance varies per dataset, as visualized by Figure 3.16.

These differences in performance suggest that parameter space regarding the MIP solver should be further explored.



(a) : Comparison of the FCT model to models with default parameter configurations show varying results. MIPFocus seems to influence the search only very slightly. Heuristics, on the other hand, show significant improvement on numerical datasets and a decrease in performance on categorical datasets, with about the same value.



(b) : Mean MIP optimality gap development over the solving time, averaged over all datasets. For a non-aggregated version, see Figure 3.16.

Figure 3.15: Comparison of models with the proposed base configuration of Gurobi parameters and default parameters.

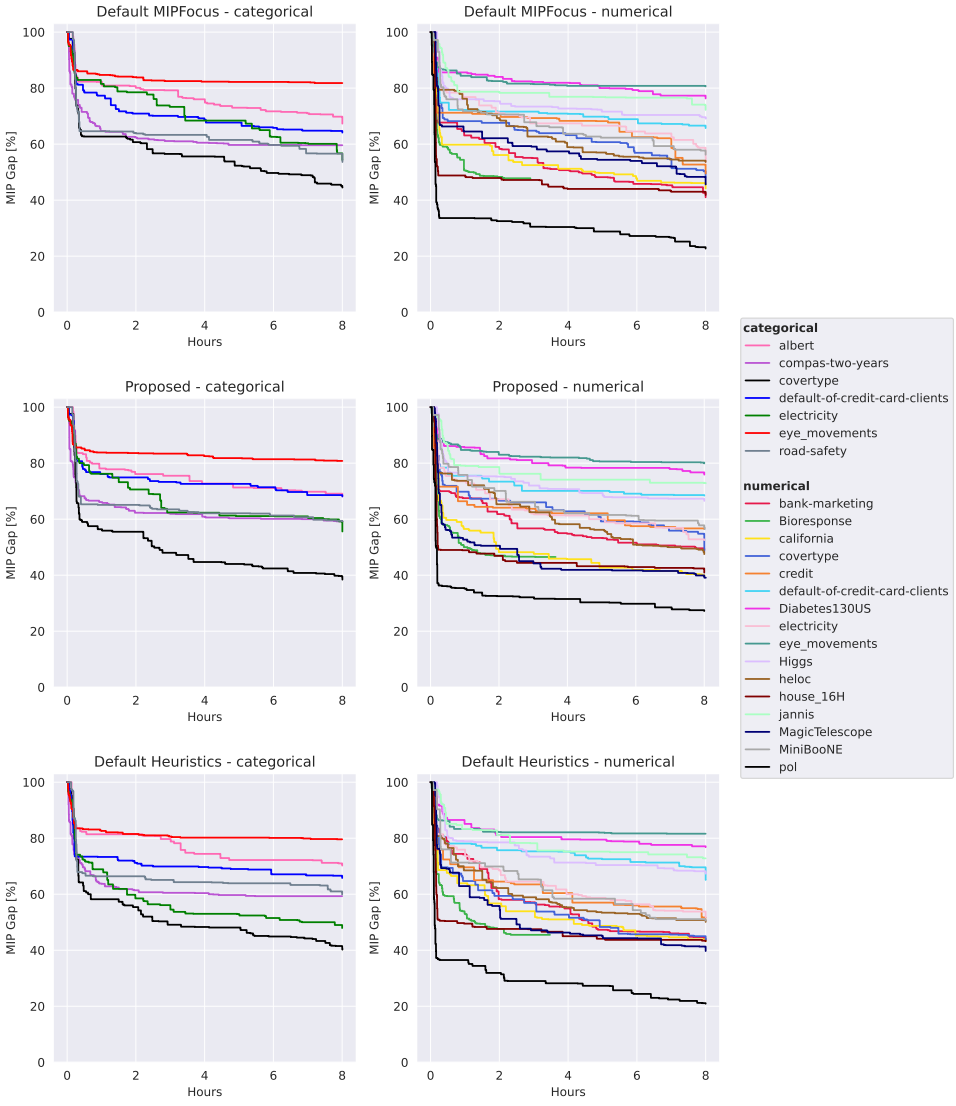


Figure 3.16: Mean MIP optimality gap development over the solving time. The figure shows the progress of the value of the MIP optimality gap averaged over all splits of each dataset. Each line corresponds to one dataset. For an aggregated version, see Figure 3.15b

Default Heuristics	**Data type**	**Min**	**Mean (± std)**	**Max**
Leaf Accuracy	categorical	−0.0117	0.0158 ± 0.0234	0.0531
	numerical	−0.1178	-0.0143 ± 0.0402	0.0435
Model Accuracy	categorical	−0.0011	0.0017 ± 0.0035	0.0094
	numerical	−0.0047	0.0005 ± 0.0025	0.0062

Table 3.4: Detailed view of the differences in the accuracy between default Heuristics value and the proposed configuration. A positive number means the accuracy advantage of the proposed parameter configuration. This shows a numerical representation of what can be seen in Figure 3.15a. We see absolute mean differences of comparable values. The negative difference on numerical datasets also has a higher standard deviation, suggesting a stronger influence by an outlier dataset.

Default MIPFocus	**Data type**	**Min**	**Mean (± std)**	**Max**
Leaf Accuracy	categorical	−0.0304	0.0036 ± 0.0213	0.0299
	numerical	−0.0528	0.0034 ± 0.0342	0.0788
Model Accuracy	categorical	−0.0028	0.0016 ± 0.0043	0.0088
	numerical	−0.0026	0.0001 ± 0.0019	0.0032

Table 3.5: Detailed view of the differences in the accuracy between the default MIPFocus value and the proposed configuration. A positive number means the accuracy advantage of the proposed parameter configuration. This shows a numerical representation of what can be seen in Figure 3.15a. Both variants seem to perform comparably, with a potential slight edge in favor of the proposed configuration.

3.4.8 Further Ablation Analyses

We provide some comparing experiments performed by changing a single hyperparameter, or a few closely related hyperparameters, in the case of CART.

Unlimited depth CART

An argument could be made against the choice to compare our method to CART trees with depth maximally equal to 4. Figure 3.17 and Table 3.6 in more detail show a comparison of CART models with a maximal depth of 4 and a maximal depth of 20. The actual depth limit for each model was optimized along with other hyperparameters using the Bayes hyperparameter optimization procedure.

The aggregated results show worse performance regarding both leaf accuracy and hybrid-tree accuracy. Not only do the deeper trees perform worse, but the length of provided explanations is also well above the 5-9 threshold suggested as the limit of human understanding.

	Data type	Min	Mean ($\pm$ std)	Max
Leaf Accuracy	categorical	-0.0769	0.2053 ± 0.2389	0.5404
	numerical	-0.1183	0.2441 ± 0.2115	0.5680
Model Accuracy	categorical	-0.0025	0.0173 ± 0.0185	0.0420
	numerical	-0.0006	0.0156 ± 0.0119	0.0370

Table 3.6: More detailed view of the differences in the accuracy between CART trees with max depth 4 and CART trees with max depth 20. A positive number means the accuracy advantage of the more constrained model. This shows a numerical representation of what can be seen in Figure 3.17

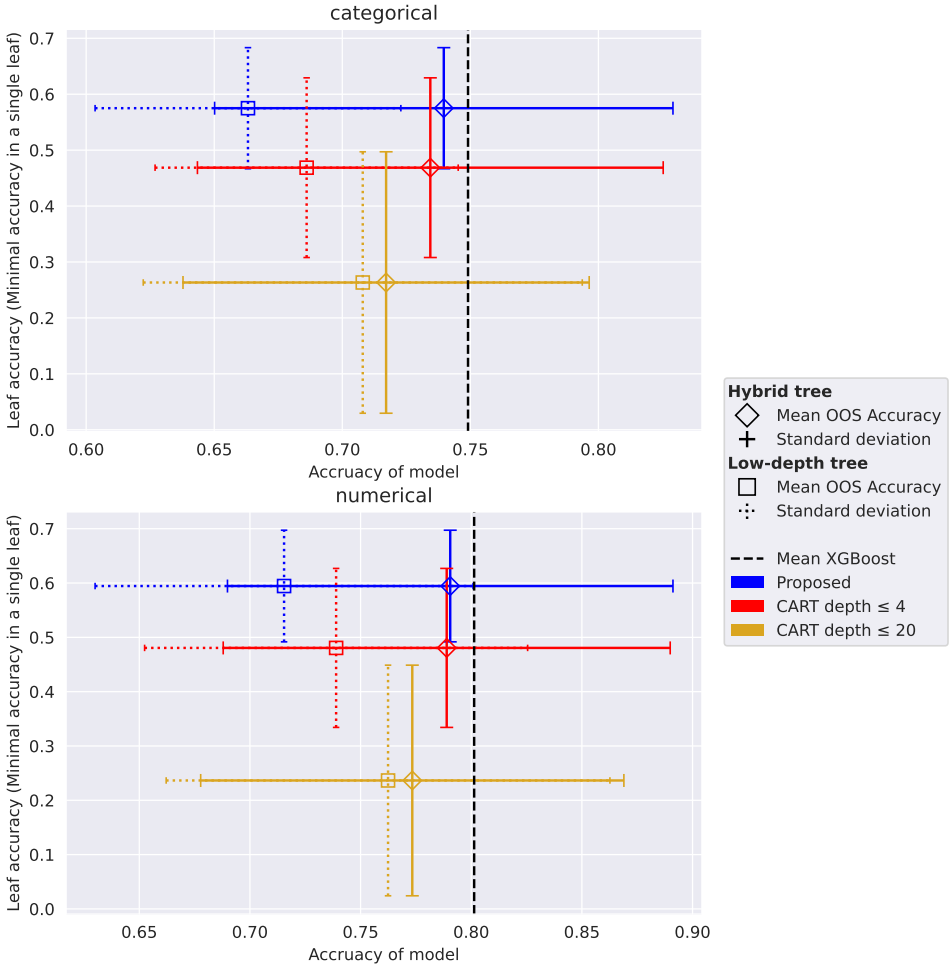


Figure 3.17: Comparison of CART tree results with limited depth and without such a strict limit on the depth. Deeper trees provide worse explanations (due to the length of explanation) and perform worse in general accuracy. For a more detailed description of the differences introduced by the depth, consult Table 3.6

No minimum number of samples in leaves

This comparison, see Figure 3.18, shows the importance of setting a minimal amount of samples in leaves. Without enough points to support the obtained leaf accuracy, it is more likely to be overfitted. On the other hand, when choosing the N_{min} parameter too high, we disable the solver to perform some possibly beneficial splits, if they are supported by only a small amount of training data.

This is certainly an important hyperparameter and further testing could provide more insight into the proposed model's performance.

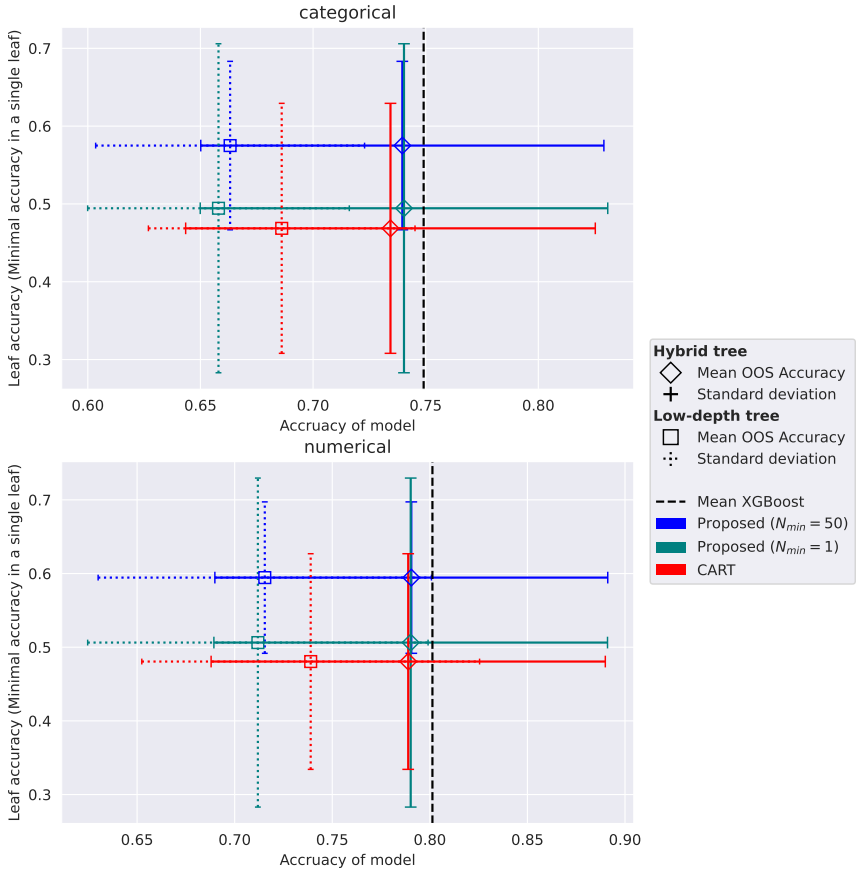


Figure 3.18: Comparison of performance of the proposed model with minimum samples in leaves equal 50 and 1. Without the constraint, leaf accuracy is supported by a low amount of samples, leading to higher overfitting to training data and worse out-of-sample performance. Notice the high variance of the model without a lower bound on the number of samples. CART optimized its minimal amount of samples in leaves in hyperparameter search, according to Table C.3.

Non-warmstarted OCT

We compare our method to warmstarted OCT because both start from the same CART initial solution. This makes them more comparable. However, we also tested the OCT variant, directly optimized from the MIP formulation. See the results in Figure 3.19. All OCT models were run with the same hyperparameters as the proposed model. Heuristics-oriented solver, depth equal to 4, and a minimal amount of samples in leaves equal to 50.

The average OCT performs worse than all our approaches (see Figure 3.12), but the improvement from the warmstarted variant is intriguing. Especially considering that it is not caused by the direct OCT method's inability to create complex trees without warmstarting. This is supported by Figure 3.20 showing a distribution of leaves similar to the distribution of CART trees (see Figure 3.11).

This suggests that the OCT trees have comparable complexity to CART and provide more valid explanations than CART, even without our extension to the formulation. This is an interesting result, considering the fact that neither CART nor OCT methods optimize for leaf accuracy. Our model, however, almost doubles the improvement of direct OCT. It improves by a similar number of percentage points as OCT improves on CART.

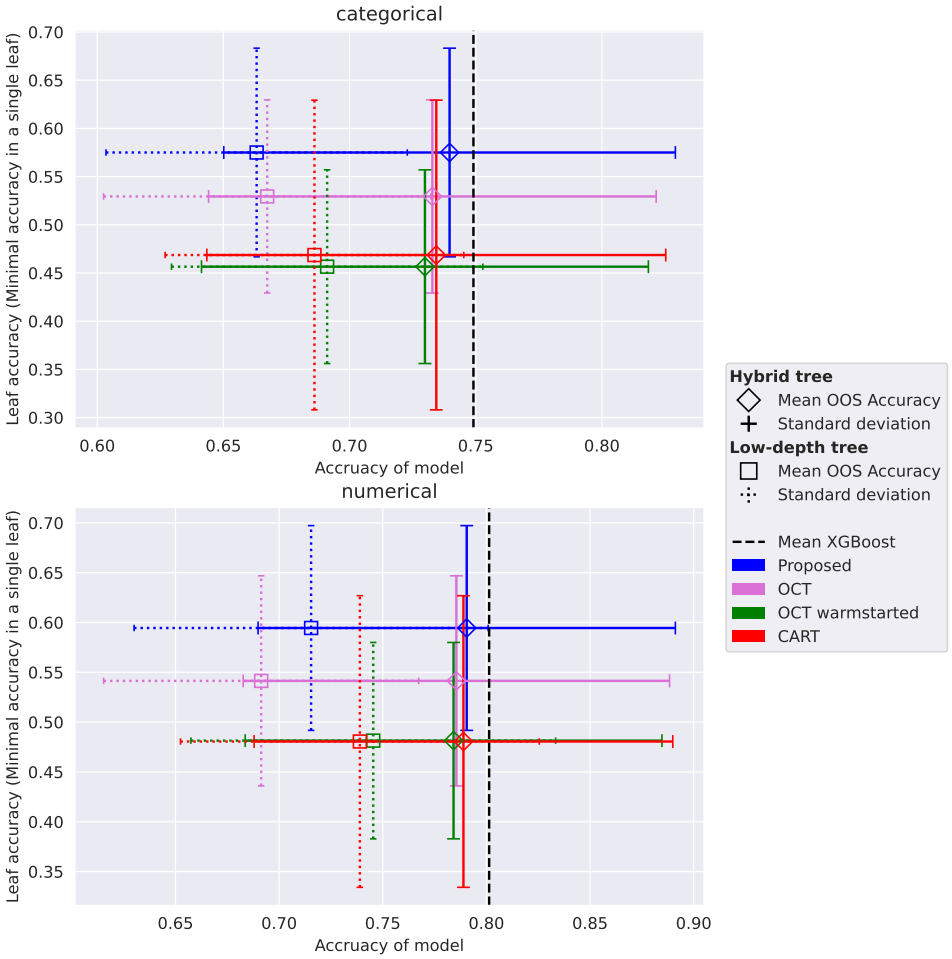
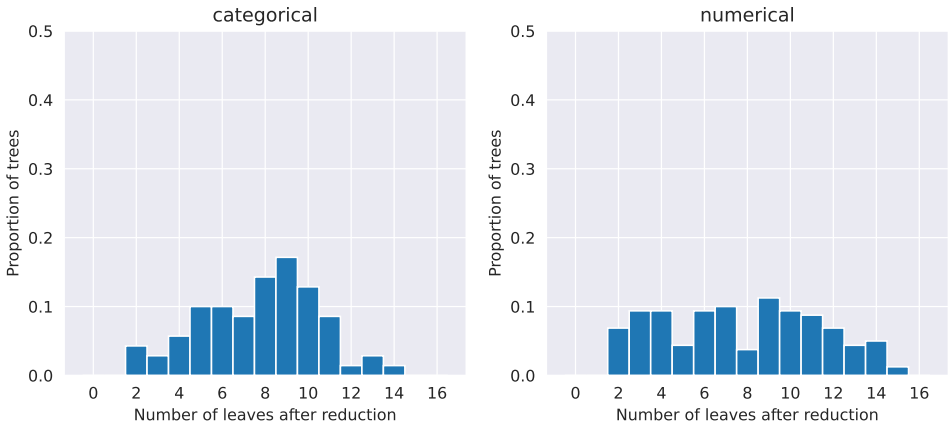
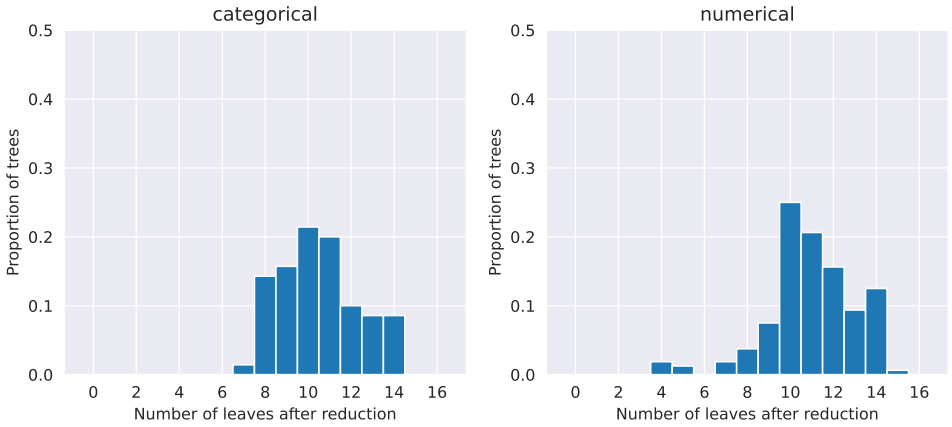


Figure 3.19: Comparison of OCT trees that are warmstarted the same way as our FCT model and OCT without the warmstart, optimized directly. Interestingly, direct OCT performs significantly better.



(a) : Histogram of the number of leaves in the reduced trees of the direct OCT method



(b) : Histogram of the number of leaves in the reduced trees of the warmstarted OCT method.

Figure 3.20: Comparison of reduced tree complexity of the OCT with and without warmstart. OCT without warmstart creates trees of similar distribution as the CART method (see Figure 3.11) but achieves better leaf accuracy than CART (see Figure 3.19) despite neither of them optimizes that objective.

Influence of depth

Lastly, we provide a comparison of our trees of depths 3, 4, and 5. Figure 3.22 shows the best results for the shallowest tree. Because of the exponential increase in memory requirements, deeper trees are not able to perform the same amount of optimization as the shallower ones. This is clear from the formulation, but we also provide experiment data in Figure 3.23.

With a model of twice the complexity, the solver struggles to achieve comparable results to our proposed model with lower depth. In the optimum, deeper trees should only improve the leaf accuracy or stay at the same level. We work with a time limit, and the solver handles smaller formulations better. This is certainly a topic of further exploration, for example, by incorporating scalability improvements to the model proposed in the literature [e.g. Verwer and Zhang, 2019; Aghaei et al., 2020, 2022].

We see that the model with depth 3 is performing the best by far. Since we selected the initial configuration to use a depth of 4, all other compared configurations were set to use it for comparability. Although the trees of depth 3 have the best leaf accuracy, we do not present them as the main method. The reason is that the trees are significantly smaller (see Figure 3.21) and that we would further restrict the length of explanations to at most 3.

This is certainly a hyperparameter worth thinking about, to figure out what kind of trees one wants.

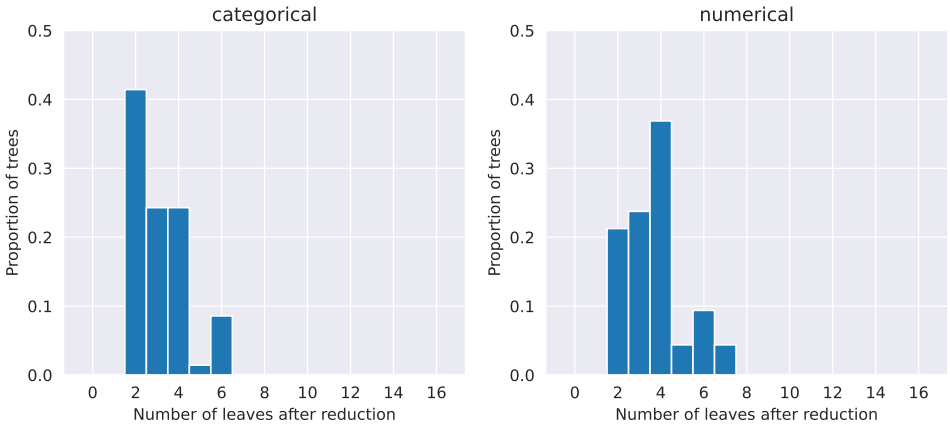


Figure 3.21: Number of leaves in the reduced FCT models of depth 3. Distribution is understandably skewed compared to the distribution of FCT of depth 4 in Figure 3.11b.

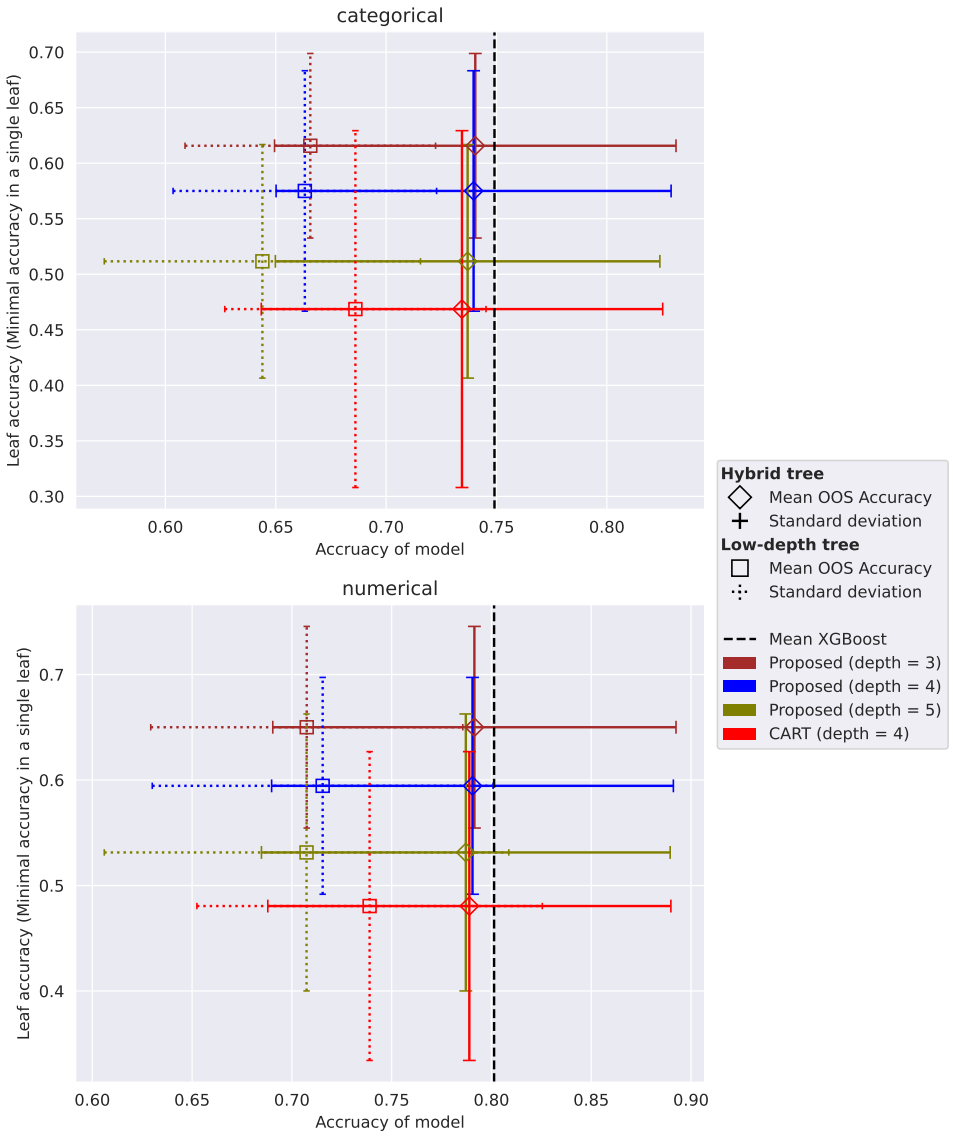
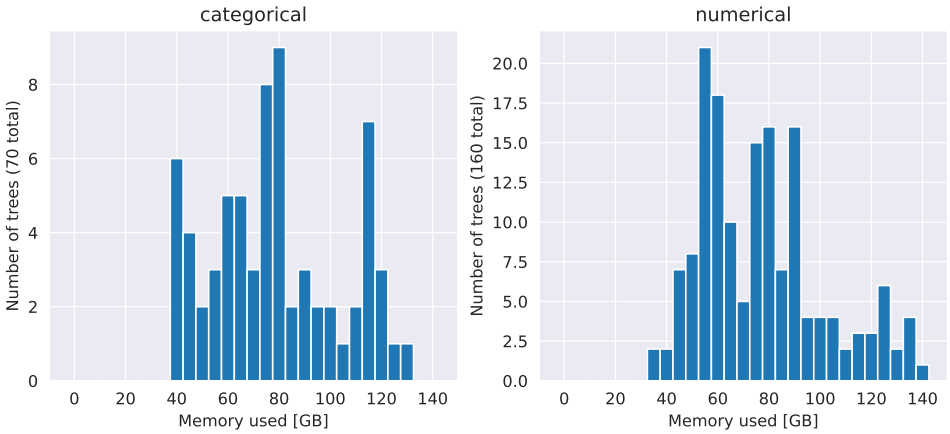
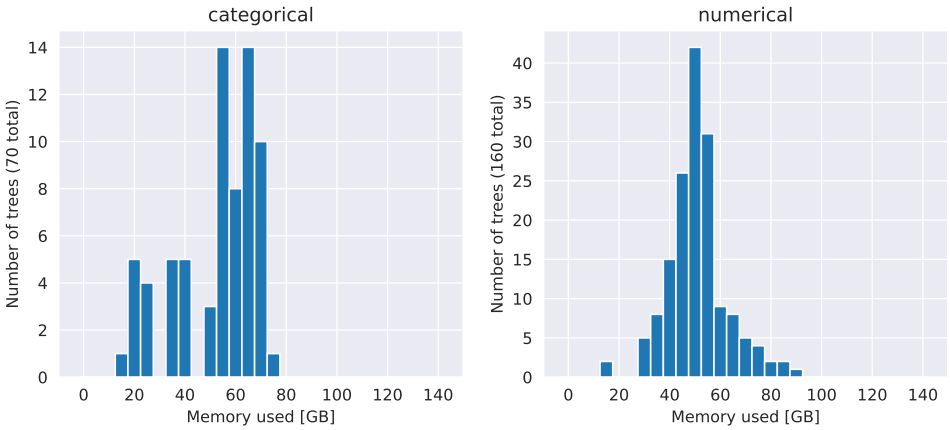


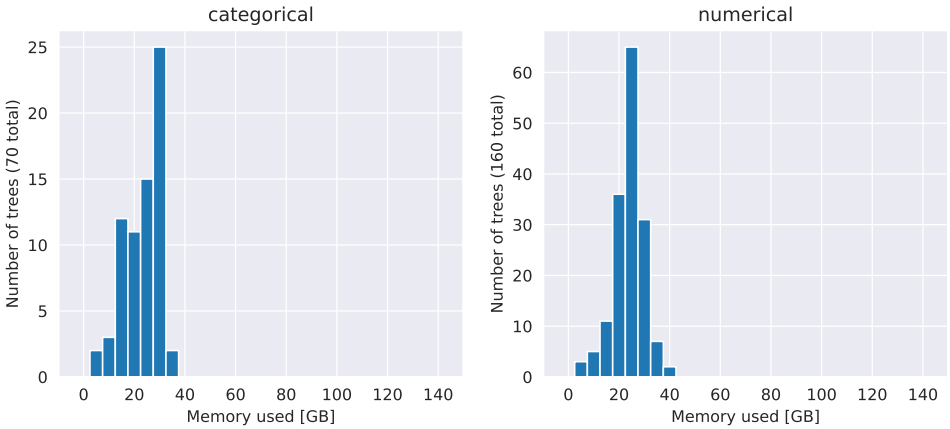
Figure 3.22: Comparison of performance of the proposed model with depth 3, 4, and 5. Depth 5 seems to be beyond the limit of what is tractable to efficiently optimize in 8 hours, and depth 3 limits the possibilities of explanations too much. However, it has by far the best leaf accuracy. Comparisons to CART of depths 5 and 3 would bring more information and a better understanding of up to what degree this is a feature of trees as a model and up to what degree is this a feature of our optimization formulation.



(a) : Histogram of memory requirements of MIP solver for all dataset splits for trees of depth 5, solved for 8 hours.



(b) : Histogram of memory requirements of MIP solver for all dataset splits for trees of depth 4, solved for 8 hours.



(c) : Histogram of memory requirements of MIP solver for all dataset splits for trees of depth 3, solved for 8 hours. All solving processes fit under 45 GB.

Figure 3.23: Comparison of the memory requirements of the FCT model with depths 3 through 5. The mean memory requirement almost doubles from cca 23.9 GB for depth 3 through 51.1 GB for depth 4 to 77.3 GB for depth 5.

Interestingly, the proposition that leaf accuracy is higher for shallower trees seems valid even on commonly used CART trees. Figure 3.25 shows the comparison of CART and FCT models with depth 3. We see a significant increase in the leaf accuracy compared to depth 4. Nonetheless, CART models of depth 3 still do not outperform FCT models of depth 4, which are the base configurations.

This suggests that the deeper the tree we allow, the worse our leaf accuracy gets. This seems to be an issue of overfitting, and indeed, shallower trees are less likely to overfit due to limits on expressiveness. It might also be true that there is an ideal depth of a tree regarding the leaf accuracy.

Further, in Figure 3.24, we illustrate the relation between the depth of the tree and the amount of working memory used (in GB) to optimize the model for 8 hours. We have seen the influence of time on the amount of memory needed, so it is clear that the formulation size is not the only relevant parameter. Interestingly, Figure 3.24 does not show a superlinear dependence. However, that might be because of noise due to the presence of mere 3 data points in the plot.

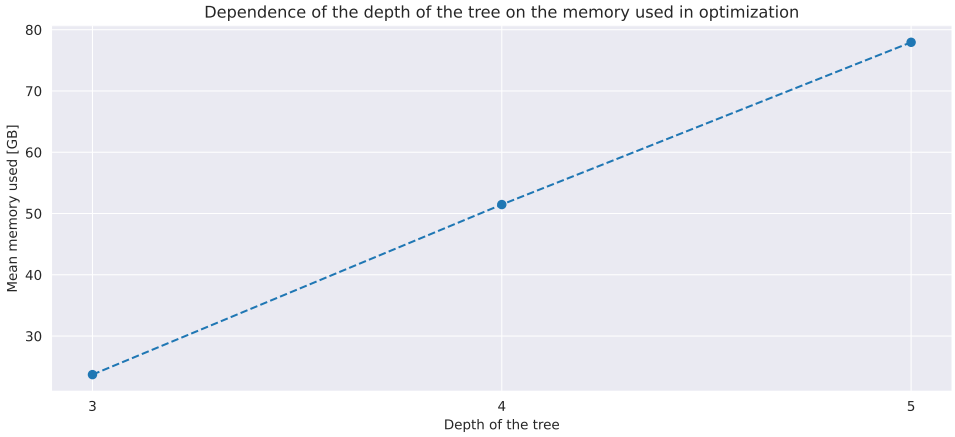


Figure 3.24: Dependence of mean memory requirements for MIP formulations on the depth of the tree.

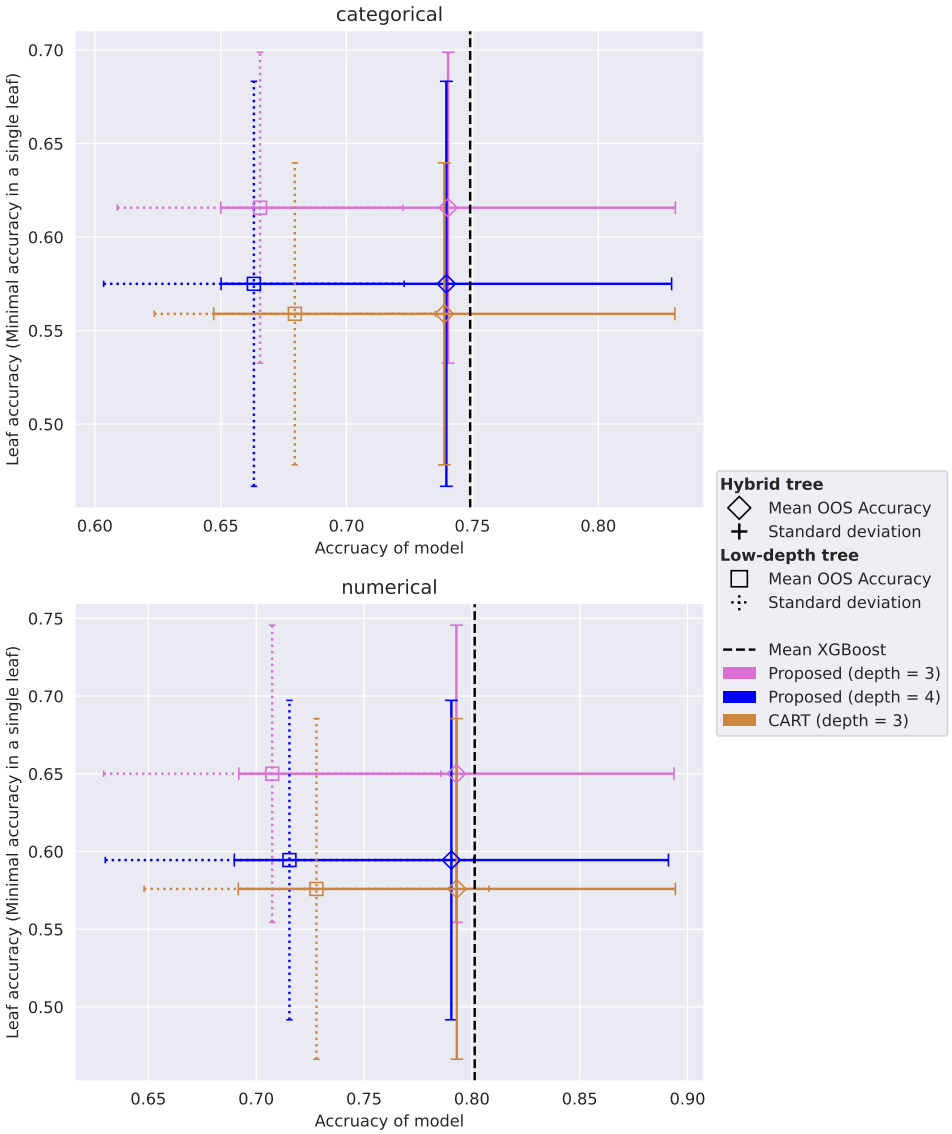


Figure 3.25: Detailed comparison of CART and FCT for depth 3. With the FCT of depth 4 as a reference. Shallower trees seem to have better leaf accuracy in general, possibly due to the lack of overfitting to the training data.

3.4.9 More input data

The 10,000 limit on training samples could be seen as arbitrary. The reason for it, other than this was the practice of Grinsztajn et al. [2022], is that we want our model to balance the size of the formulation and the capability of the formulated model. In other words, if we take a small amount of data, we are less likely to grasp the intricacies of the target variable distribution within the dataset. And if we take too many samples, we create a formulation that will not achieve good performance in a reasonable time.

Nonetheless, in a direct comparison of a model learned on a training dataset limited to 10,000 samples and 50,000 samples, we see that more data does not necessarily lead to a better model, see Figure 3.26. The 50,000 model is worse because of the too-demanding complexity of the formulation.

It improves the model accuracy, which is not surprising since each leaf obtains more samples. The comparison to XGBoost in this regard is unfortunately not fully reliable since the mean value was computed from performances of models trained on at most 10,000 samples.

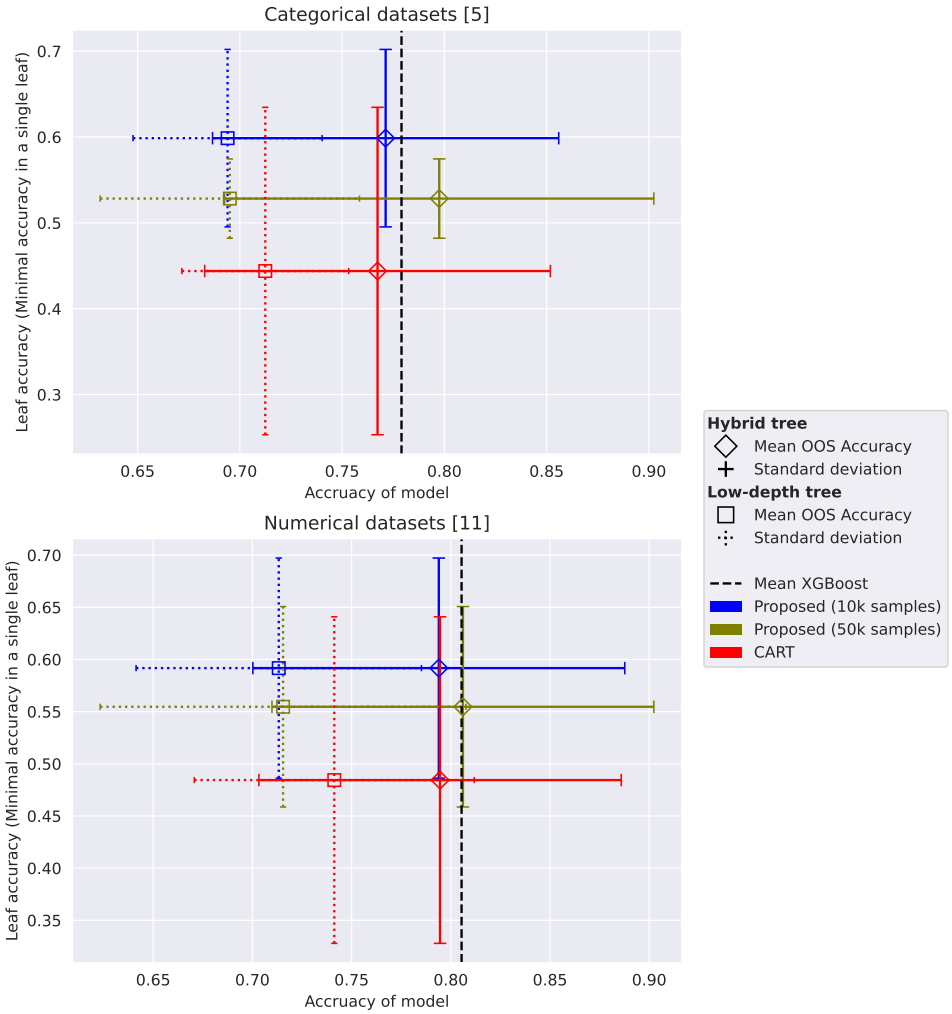


Figure 3.26: Comparison of a FCT trained on at most 10,000 and 50,000 data samples. We compare only on datasets where the constraint meant a change. Thus we omit datasets with enough samples to be covered by the main configuration of taking 80% of data as training samples and cropping it to at most 10,000. Essentially, this meant we excluded all datasets with less than 12,500 samples. The number of datasets is in square brackets. CART and XGBoost both use also only 10,000 samples, so the better mean score on the categorical dataset is to be taken with a grain of salt.

Chapter 4

Counterfactual optimization

Lastly, we introduce work regarding local explanations. We aim to create an optimizable function based on counterfactuals. Such a function would assess the ability of a model to generate sound, informative counterfactuals. The model could then be optimized with such a measure in mind. This could improve the local explainability of a model for which it is otherwise difficult to provide global explanations. Or if such global explanations are not representative enough or fair and valid, as discussed in the previous chapter.

As presented in Chapter 2, counterfactuals can be found optimally using Mixed-integer programming (MIP). We can optimize the proximity of our counterfactual x' to the original input x while limiting the search space only to samples that lead to a different result. This goal can be expressed in the following way:

$$\arg \min_{x'} dist(x, x') \tag{4.1}$$

$$\text{s.t. } \mathcal{N}(x) \neq \mathcal{N}(x') \tag{4.2}$$

$$x' \text{ remains a valid representation of a sample} \tag{4.3}$$

where $dist(\cdot, \cdot)$ is some distance function and $\mathcal{N}(x)$ represents a result of model $\mathcal{N}$ on sample x.

When using a MIP solver, we can generate all counterfactuals relatively close to the optimal value v. We can set an ϵ and use the solver to obtain all counterfactuals with their objective value less than $(1 + \epsilon) * v$ where v is the objective value of the best counterfactual. The objective function (4.1) is some distance to the factual x.

We search for all locally optimal samples within this range. We call the set of such samples C_x, where $C_x \subseteq C$. The set C is a union of all counterfactuals for every sample from $\mathcal{D}$

$$C = \bigcup_{x \in \mathcal{D}} C_x$$

where $\mathcal{D}$ is a set of input samples, in our case, usually a kind of validation set.

The actual MIP formulation consists of two main parts. The encoding of the input and encoding of the NN computation.

4.1 Encoding of the input

For the input encoding, we use one introduced by Russell [2019]. He proposes a clever way to incorporate features of mixed types. Consider a feature representing the number of months a loan payment was late. This has a continuous range, but optionally, one might consider extra categorical values. For example, to represent missing records or that no previous loan was recorded. These extra values can be useful for the model. One usually implements them using a one-hot encoding and a default value for the continuous range, usually 0.

The MIP encoding by Russell [2019] binds a one-hot vector and a continuous variable together using a binary variable as a switch. This construct is called a mixed polytope. We extended the original formulation to account for entirely categorical features correctly. Such features are added as a standard one-hot encoding to a binary vector giving up the continuous feature.

Following the work of Russell [2019], we minimize a weighted 1-norm of the difference of original and counterfactual vectors $\|x - x'\|_{1,w}$. Since the input sample x contains categorical, continuous, and possibly mixed features, we create our input vector $\bar{x}$ by performing a one-hot encoding for all categorical values. In the input vector, we have thus binary and continuous fields. We denote the sets of indices of binary and continuous fields by $\mathcal{B}$ and $\mathcal{C}$, respectively.

The general formulation is the following:

$$\arg\min \ (\boldsymbol{g}+\boldsymbol{h})^{\mathsf{T}}\boldsymbol{w}_{\mathcal{C}} + \sum_{j\in\mathcal{B}} s_j w_j \cdot (d'_j - \bar{x}_j) \tag{4.4}$$

$$\text{s.t. } \mathcal{N}(x) \neq \mathcal{N}(x') \tag{4.5}$$

$$0 \leq g_i, \quad \bar{x}_i - c_i \leq g_i \quad \forall i \in \mathcal{C} \tag{4.6}$$

$$0 \leq h_i, \quad c_i - \bar{x}_i \leq h_i \quad \forall i \in \mathcal{C} \tag{4.7}$$

conditions for mixed polytope and one hot encoding hold

$$d_i \in \{0,1\} \quad \forall i \in \mathcal{C} \tag{4.8}$$

where $\boldsymbol{g}$ is a vector of the negative deviation from the original values, $\boldsymbol{h}$ is the opposite, and $\boldsymbol{c}$ is the vector of the counterfactual values of the continuous fields.

In the objective function (4.4), $\boldsymbol{d}'$ is the vector of binary values of the counterfactual. The vector $\boldsymbol{w}_{\mathcal{C}}$ together with all parameters w_j are weights. These weights can be used to tune the generation of counterfactuals (e.g., to reflect actionability). We set them in a predefined way as inverse median absolute deviations in alignment with Russell [2019]. By setting $s_j \in \{-1, 1\}$ appropriately, the correct influence of each binary variable on the objective can be set.

$$s_j = \begin{cases} -1 & \text{if } \bar{x}_j = 1 \\ 1 & \text{otherwise} \end{cases}$$

because $(d'_j - x_j) < 0$ is true only if binary variable x_j is 1, and changing that should increase the objective value since the value has been changed in the counterfactual.

4.1.1 Mixed polytope

Mixed-polytope conditions are conditions that, for each mixed-type feature, bind its continuous and categorical values together. To stabilize the impact on the optimization, they ensure that the continuous value is set to an "anchor" value (typically 0) if one of the categorical values is chosen. This "anchor" value for each feature i of the input sample is denoted F_i. We denote c_i, the continuous value of feature i, that must lie in the range $[L_i, U_i]$.

We also introduce binary indicators $d_{i,j}$ for $j \in \{1, 2, \ldots, k_i\}$ values, where k_i is the number of categorical values of feature i. And one additional binary indicator, $d_{i,c}$ equal to 1 if feature i takes continuous value.

The conditions defining a mixed polytope for a feature i are thus:

$$\sum_{j=1}^{k_i} d_{i,j} + d_{i,c} = 1 \tag{4.9}$$

$$F_i - l_i + u_i = c_i \tag{4.10}$$

$$0 \leq l_i \leq (F_i - L_i)d_{i,c} \tag{4.11}$$

$$0 \leq u_i \leq (U_i - F_i)d_{i,c} \tag{4.12}$$

$$d_{i,j} \in \{0, 1\} \quad \forall j \in \{1, 2, \ldots, k_i\} \cup \{c\} \tag{4.13}$$

We encode fully continuous features in this way too.

4.1.2 Additions to the formulation

The formulation, as it stands, does not correctly account for categorical features. The original implementation mapping prefers the choice represented by value 0 (the first index of the one-hot encoding). It does so because of a simplification in the Python implementation, where it used the d_c variable for the first categorical value. The d_c otherwise represents only the change to use the continuous spectrum.

That decision variable was disregarded in the minimization because it would lead to increased penalization for switching to the continuous spectrum. Thus, instead of being represented as a change of 2 weighed binary variables, a change to option 0 meant only a change in 1, making it significantly "cheaper".

Another issue with this representation was the additional continuous variable without any meaning. Since all features were assumed to be continuous or mixed (continuous, with some categorical values for reasons of missing data, etc.), there was always a continuous variable. Any change in it was minimized, so in the original application to linear models, this posed no issues. However, applied to NNs, the objective function space is so varied that a change in such nonsensical continuous value can positively influence the result of the computation. For example, it was possible to obtain values like 0.2 for features where 1 encoded male and 0 encoded female. This happened because when $d_c = 1$ to represent that the value is "female", nothing was in the way of setting some non-zero value to $u_i = 0.2$ that was added together with $F_i = 0$ to the final odd value.

This implementation flaw led us to extend the encoding for pure categorical

features. For continuous and mixed continuous features, this works exactly the same way. But when a categorical feature is encoded, the decision variable for shifting to a continuous spectrum is replaced by a weighted binary decision variable and given as input to the model instead of the continuous variant. Such encoding is thus a standard one-hot encoding. For a fully categorical feature i with k_i possible values:

$$\sum_{j=1}^{k_i} d_{i,j} = 1 \tag{4.14}$$

$$d_{i,j} \in \{0,1\} \quad \forall j \in \{1,2,\ldots,k_i\} \tag{4.15}$$

4.2 Encoding of the neural network

For encoding the neural network, we used the MIP model by Fischetti and Jo [2018]. In our case, the weights of the neural network are parameters, not variables, since the MIP formulation does not optimize them.

We limit ourselves, as is common, only to Rectified Linear Unit (ReLU) activation function. We also assume fully connected linear (dense) layers only. Let our network have K such layers. A ReLU activated k-th layer for $k \in \{1,\ldots,K\}$ could be represented in computation as:

$$\boldsymbol{x}^k = ReLU(\boldsymbol{W}^k \boldsymbol{x}^{k-1} + \boldsymbol{b}^k)$$

where $\boldsymbol{W}^k$ and $\boldsymbol{b}^k$ are a weight matrix and a bias vector of k-th layer respectively, and $\boldsymbol{x}^k$ is the output of k-th layer. The $\boldsymbol{x}^0$ is thus the encoded input vector $\boldsymbol{x}$, and $\boldsymbol{x}^K$ is the output of the neural network.

Computation of a j-th neuron of k-th layer is encoded into MIP conditions as follows:

$$\sum_{i=1}^{n_{k-1}} w_{i,j}^k x_i^{k-1} + b_j^k = x_j^k - s_j^k \tag{4.16}$$

$$x_j^k, s_j^k \geq 0 \tag{4.17}$$

$$z_j^k \in \{0,1\} \tag{4.18}$$

$$z_j^k = 1 \implies x_j^k \leq 0 \tag{4.19}$$

$$z_j^k = 0 \implies s_j^k \leq 0 \tag{4.20}$$

where n_k is the number of neurons in k-th layer and n_0 is the size of input vector. out_i^k is the output value of i-th neuron in the k-th layer. w and b

correspond to the weights and biases of the neural network. s_j^k represents the possible negative output of the neuron before being discarded by ReLU. Notice that negative contributions of the previous layer $s_j^{(k-1)}$ are not present in the formulation.

The binary variable z_j^k secures the uniqueness of the solution, setting either the output to be influenced either negatively or positively by setting the other value to 0.

The implication condition (constraints (4.19) and (4.20)) is implemented in most modern solvers or can be implemented by a set of standard big-M conditions.

4.2.1 Modifications

Because our goal is to generate multiple closest solutions, we ran into problems with generating multiple solutions with the same counterfactuals. This is caused by a known problem with indicators that decide whether the output of a node is positive or negative. This indicator is loosely bound if neuron x_j^k outputs 0. The indicator z_j^k can be 0 or 1 in such a case. This generates multiple MIP solutions with the same inputs. An attempt to battle this was made by adding a constraint

$$z_j^k \leq s_j^k \cdot M \tag{4.21}$$

where M is an arbitrarily high number. The lowest possible value of s_j^k for your neural network will equal $1/M$. This is constrained by the numerical precision of the solver.

4.3 Counterfact encoding

Now that we have the formulation of the change in input and the model computation, we need only the condition to require the change in the output of the model.

Many variants of output functions could be modeled, and we refer the reader to the vast existing literature [e.g. Fischetti and Jo, 2018; Akutsu and

Nagamochi, 2019]. For our case, we present the case of binary classification and assume we decide based on whether the output of a single output neuron is a negative or positive number. This corresponds to a standard setting without sigmoid as the last activation function. Constraints for such a decision in our MIP formulation are then:

$$S \cdot x_1^K \geq \alpha \tag{4.22}$$

where S is the sign of the desired result, x_1^K is the single output of the network, and $\alpha \geq 0$ is a margin that can be used to introduce a margin to the decision. We usually choose $\alpha = 0$.

The complete formulation combines all presented constraints, depending on the type of features in the input, the number, and size of the layers, etc. The input vector $\boldsymbol{x}^0$ (equivalent to $\bar{x}$) is represented in MIP by correctly concatenating one hot binary indicator and continuous variables. We used Gurobi solver [Gurobi Optimization, LLC, 2023] to solve the model.

4.4 Proposed functions

Now, being able to generate sets of close counterfactuals, we can generate the counterfactuals for sets of validation samples and compare the distributions for each feature. Then we combine the values of all features into one value representing some notion of explainability.

We propose three novel functions to track the performance of a NN model with respect to the ability to produce valuable counterfactuals. The first two use standard Shannon entropy, and the third uses Wasserstein 2-distance, one of the most useful variants from the Wasserstein distance family [Villani, 2009].

4.4.1 Entropy-based functions

The first two measures are a straightforward usage of Shannon entropy. We first estimate the distribution of each feature, then compute its entropy. Then we compute the difference between the entropy of the feature in the generated counterfactuals and "factuals" (true data) that are classified as the same

class. Finally, we average the measures over the features to get a single value. Formally, we would write it as:

$$U_{e_1}(X) = \frac{1}{p}\sum_{i=1}^{p}(\mathcal{H}(C_X[i]) - \mathcal{H}(X[i])) \tag{4.23}$$

where X is a validation set, C_X is the set of generated counterfactuals for X, square brackets $[i]$ denote that we take the value of the feature i from the set of all features, p is the number of features, and $\mathcal{H}$ represents the computation of Shannon entropy of the distribution sampled by the values in the set.

The second proposed function is similar in that it uses entropy and difference, but now we compute the entropy last. We first compute the difference between a factual and its counterfactuals, that is, samples of a different class. Then we compute the entropy for each feature and average that, similarly to the first function. We again compute this for each target class separately. Formally,

$$U_{e_2}(X) = \frac{1}{p}\sum_{i=1}^{p}\mathcal{H}(\{d(x[i], C_x[i]) \mid \forall x \in X\}) \tag{4.24}$$

$$\text{s.t. } d(x, Y) = \{x - y \mid \forall y \in Y\} \tag{4.25}$$

where C_x is set of close counterfactuals of a single sample x and all other symbols are the same as for the $U_{e_1}(X)$.

The first function $U_{e_1}(X)$ should be minimized since our goal is to have the counterfactuals represent the original distribution as well as possible to generate valid and plausible explanations. In the second example, we wish to maximize the function because we want the counterfactual representatives to be generated differently for each sample. We want maximal diversity.

4.4.2 Wasserstein-based function

Wasserstein 2-distance is a standard metric used as a distance measure for probability distributions. Because Wasserstein distances require a metric space, it enables the user to set custom distances between certain features of the data.

We compute the utility function similarly to U_{e_1} (4.23), but instead of the difference between entropies, we compute the Wasserstein 2-distance between

the distributions directly:

$$U_{wd}(X) = \frac{1}{p}\sum_{i=1}^{p} \mathcal{W}_2(C_X[i], X[i]) \tag{4.26}$$

where all symbols have the same meaning as in previous formulations, and $\mathcal{W}_2$ is the Wasserstein 2-distance. Similarly to U_{e_1}, we would use this function to compare samples and counterfactuals of the same class and minimize this function.

The way these utility functions could be used is, for example, to get an "explainability" measure that could help the developer to see during training when their model is in the best conditions. In this way, these proposed measures could work quite similarly to the validation set error in the training process of Neural Networks.

The implementation of the whole MIP formulation with all of the optimizable functions is publicly available at:
`https://github.com/Epanemu/counterfactual_explanations`

Chapter 5

Conclusion

We have shown that not only that there is a place for Mixed-integer programming (MIP) in Machine learning (ML). We have described increasing interest in these applications and supported this interest by adding our own.

In Chapter 2 we went through some existing approaches of using MIP and direct MIP formulations of ML models. We described its use regarding NNs and decision trees in more detail. We described the importance of MIP in XAI, and the usefulness of tweaking the objective function to one's needs. We also summarized the definition, properties, and importance of counterfactual explanations. We pointed out many variants of MIP formulations of the counterfactual generation.

In Chapter 3, because the local explainability of the model's decisions is not enough in many cases, in Chapter 3 we took it upon ourselves to produce a MIP formulation that improves the interpretability of decision trees in general. By tackling the unfairness of accuracy distribution over leaves, we introduced the idea of fair explanation. With that, we proposed a MIP formulation of Classification trees that optimize the leaf accuracy. We call it Fair Classification Tree (FCT).

We performed extensive testing of FCT and showed our method to work well. Compared to the widely used CART, we improved the leaf accuracy by more than 11 percentage points on average. Within the experimentation, our method has shown some surprising results and made us question our own configuration. We outlined everything a user of the FCT model might need in order to decide for themselves which configuration suits them best.

In Chapter 4, we selected one of the many formulations for counterfactual generation and used it to generate a set of counterfactuals within some distance to the optimum. We added better support for the input features by including proper definitions for the pure categorical variables. We further proposed three utility functions that a model could optimize to achieve improved post-hoc explainability.

5.1 Furhter improvements

All goals of the thesis were completed, and the last of them was completed in the greatest detail. That being said, there are always more things to do.

Leaf accuracy oriented trees

While this area underwent a close inspection and many experiments were performed, there are also topics requiring further exploration.

The most significant topic is the trees of depth 3 outperforming trees of depth 4, which were selected as the reference configuration. This was a surprising result for us since in earlier testing trees of depth 4 seemed like the tipping point where sufficient depth met the computational tractability of the formulation. We were unfortunately not able to provide further comparing tests due to time limitations. Nonetheless, it is important for future work to investigate the relation between low depth and high leaf accuracy. There might be more reasons for the shallower trees outperforming the deeper ones other than the less complicated formulation and better performance of the MIP solver. It is certainly an exciting area waiting for further research.

The performance increase of the shallower tree also suggests that for deeper trees, similar results could be achieved given enough time. The implementation of efficiency improvements regarding the speed of the formulation solving might thus influence the ordering of the performance of the FCTs based on their depth.

Otherwise, it might be beneficial to do further testing of the intricacies of varying values of other hyperparameters, especially the minimum of samples in a leaf ($N_{\min}$).

However, there is only so much time in one student's life. We believe to have done a fair attempt at comparison of our proposed FCT formulation to other methods as well as to various other configurations of FCT.

Counterfactual optimizers

There are multiple things requiring future work regarding the counterfactual optimization functions. A couple of obstacles are in the way of performing relevant experiments with the functions. For those reasons and the lack of interest at that time, no in-depth experiments were performed.

The question of how would one measure the quality of a counterfactual explanation is a non-trivial one. The measure is influenced by subjective opinions about explanations. It would thus make sense to require diversity, but we also should exclude non-actionable features from this measure. Many such concerns are needed to be taken into account.

Another issue is more technical, it is related to the counterfactual generation and the issue of generating multiple same counterfactuals because of unfortunate encoding of ReLU. This issue was tackled by incorporating a constraint, which limits the expressive capabilities of the network or reaches the point of numerical instability, depending on the tightness of the bound. This makes the generation less reliable to produce for example a fixed number of closest counterfactuals. A possible solution is to use a max() function instead. This leads to more stable results, but it also further decreases speed because of non-linearity.

Even without the max(), the utility functions are lacking in speed. The counterfactual generation takes quite a bit of time. The time demand also increases with the increase in the relative range to the optimum. This measure thus can take an order of magnitude more time than standard validation measures performed during the training.

The next good question is the landscape of these functions. It is possible to search the landscape efficiently? How would one combine such a measure with standard accuracy measures? We leave many questions unanswered to stimulate further work on this topic.

5.2 Resources

All code and aggregated testing data are available in the respective repositories of the 2 projects.

Regarding the FCT model and the methods connected to it, this is implemented and presented at:
`https://github.com/Epanemu/FCT`

For the optimization functions using counterfactual explanations generation, the repository can be found at:
`https://github.com/Epanemu/counterfactual_explanations/`

Appendix A

Acronyms

AI Artificial Intelligence. 1–3, 5

CART Classification and Regression Trees algorithm. vii, viii, 13–15, 31–39, 41–46, 49, 53–56, 58, 60, 62, 63, 65, 77

EM Expectation–maximization algorithm. 6

FCT Fair Classification Tree. iv, v, vii, viii, 14, 15, 17, 19, 21, 23, 25, 27–30, 32–37, 42, 43, 47, 48, 50, 57, 59, 61–63, 65, 77–80, 91, 95, 96

MIP Mixed-integer programming. iv, v, 2, 3, 5–17, 19–25, 27, 28, 32, 33, 41, 61, 62, 67, 68, 71–73, 75, 77, 78

ML Machine learning. iv, v, 3, 5–12, 14, 77

NN Artifical Neural network. 3, 6–8, 10, 11, 68, 70, 73, 77

OCT Optimal Classification Tree. v, vii, 7, 14, 19–21, 24, 25, 27–29, 32, 36, 44, 45, 56–58

PCA Principal Component Analysis. 9, 10

ReLU Rectified Linear Unit. 71, 72, 79

SVM Support vector machine. 7, 9

XAI Explainable Artificial Intelligence. 1, 34, 77

Appendix B

Bibliography

Amina Adadi and Mohammed Berrada. Peeking Inside the Black-Box: A Survey on Explainable Artificial Intelligence (XAI). *IEEE Access*, 6:52138–52160, 2018. ISSN 2169-3536. doi: 10.1109/ACCESS.2018.2870052.

Sina Aghaei, Andres Gomez, and Phebe Vayanos. Learning Optimal Classification Trees: Strong Max-Flow Formulations, May 2020.

Sina Aghaei, Andrés Gómez, and Phebe Vayanos. Strong Optimal Classification Trees, February 2022.

Tatsuya Akutsu and Hiroshi Nagamochi. A Mixed Integer Linear Programming Formulation to Artificial Neural Networks. In *Proceedings of the 2nd International Conference on Information Science and Systems*, ICISS '19, pages 215–220, New York, NY, USA, March 2019. Association for Computing Machinery. ISBN 978-1-4503-6103-3. doi: 10.1145/3322645.3322683.

Laura A. Albert. A mixed-integer programming model for identifying intuitive ambulance dispatching policies. *Journal of the Operational Research Society*, 0(0):1–12, November 2022. ISSN 0160-5682. doi: 10.1080/01605682.2022.2139646.

Zacharie Alès, Valentine Huré, and Amélie Lambert. New optimization models for optimal classification trees, November 2022.

Brandon Alston, Hamidreza Validi, and Illya V. Hicks. Mixed integer linear optimization formulations for learning optimal binary classification trees, June 2022.

Ross Anderson, Joey Huchette, Will Ma, Christian Tjandraatmadja, and Juan Pablo Vielma. Strong mixed-integer programming formulations for

trained neural networks. *Mathematical Programming*, 183(1):3–39, September 2020. ISSN 1436-4646. doi: 10.1007/s10107-020-01474-5.

Hari Bandi, Dimitris Bertsimas, and Rahul Mazumder. Learning a Mixture of Gaussians via Mixed-Integer Optimization. *Informs Journal on Optimization*, April 2019. doi: 10.1287/ijoo.2018.0009.

Mark Bartlett and James Cussens. Advances in Bayesian Network Learning using Integer Programming, March 2015.

Osbert Bastani, Carolyn Kim, and Hamsa Bastani. Interpreting Blackbox Models via Model Extraction, January 2019.

Marcelo Antônio Mendes Bastos, Humberto Brandão César de Oliveira, and Cristiano Arbex Valle. Ensemble pruning via an integer programming approach with diversity constraints, May 2022.

P. Bertolazzi, G. Felici, P. Festa, G. Fiscon, and E. Weitschek. Integer programming models for feature selection: New extensions and a randomized solution algorithm. *European Journal of Operational Research*, 250(2): 389–399, April 2016. ISSN 0377-2217. doi: 10.1016/j.ejor.2015.09.051.

Dimitris Bertsimas and Jack Dunn. Optimal classification trees. *Machine Learning*, 106(7):1039–1082, July 2017. ISSN 1573-0565. doi: 10.1007/s10994-017-5633-9.

Dimitris Bertsimas and Rahul Mazumder. Least quantile regression via modern optimization. *The Annals of Statistics*, 42(6):2494–2525, December 2014. ISSN 0090-5364, 2168-8966. doi: 10.1214/14-AOS1223.

Dimitris Bertsimas, Angela King, and Rahul Mazumder. Best subset selection via a modern optimization lens. *The Annals of Statistics*, 44(2):813–852, April 2016. ISSN 0090-5364, 2168-8966. doi: 10.1214/15-AOS1388.

Victor Blanco, Alberto Japón, and Justo Puerto. Robust optimal classification trees under noisy labels. *Advances in Data Analysis and Classification*, 16 (1):155–179, 2022. ISSN 18625355. doi: 10.1007/s11634-021-00467-2.

L. Breiman, J. Friedman, C.J. Stone, and R.A. Olshen. *Classification and Regression Trees*. Taylor & Francis, 1984. ISBN 978-0-412-04841-8.

Miguel Á Carreira-Perpiñán and Suryabhan Singh Hada. Counterfactual Explanations for Oblique Decision Trees:Exact, Efficient Algorithms. *Proceedings of the AAAI Conference on Artificial Intelligence*, 35(8):6903–6911, May 2021. ISSN 2374-3468. doi: 10.1609/aaai.v35i8.16851.

Tianqi Chen and Carlos Guestrin. XGBoost: A scalable tree boosting system. In *Proceedings of the 22nd ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, KDD '16, pages 785–794, New York, NY, USA, 2016. Association for Computing Machinery. ISBN 978-1-4503-4232-2. doi: 10.1145/2939672.2939785.

Ziyi Chen, Patrick De Causmaecker, and Yajie Dou. A combined mixed integer programming and deep neural network-assisted heuristics algorithm for the nurse rostering problem. *Applied Soft Computing*, 136:109919, March 2023. ISSN 1568-4946. doi: 10.1016/j.asoc.2022.109919.

Yu-Neng Chuang, Guanchu Wang, Fan Yang, Zirui Liu, Xuanting Cai, Mengnan Du, and Xia Hu. Efficient XAI Techniques: A Taxonomic Survey, February 2023.

IBM ILOG Cplex. V12. 1: User's manual for CPLEX. *International Business Machines Corporation*, 46(53):157, 2009.

James Cussens. Bayesian network learning with cutting planes. In *Proceedings of the Twenty-Seventh Conference on Uncertainty in Artificial Intelligence*, UAI'11, pages 153–160, Arlington, Virginia, USA, July 2011. AUAI Press. ISBN 978-0-9749039-7-2.

DARPA. Broad agency announcement explainable artificial intelligence (xai). Technical report, DARPA, August 2016.

Ayhan Demiriz and Kristin P. Bennett. Optimization Approaches to Semi-Supervised Learning. In Michael C. Ferris, Olvi L. Mangasarian, and Jong-Shi Pang, editors, *Complementarity: Applications, Algorithms and Extensions*, Applied Optimization, pages 121–141. Springer US, Boston, MA, 2001. ISBN 978-1-4757-3279-5. doi: 10.1007/978-1-4757-3279-5_6.

Santanu S. Dey, R. Mazumder, and Guanyi Wang. A convex integer programming approach for optimal sparse PCA. *arXiv: Optimization and Control*, October 2018.

Federico D'Onofrio, Giorgio Grani, Marta Monaci, and Laura Palagi. Margin Optimal Classification Trees, January 2023.

Vivek Dua. A mixed-integer programming approach for optimal configuration of artificial neural networks. *Chemical Engineering Research and Design*, 88 (1):55–60, January 2010. ISSN 0263-8762. doi: 10.1016/j.cherd.2009.06.007.

Jacob Feldman. Minimization of Boolean complexity in human concept learning. *Nature*, 407(6804):630–633, October 2000. ISSN 0028-0836, 1476-4687. doi: 10.1038/35036586.

Aaron Ferber, Bryan Wilder, Bistra Dilkina, and Milind Tambe. MIPaaL: Mixed Integer Program as a Layer. *Proceedings of the AAAI Conference on Artificial Intelligence*, 34(02):1504–1511, April 2020. ISSN 2374-3468. doi: 10.1609/aaai.v34i02.5509.

Rosa Figueiredo and Gisele Moura. Mixed integer programming formulations for clustering problems related to structural balance. *Social Networks*, 35(4): 639–651, October 2013. ISSN 0378-8733. doi: 10.1016/j.socnet.2013.09.002.

Matteo Fischetti and Jason Jo. Deep neural networks and mixed integer linear optimization. *Constraints*, 23(3):296–309, July 2018. ISSN 1572-9354. doi: 10.1007/s10601-018-9285-6.

Alexandre M. Florio, Pedro Martins, Maximilian Schiffer, Thiago Serra, and Thibaut Vidal. Optimal Decision Diagrams for Classification, May 2022.

Fred Glover. Future paths for integer programming and links to artificial intelligence. *Computers & Operations Research*, 13(5):533–549, January 1986. ISSN 0305-0548. doi: 10.1016/0305-0548(86)90048-1.

Leo Grinsztajn, Edouard Oyallon, and Gael Varoquaux. Why do tree-based models still outperform deep learning on typical tabular data? In S. Koyejo, S. Mohamed, A. Agarwal, D. Belgrave, K. Cho, and A. Oh, editors, *Advances in Neural Information Processing Systems*, volume 35, pages 507–520. Curran Associates, Inc., 2022.

M. Grötschel and Y. Wakabayashi. A cutting plane algorithm for a clustering problem. *Mathematical Programming*, 45(1):59–96, August 1989. ISSN 1436-4646. doi: 10.1007/BF01589097.

Riccardo Guidotti. Counterfactual explanations and how to find them: Literature review and benchmarking. *Data Mining and Knowledge Discovery*, April 2022. ISSN 1573-756X. doi: 10.1007/s10618-022-00831-6.

Gurobi Optimization, LLC. Gurobi optimizer reference manual, 2023.

Alvin X Han, Edyth Parker, Frits Scholer, Sebastian Maurer-Stroh, and Colin A Russell. Phylogenetic Clustering by Linear Integer Programming (PhyCLIP). *Molecular Biology and Evolution*, 36(7):1580–1595, July 2019. ISSN 0737-4038. doi: 10.1093/molbev/msz053.

Tim Head, Manoj Kumar, Holger Nahrstaedt, Gilles Louppe, and Iaroslav Shcherbatyi. Scikit-optimize/scikit-optimize. Zenodo, October 2021.

Krystal Hu. ChatGPT sets record for fastest-growing user base - analyst note. *Reuters*, February 2023.

Itay Hubara, Yury Nahshan, Yair Hanani, Ron Banner, and Daniel Soudry. Improving Post Training Neural Quantization: Layer-wise Calibration and Integer Programming, December 2020.

Laurent Hyafil and Ronald L. Rivest. Constructing optimal binary decision trees is NP-complete. *Information Processing Letters*, 5(1):15–17, May 1976. ISSN 0020-0190. doi: 10.1016/0020-0190(76)90095-8.

Nathanael Jo, Sina Aghaei, Andrés Gómez, and Phebe Vayanos. Learning Optimal Fair Classification Trees: Trade-offs Between Interpretability, Fairness, and Accuracy, May 2023.

Julia Angwin, Jeff Larson, Lauren Kirchner, and Surya Mattu. Machine Bias. https://www.propublica.org/article/machine-bias-risk-assessments-in-criminal-sentencing, May 2016.

Kentaro Kanamori, Takuya Takagi, Ken Kobayashi, and Hiroki Arimura. DACE: Distribution-Aware Counterfactual Explanation by Mixed-Integer Linear Optimization. In *Proceedings of the Twenty-Ninth International Joint Conference on Artificial Intelligence*, pages 2855–2862, Yokohama, Japan, July 2020. International Joint Conferences on Artificial Intelligence Organization. ISBN 978-0-9992411-6-5. doi: 10.24963/ijcai.2020/395.

Kentaro Kanamori, Takuya Takagi, Ken Kobayashi, Yuichi Ike, Kento Uemura, and Hiroki Arimura. Ordered Counterfactual Explanation by Mixed-Integer Linear Optimization: 35th AAAI Conference on Artificial Intelligence, AAAI 2021. *35th AAAI Conference on Artificial Intelligence, AAAI 2021*, pages 11564–11574, 2021.

Elias Khalil, Pierre Le Bodic, Le Song, George Nemhauser, and Bistra Dilkina. Learning to Branch in Mixed Integer Programming. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 30, February 2016. doi: 10.1609/aaai.v30i1.10080.

Matthias König, Holger H. Hoos, and Jan N. van Rijn. Speeding up neural network robustness verification via algorithm configuration and an optimised mixed integer linear programming solver portfolio. *Machine Learning*, 111(12):4565–4584, December 2022. ISSN 1573-0565. doi: 10.1007/s10994-022-06212-w.

Simge Küçükyavuz, A. Shojaie, Hasan Manzour, and Linchuan Wei. Consistent Second-Order Conic Integer Programming for Learning Bayesian Networks. *ArXiv*, May 2020.

Jannis Kurtz. Ensemble Methods for Robust Support Vector Machines using Integer Programming, March 2022.

Andrew Kusiak. Analysis of integer programming formulations of clustering problems. *Image and Vision Computing*, 2(1):35–40, February 1984. ISSN 0262-8856. doi: 10.1016/0262-8856(84)90042-8.

Eduardo Laber, Lucas Murtinho, and Felipe Oliveira. Shallow decision trees for explainable k-means clustering. *Pattern Recognition*, 137:109239, May 2023. ISSN 0031-3203. doi: 10.1016/j.patcog.2022.109239.

Sebastián Maldonado, Juan Pérez, Richard Weber, and Martine Labbé. Feature selection for Support Vector Machines via Mixed Integer Linear Programming. *Information Sciences*, 279:163–175, September 2014. ISSN 0020-0255. doi: 10.1016/j.ins.2014.03.110.

Rahul Mazumder and Peter Radchenko. The Discrete Dantzig Selector: Estimating Sparse Linear Models via Mixed Integer Linear Optimization.

IEEE Transactions on Information Theory, 63(5):3053–3075, May 2017. ISSN 1557-9654. doi: 10.1109/TIT.2017.2658023.

M. Menickelly, O. Günlük, J. Kalagnanam, and K. Scheinberg. Optimal Generalized Decision Trees via Integer Programming. *ArXiv*, December 2016.

Atsushi Miyauchi, Tomohiro Sonobe, and Noriyoshi Sukegawa. Exact Clustering via Integer Programming and Maximum Satisfiability. *Proceedings of the AAAI Conference on Artificial Intelligence*, 32(1), April 2018. ISSN 2374-3468. doi: 10.1609/aaai.v32i1.11519.

Kiarash Mohammadi, Amir-Hossein Karimi, Gilles Barthe, and Isabel Valera. Scaling Guarantees for Nearest Counterfactual Explanations. In *Proceedings of the 2021 AAAI/ACM Conference on AI, Ethics, and Society*, AIES '21, pages 177–187, New York, NY, USA, July 2021. Association for Computing Machinery. ISBN 978-1-4503-8473-5. doi: 10.1145/3461702.3462514.

Axel Parmentier and Thibaut Vidal. Optimal Counterfactual Explanations in Tree Ensembles. In *Proceedings of the 38th International Conference on Machine Learning*, pages 8422–8431. PMLR, July 2021.

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

Eduardo Queiroga, Anand Subramanian, Rosa Figueiredo, and Yuri Frota. Integer programming formulations and efficient local search for relaxed correlation clustering. *Journal of Global Optimization*, 81(4):919–966, December 2021. ISSN 1573-2916. doi: 10.1007/s10898-020-00989-7.

Cynthia Rudin. Stop explaining black box machine learning models for high stakes decisions and use interpretable models instead. *Nature Machine Intelligence*, 1(5):206–215, May 2019. ISSN 2522-5839. doi: 10.1038/s42256-019-0048-x.

Cynthia Rudin and Şeyda Ertekin. Learning customized and optimized lists of rules with mathematical programming. *Mathematical Programming Computation*, 10(4):659–702, December 2018. ISSN 1867-2957. doi: 10.1007/s12532-018-0143-8.

Chris Russell. Efficient Search for Diverse Coherent Explanations. In *Proceedings of the Conference on Fairness, Accountability, and Transparency*, FAT* '19, pages 20–28, New York, NY, USA, January 2019. Association for Computing Machinery. ISBN 978-1-4503-6125-5. doi: 10.1145/3287560.3287569.

Waddah Saeed and Christian Omlin. Explainable AI (XAI): A Systematic Meta-Survey of Current Challenges and Future Opportunities, November 2021.

Burcu Sağlam, F. Sibel Salman, Serpil Sayın, and Metin Türkay. A mixed-integer programming approach to the clustering problem with an application in customer segmentation. *European Journal of Operational Research*, 173 (3):866–879, September 2006. ISSN 0377-2217. doi: 10.1016/j.ejor.2005.04.048.

Hossein Shahrabi Farahani and Jens Lagergren. Learning Oncogenetic Networks by Reducing to Mixed Integer Linear Programming. *PLoS ONE*, 8 (6):e65773, June 2013. ISSN 1932-6203. doi: 10.1371/journal.pone.0065773.

Hasan Sildir and Erdal Aydin. A Mixed-Integer linear programming based training and feature selection method for artificial neural networks using piece-wise linear approximations. *Chemical Engineering Science*, 249: 117273, February 2022. ISSN 0009-2509. doi: 10.1016/j.ces.2021.117273.

Foo Yoon-Pin Simon and Takefuji. Integer linear programming neural networks for job-shop scheduling. In *IEEE 1988 International Conference on Neural Networks*, pages 341–348 vol.2, July 1988. doi: 10.1109/ICNN.1988.23946.

Yunhao Tang, Shipra Agrawal, and Yuri Faenza. Reinforcement learning for integer programming: Learning to cut. In Hal Daumé III and Aarti Singh, editors, *Proceedings of the 37th International Conference on Machine Learning*, volume 119 of *Proceedings of Machine Learning Research*, pages 9367–9376. PMLR, July 2020.

Tomas Thorbjarnarson and Neil Yorke-Smith. On Training Neural Networks with Mixed Integer Programming. In *IJCAI-PRICAI'20 Workshop on Data Science Meets Optimisation*, 2021.

Vincent Tjeng, Kai Y. Xiao, and Russ Tedrake. Evaluating Robustness of Neural Networks with Mixed Integer Programming. In *International Conference on Learning Representations*, November 2017.

Fadime Üney and Metin Türkay. A mixed-integer programming approach to multi-class data classification problem. *European Journal of Operational Research*, 173(3):910–920, September 2006. ISSN 0377-2217. doi: 10.1016/j.ejor.2005.04.049.

Berk Ustun, Alexander Spangher, and Yang Liu. Actionable Recourse in Linear Classification. In *Proceedings of the Conference on Fairness, Accountability, and Transparency*, pages 10–19, Atlanta GA USA, January 2019. ACM. ISBN 978-1-4503-6125-5. doi: 10.1145/3287560.3287566.

Sicco Verwer and Yingqian Zhang. Learning Decision Trees with Flexible Constraints and Objectives Using Integer Optimization. In Domenico Salvagnin and Michele Lombardi, editors, *Integration of AI and OR Techniques in Constraint Programming*, Lecture Notes in Computer Science, pages 94–103, Cham, 2017. Springer International Publishing. ISBN 978-3-319-59776-8. doi: 10.1007/978-3-319-59776-8_8.

Sicco Verwer and Yingqian Zhang. Learning Optimal Classification Trees Using a Binary Linear Program Formulation. *Proceedings of the AAAI Conference on Artificial Intelligence*, 33(01):1625–1632, July 2019. ISSN 2374-3468, 2159-5399. doi: 10.1609/aaai.v33i01.33011624.

Sicco Verwer, Yingqian Zhang, and Qing Chuan Ye. Auction optimization using regression trees and linear models as integer programs. *Artificial Intelligence*, 244:368–395, March 2017. ISSN 0004-3702. doi: 10.1016/j.artint.2015.05.004.

Cédric Villani. The Wasserstein distances. In Cédric Villani, editor, *Optimal Transport: Old and New*, Grundlehren Der Mathematischen Wissenschaften, pages 93–111. Springer, Berlin, Heidelberg, 2009. ISBN 978-3-540-71050-9. doi: 10.1007/978-3-540-71050-9_6.

Hrishikesh D. Vinod. Integer Programming and the Theory of Grouping. *Journal of the American Statistical Association*, 64(326):506–519, June 1969. ISSN 0162-1459. doi: 10.1080/01621459.1969.10500990.

Daniël Vos and Sicco Verwer. Robust Optimal Classification Trees against Adversarial Examples. *Proceedings of the AAAI Conference on Artificial Intelligence*, 36(8):8520–8528, June 2022. ISSN 2374-3468. doi: 10.1609/aaai.v36i8.20829.

Thomas Vossen, Michael Ball, Amnon Lotem, and Dana Nau. On the use of integer programming models in AI planning. In *Proceedings of the 16th International Joint Conference on Artifical Intelligence - Volume 1*, IJCAI'99, pages 304–309, San Francisco, CA, USA, July 1999. Morgan Kaufmann Publishers Inc.

Jiayi Zhang, Chang Liu, Xijun Li, Hui-Ling Zhen, Mingxuan Yuan, Yawen Li, and Junchi Yan. A survey for solving mixed integer programming via machine learning. *Neurocomputing*, 519(C):205–217, January 2023. ISSN 0925-2312. doi: 10.1016/j.neucom.2022.11.024.

Zhi-Hua Zhou. *Machine Learning*. Springer Nature, August 2021. ISBN 9789811519673.

Zhi-Hua Zhou and Zhao-Qian Chen. Hybrid decision tree. *Knowledge-Based Systems*, 15(8):515–528, November 2002. ISSN 0950-7051. doi: 10.1016/S0950-7051(02)00038-2.

Jianshen Zhu, Chenxi Wang, Aleksandar Shurbevski, Hiroshi Nagamochi, and Tatsuya Akutsu. A Novel Method for Inference of Chemical Compounds of Cycle Index Two with Desired Properties Based on Artificial Neural Networks and Integer Programming. *Algorithms*, 13(5):124, May 2020. ISSN 1999-4893. doi: 10.3390/a13050124.

Appendix C

Supplementary material

C.1 Dataset descriptions

For the experiments on FCT, we used the classification part of the datasets from the benchmark of mid-sized tabular data, created by [Grinsztajn et al., 2022]. The datasets, with their properties, are listed in Table C.1. Training sets contained 80% of the total amount of samples or 10 000 samples, whichever was higher. This constraint affects 16 out of the 23 total datasets, though some only minimally. The affected datasets have the number of samples in the Table C.1 in bold.

In tests, we used 10 random seeds that determined the train-test splits, of each dataset. Datasets are split into two kinds. Categorical and numerical. Categorical are those that contain at least one categorical feature. Numerical datasets have no categorical features. Four numerical datasets are the same as categorical datasets but with the categorical features removed (`covertype`, `default-of-credit-card-clients`, `electricity`, `eye_movements`). Only datasets without missing features and with sufficient complexity are contained in the benchmark. For a more detailed methodology of how the datasets were selected, we refer the reader to the original paper. [Grinsztajn et al., 2022]

categorical datasets	# samples	# features	# classes
albert	**58252**	31	2
compas-two-years	4966	11	2
covertype	**423680**	54	2
default-of-credit-card-clients	**13272**	21	2
electricity	**38474**	8	2
eye_movements	7608	23	2
road-safety	**111762**	32	2
numerical datasets	# samples	# features	# classes
bank-marketing	10578	7	2
Bioresponse	3434	419	2
california	**20634**	8	2
covertype	**566602**	32	2
credit	**16714**	10	2
default-of-credit-card-clients	**13272**	20	2
Diabetes130US	**71090**	7	2
electricity	**38474**	7	2
eye_movements	7608	20	2
Higgs	**940160**	24	2
heloc	10000	22	2
house_16H	**13488**	16	2
jannis	**57580**	54	2
MagicTelescope	**13376**	10	2
MiniBooNE	**72998**	50	2
pol	10082	26	2

Table C.1: Listed classification datasets of the tabular benchmark. Train sets contained 80% of the total amount of samples or 10 000 samples, whichever was lower. 16 affected datasets have the number of samples in bold.

C.2 Hyperparameter search distributions

We needed to optimize hyperparameters for some quickly trainable models. We used Bayesian hyperparameter search for that purpose.

Extending XGBoost models

For the hyperparameter search of XGBoost models in leaves, we used the distributions listed in Table C.2. The parameters are almost all the same as used by Grinsztajn et al. [2022]. Only the number of estimators and maximal

depth were more constrained to account for the fewer samples available for training.

The Bayesian optimization was run for 50 iterations, with 3-fold cross-validation in every leaf that contained enough points to perform the optimization. The same process was used to extend all tested methods.

Parameter name	**Distribution** [range (inclusive)]
Max depth	UniformInteger [1, 7]
Number of estimators	UniformInteger [10, 500]
Min child weight	LogUniformInteger [1, 1e2]
Learning rate	Uniform [1e-5, 0.7]
Subsample	Uniform [0.5, 1]
Col sample by level	Uniform [0.5, 1]
Col sample by tree	Uniform [0.5, 1]
Gamma	LogUniform [1e-8, 7]
Alpha	LogUniform [1e-8, 1e2]
Lambda	LogUniform [1, 4]

Table C.2: Distributions of hyperparameters of extending XGBoost models in leaves. These were used in the Bayesian hyperparameter search in each leaf separately. All distributions except Max depth and Number of estimators are the same as in Grinsztajn et al. [2022]. The two different distributions were made smaller to improve the optimization time and to account for lower amounts of data.

In leaves with an insufficient amount of samples to perform the cross-validation (less than 3 samples of at least one class in our case), we train a model with a single tree of max depth 5.

CART models

For the hyperparameter optimization of CART models, we also used Bayesian search, with the distributions shown in the table C.3.

The search was run for 100 iterations, with 5-fold cross-validation on the same training data sets as our model. After this search, the best hyperparameters were used to train the model on the full training data. The resulting tree was reduced and every leaf was extended by an XGBoost model in the same way as our models.

In comparisons in Section 3.4.8, we optimized a deeper variant of the CART.

Parameter name	**Distribution** [range (inclusive)]
Max depth	UniformInteger [2, 4]
Min samples split	UniformInteger [2, 60]
Min samples leaf	UniformInteger [1, 60]
Max leaf nodes	UniformInteger [8, 16]
Min impurity decrease	Uniform [0, 0.2]

Table C.3: Distributions of hyperparameters of CART models used to compare to our method.

	Leaf Accuracy	
categorical datasets	CART	Proposed
albert	0.4674	**0.5706**
compas-two-years	0.4631	**0.5711**
covertype	0.5166	**0.7071**
default-of-credit-card-clients	0.3346	**0.5246**
electricity	0.5404	**0.6250**
eye_movements	**0.4356**	0.4109
road-safety	0.5228	**0.6158**
Mean rank	1.8571	**1.1429**

Table C.4: Categorical datasets. Mean leaf accuracies of models on out-of-sample data and average ranks.

The process was the same, except for initial distributions of hyperparameters for Max depth and Max leaf nodes. Those were UniformInteger [2, 20] and UniformInteger [8, 512] respectively.

C.3 Detailed results

We also provide the full results for each dataset. Figures C.1 and C.2 are decomposed variants of Figure 3.6 for categorical and numerical datasets respectively. We also provide exact results for leaf accuracy, in Tables C.4 and C.6, plus the exact results for hybrid tree accuracy, in Tables C.5 and C.7.

The detailed results show that the proposed model outperforms the CART model in both accuracy measures on almost all datasets and has comparable accuracy to XGBoost.

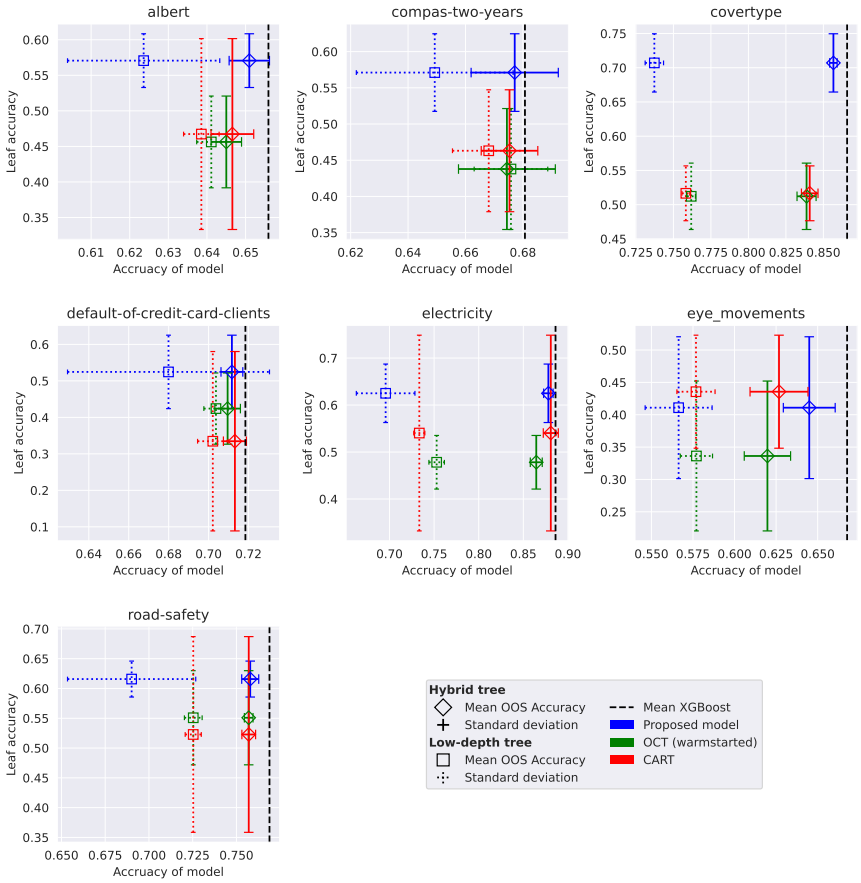


Figure C.1: Detailed performance comparison FCT model on individual categorical datasets.

	Hybrid-tree Accuracy		
categorical datasets	CART	Proposed	XGBoost
albert	0.6466	0.6510	**0.6559**
compas-two-years	0.6754	0.6772	**0.6807**
covertype	0.8409	0.8567	**0.8658**
default-of-credit-card-clients	0.7132	0.7117	**0.7184**
electricity	0.8808	0.8781	**0.8861**
eye_movements	0.6267	0.6449	**0.6677**
road-safety	0.7570	0.7579	**0.7689**
Mean rank	2.7143	2.2857	**1.0000**

Table C.5: Categorical datasets. Mean hybrid tree accuracies of models on out-of-sample data and average ranks.

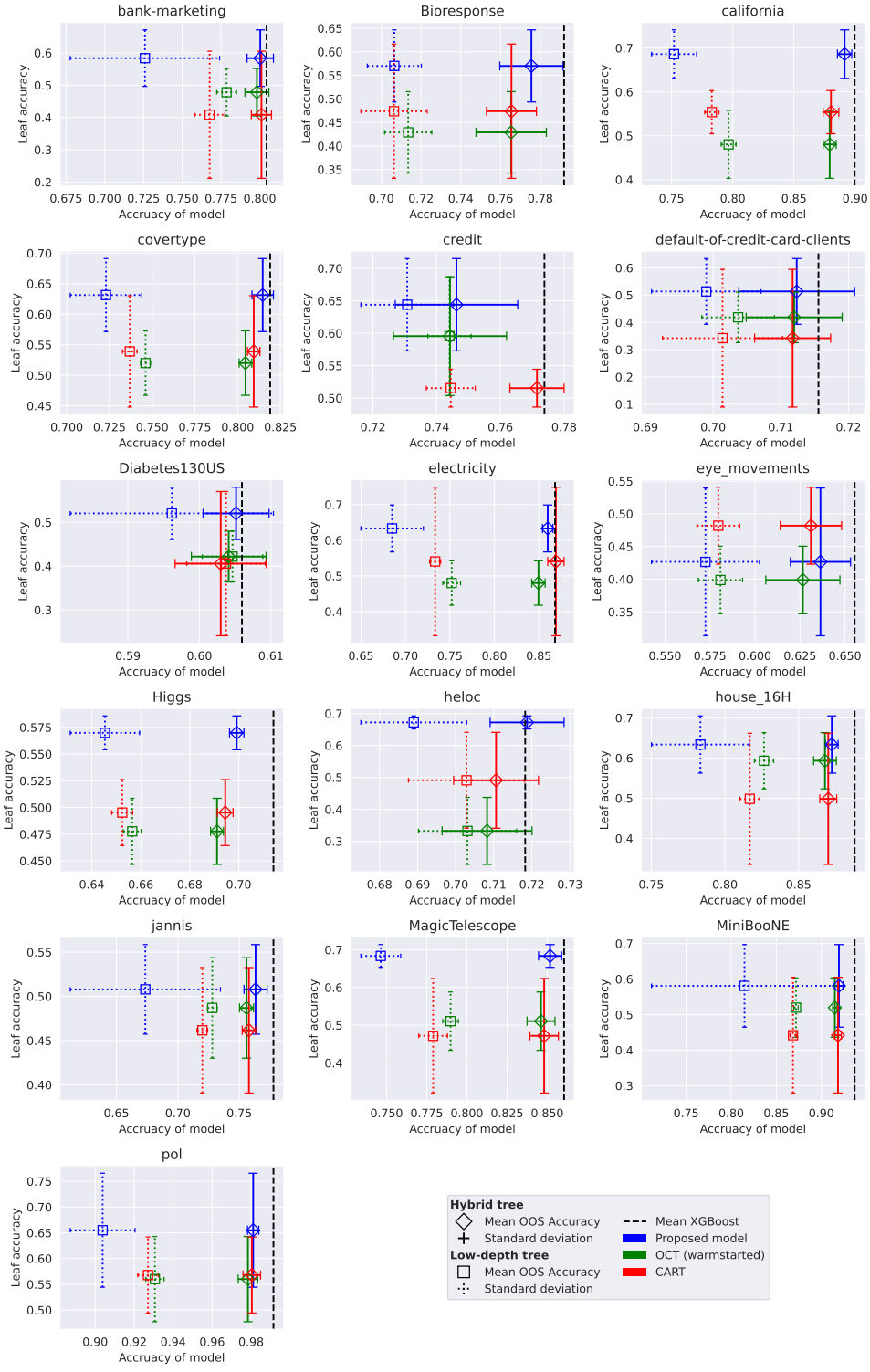


Figure C.2: Detailed performance comparison of FCT on numerical datasets

	Leaf Accuracy	
numerical datasets	CART	Proposed
bank-marketing	0.4083	**0.5837**
Bioresponse	0.4738	**0.5700**
california	0.5538	**0.6861**
covertype	0.5391	**0.6314**
credit	0.5153	**0.6439**
default-of-credit-card-clients	0.3422	**0.5136**
Diabetes130US	0.4057	**0.5204**
electricity	0.5404	**0.6331**
eye_movements	**0.4819**	0.4265
Higgs	0.4953	**0.5698**
heloc	0.4909	**0.6722**
house_16H	0.4985	**0.6336**
jannis	0.4617	**0.5079**
MagicTelescope	0.4719	**0.6835**
MiniBooNE	0.4420	**0.5809**
pol	0.5680	**0.6550**
Mean rank	1.9375	**1.0625**

Table C.6: Numerical datasets. Mean leaf accuracies of models on out-of-sample data and average ranks.

	Hybrid-tree Accuracy		
numerical datasets	CART	Proposed	XGBoost
bank-marketing	0.8011	0.8003	**0.8044**
Bioresponse	0.7655	0.7755	**0.7920**
california	0.8803	0.8914	**0.8997**
covertype	0.8094	0.8147	**0.8190**
credit	0.7715	0.7462	**0.7738**
default-of-credit-card-clients	0.7118	0.7124	**0.7156**
Diabetes130US	0.6030	0.6051	**0.6059**
electricity	**0.8692**	0.8600	0.8683
eye_movements	0.6311	0.6364	**0.6554**
Higgs	0.6945	0.6992	**0.7142**
heloc	0.7106	**0.7188**	0.7183
house_16H	0.8702	0.8726	**0.8881**
jannis	0.7578	0.7632	**0.7778**
MagicTelescope	0.8481	0.8518	**0.8605**
MiniBooNE	0.9184	0.9194	**0.9369**
pol	0.9804	0.9811	**0.9915**
Mean rank	2.7500	2.1250	**1.1250**

Table C.7: Numerical datasets. Mean hybrid tree accuracies of models on out-of-sample data and average ranks.

MASTER‘S THESIS ASSIGNMENT

I. Personal and study details

Student's name: **Němeček Jiří** Personal ID number: **475701**

Faculty / Institute: **Faculty of Electrical Engineering**

Department / Institute: **Department of Computer Science**

Study program: **Open Informatics**

Specialisation: **Artificial Intelligence**

II. Master’s thesis details

Master’s thesis title in English:

Mixed-integer Programming in Machine Learning: Decision Trees and Neural Networks

Master’s thesis title in Czech:

Smíšené celočíselné programování ve strojovém učení

Guidelines:

Mixed-integer Programming (MIP) is a framework for formulating a number of problems in machine learning (ML) and prototyping therein. With the increased interest in methods combining symbolic and statistical approaches, this seems particularly useful. Consider, for instance, training a decision tree with neural networks (NN) with ReLU activations in the leaf nodes (Zhou & Chen, 2002), whose training can clearly be cast as a MIP using the techniques of (Bertsimas & Dunn, 2017) and (Anderson et al., 2020).

Bibliography / sources:

Zhi-Hua Zhou & Zhao-Qian Chen: Hybrid decision tree. Knowledge-Based Systems, volume 15, Issue 8, pages 515-528 (2002).
Dimitris Bertsimas & Jack Dunn: Optimal classification trees. Machine Learning, volume 106, pages 1039–1082 (2017).
Ross Anderson, Joey Huchette, Will Ma, Christian Tjandraatmadja & Juan Pablo Vielma: Strong mixed-integer programming formulations for trained neural networks. Mathematical Programming, volume 183, pages 3–39 (2020).

Name and workplace of master’s thesis supervisor:

Mgr. Jakub Mareček, Ph.D. Artificial Intelligence Center FEE

Name and workplace of second master’s thesis supervisor or consultant:

Date of master’s thesis assignment: **31.01.2023** Deadline for master's thesis submission: **26.05.2023**

Assignment valid until: **22.09.2024**

Mgr. Jakub Mareček, Ph.D. Supervisor’s signature	Head of department’s signature	prof. Mgr. Petr Páta, Ph.D. Dean’s signature

III. Assignment receipt

The student acknowledges that the master’s thesis is an individual work. The student must produce his thesis without the assistance of others, with the exception of provided consultations. Within the master’s thesis, the author must state the names of consultants and include a list of references.

. Date of assignment receipt

Student’s signature