Verifying the Applicability of Optimal Decision Trees for (Imbalanced) Categorical Data

ERASMUS UNIVERSITY ROTTERDAM

Erasmus School of Economics

Bachelor Thesis Business Analytics & Quantitative Marketing

Matthijs Otten 453401

Supervisor: prof.dr. S.I. Birbil

Second assessor: dr. O. Karabag

July 5, 2020

Abstract

In this research we consider the applicability of Optimal Decision Trees generated by using a Mixed-Integer Linear Programming (MILP) formulation as described by Günlük et al. (2019) for categorizing data. We do so to verify whether this approach is a superior alternative to existing heuristics like CART for creating Decision Trees. In many fields, such as healthcare, data-driven policing and business, Artificial Intelligence assists humans in binary decision making. Therefore, it is important to improve the existing methods so that technology can help in the best possible way when categorizing data in two possible classes. In this research, we confirm previous conclusions that good Decision Trees can be generated in reasonable time through solving MILP problems and perform better in terms of accuracy than trees generated through the existing method CART. Furthermore, we investigate ways to reduce the complexity brought by continuous variables. Additionally, we show that the MILP formulation can outperform CART in generating trees to handle data for which the distribution of observations of the two classes is imbalanced, as is the case in many real world scenarios.

Contents

1 Introduction **3**

2 Related Work **3**

3 Theoretical Framework **4**

3.1 Decision Trees 5

3.2 Mixed-Integer Linear Programming 6

3.3 Formulation by Günlük et al. (2019) 7

3.4 Extension Imbalanced Data 10

4 Computational Results **11**

4.1 Replication 11

4.2 Extension Imbalanced Data 18

4.3 Extension dealing with many numerical values 20

5 Conclusion and Discussion **21**

A Topologies by Günlük et al. (2019) **25**

B Summary description of the datasets **26**

C Programming **26**

D Extension Imbalanced Data **27**

1 Introduction

Life is all about making choices. What to eat, what to wear, what to buy, whom to marry. The human species is the most intelligent form of life on earth, but its brain capacity and computing performance remain very limited. Therefore, humans started to use writing to store information and since the 1930's, programmable computing devices have been assisting humans by performing computations. Almost a century later, humans have produced forms of Artificial Intelligence (AI) that can process stored data in order to help us make wise decisions by applying statistical and mathematical theory. This can reach from deciding what assets a firm should invest in or whether a person can be regarded as a suspect in a crime. Several different "machine learning" methods and models exist nowadays to perform these data driving decision making processes. However, one of the challenges is to build models that provide interpretable results.

The focus in this research lies on the Decision Tree (DT) as particular method to produce accurate and interpretable results. Specifically, we use Integer Programming to build an Optimal Decision Tree (ODT) as done before by Günlük et al. (2019). Such an ODT is used to correctly classify observations with a binary outcome, e.g. passing or failing final exams. Our aim is to verify the conclusions of Günlük et al. (2019) on the adequacy of such an ODT and produce new insights ourselves. Additionally, we investigate for the first time if the ODT classifies well for imbalanced datasets (i.e. when one of the classes corresponds to 90% or more of the observations) and explore a suggested approach to deal with continuous variables.

First, attention is paid to previous insights from related work on the subject to describe the context of our research in Section 2. With emphasis, we look at the findings by Günlük et al. (2019). Secondly, the theoretical framework for the research is presented in Section 3. This means we look at the concepts of a DT and Mixed-Integer Linear Programming (MILP), after which we elaborate on the formulation by Günlük et al. (2019) and describe the extension for imbalanced data. Thirdly, the results are shown in Section 4 after which we finally draw our conclusions and discuss our findings in Section 5.

2 Related Work

In recent decades, researchers have found ways to generate an interpretable DT that classifies data in such a way that good decisions can be made (Breiman et al., 1984). These so called Classification and Regression Trees (CART) are applicable to uncountable scenarios, such as deciding which patients to classify as potential breast cancer patients or not and to classify a person as suspect in a crime. These trees are "grown" stepwise by means of heuristics. This means that, starting from the "root" of the DT, a split of the datapoints is chosen in such a way that best distinguishes between the different classes of observations, as if this single split was the final split of the whole procedure. This is repeated for the two resulting "nodes" (and on the next four nodes etc.) until a stopping criterion is met or the observations cannot be splitted anymore in a way that improves the distinction. The same Breiman that introduced CART in 1984 already admitted that it would be better to find a method that generates an optimal decision tree (ODT) in one go. For this end, we will focus on a method that uses Integer Programming to generate such an ODT. This idea was already present in the previous decades, but the technology did not allow for good results in reasonable time. Nowadays, with faster processors, these Integer Programming applications prove to be feasible. In several researches, Integer Programming was used and optimal DTs were found that performed better than existing alternatives like the aforementioned CART (Bertsimas & Dunn, 2017; Blanquero et al., 2019; Misic, 2019). Of these, Bertsimas & Dunn (2017) were the first to publish a research that uses MILP for generating DTs. However, as also noted by Gambella et al. (2019), their method does not exploit the combinatorial structure that accompanies categorical variables. In order for us to verify the claimed superiority of ODTs, we choose to replicate the promising research that was carried out by Günlük et al. (2019). In their MILP formulation, they do exploit the combinatorial structure brought along by categorical values, while also providing rules to cope with numerical variables. In order to keep the computation time of an ODT restricted, they predefine candidate topologies for the tree and stick with binary classification.

To improve the accuracy of DTs, aggregated predictions from an ensemble of different trees are often used. Several different methods exist to generate such an ensemble, such as boosting, bagging and generating random forests. However, the resulting models are not interpretable, as addressed by Hara & Hayashi (2016) and Palczewska et al. (2013). As interpretability is crucial in many applications, such as for creating a model that classifies people as suspects in a crime, we focus on applying Integer Programming to generate a single binary DT. In contrast, Misic (2019) applies Integer Programming on tree ensembles, so this field is also being explored. We expect that more research on the usage of MILPs will be carried out in the next years, as more research on this topic has already been done in 2020 (Aghaei et al., 2020).

3 Theoretical Framework

In this section, insights are given in the different methods used. The same notation as Günlük et al. (2019) is used for consistency, besides small improvements in terms of clarity. We will also explain what is meant in this context by *topology*, *overfitting*, *interpretability* and *performance.*

3.1 Decision Trees

Before stepping into the specific MILP formulation for creating a binary DT, we will pay attention to binary DTs in general. A DT is a machine learning tool for decision making. As an example, the setting is used in which the DT should predict whether students will pass their year or not. This prediction is based on the characteristics of the students, such as gender, family size, historical performance at school and times absent in class. By splitting the sample of students in two subsamples in each step based on a certain characteristic or combination of characteristics, the goal of the DT is to generate optimal distinctions between groups of students that tend to pass and those who tend to fail. A criterion to evaluate the *performance* of a DT upon, is the mean prediction accuracy. This is illustrated in Figure 1, where a tree of depth 2 is shown. Note that this tree has a certain symmetrical *topology* (structure) of three "decision nodes" and four "leaves". After starting with a sample of 1000 students that pass and 1000 that fail, the DT yields 4 resulting groups of students that are classified based on the pass results of the majority in the group. In our example, this yields a mean prediction accuracy of $(160 + 110 + 500 + 500)/2000 = 63.5\%$, which is higher than the 50% accuracy that a random guess would give. However, the decision rules at each decision node are to be provided by the Artificial Intelligence (AI) instead of the human. Namely, by using mathematics and statistics, the AI is much faster in finding good choices on which characteristics to use in each step and what values to split upon. These choices are made by using heuristics that result in, for instance, the CART or new methods such as optimization through MILP. Independent from the method used to generate the DT, each DT is constructed using a training dataset for which the best distinction between different classes is made. However, the aim of the DT is to correctly predict the class of observations based on their characteristics, without knowing the class beforehand. Therefore, the classification of new observations by using the generated DT should be evaluated to find the actual *performance*. Even more, a good performance on the training set does not guarantee a good performance on the testing set, even if both sets are randomly drawn from one larger dataset. This undesired phenomenon of yielding significantly higher performance on the training set than on the testing set, due to the training data-specific DT, is called *overfitting*. Unlike other machine learning tools, a DT is *interpretable* as each split is comprehensive and the resulting leaves contain clearly identifiable observations. For example, we can see from the DT in Figure 1 that a student who never failed a year and was absent at most 10 times is classified to pass this year.

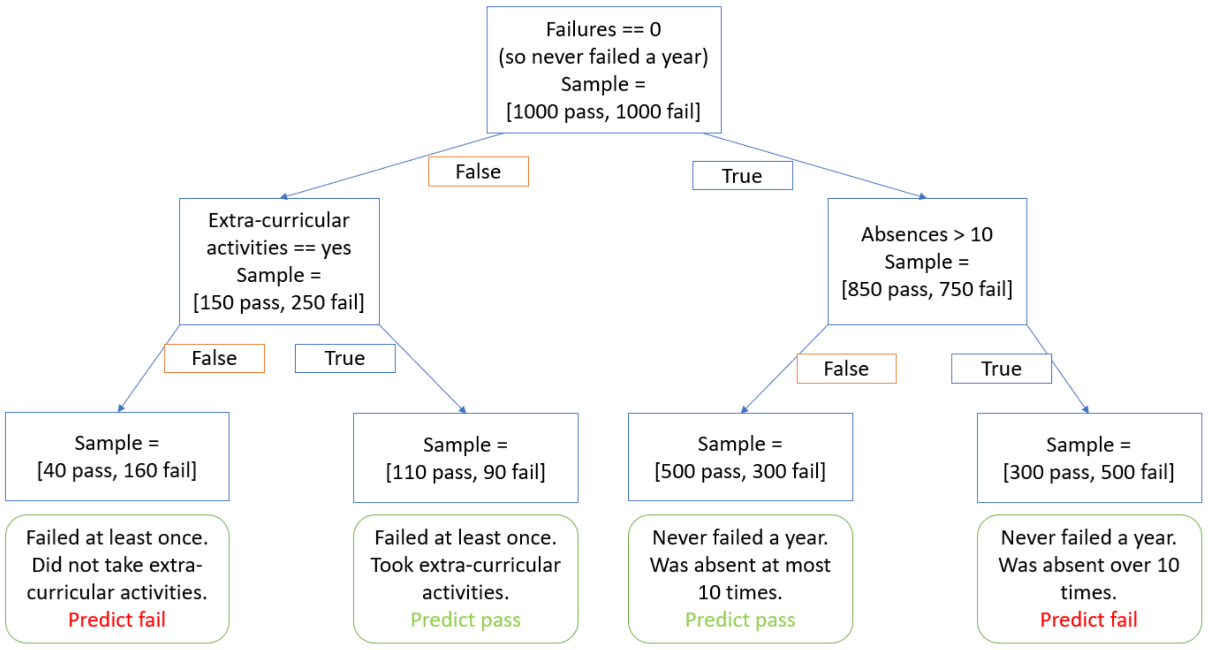


Figure 1: Decision Tree on passing of mathematics students

3.2 Mixed-Integer Linear Programming

The basic idea of solving an MILP is to optimize a given function $f(\cdot)$, subject to a certain set of constraints. For us, this means maximizing the number of correctly classified observations subject to constraints that define the structure of and splits in the DT. The programming problem gets its name "Mixed-Integer" from the fact that many, but not all, variables in the function or constraints should be integer valued. For instance, this is the case when deciding on which value of a characteristic, such as gender, to split upon in one step. As long as the problem is correctly defined, an (optimal) integer solution can be found.

Solving the MILP problem is used to find the best binary tests to use at each decision node. For example, the test whether a student has a (very) low alcohol consumption in the weekend against medium, high or very high. Note that the optimization process does not involve choosing a certain topology, but is applied on a given topology for a binary tree. The topologies considered are that of the aforementioned symmetric tree (topology *2*) of depth 2, its extension (topology *3*) of depth 3 (so 7 decision nodes and 8 leaves), its asymmetrical extension (topology *2.5*) of replacing the two left side leaves by decision nodes (resulting in 5 decision nodes and 6 leaves). The final topology *IB* (Imbalanced) is the extension of topology 2.5 by replacing the two utmost left leaves by decision nodes (resulting in 7 decision nodes and 8 leaves). The topologies are shown in Appendix A, which contains Figures 2 and 3 from respectively page 8 and 10 from the paper by Günlük et al. (2019).

3.3 Formulation by Günlük et al. (2019)

Günlük et al. (2019) present their formulation step by step, showing propositions and proofs to account for each choice made in the formulation. While using their notation, we present the final *strengthened* formulation at once, after which we provide a concise explanation.

First of all, consider a given dataset of the form $\{(g_1^i, ..., g_t^i, y^i) : i \in 1, 2, ..., N\}$. Hence, for each observation i, we have $g_j^i \in G_j$ which is the corresponding value of characteristic $j \in 1, ..., t$ and thus an element of the finite set of possible values G_j. Besides these values of the characteristics, we have $y^i \in \{0, 1\}$ which is the class label (e.g., fail or pass).

However, we replace every $g_j^i \in G_j$ by a vector of size $|G_j|$ with only zeroes, except for a single non-zero entry at the position in the vector that corresponds to the value of characteristic j corresponding to observation i. For clarity, we redefine characteristics as *groups*, indexed by $G = \{1, 2, ..., |G|\}$ that contain corresponding binary *features* $J(g)$, a subset of the set of all present features $J = \{1, 2, ..., |J|\}$. For a numerical group, the features can be combined into "bins". This restricts the number of possible solutions to consider in the optimization process, making the problem simpler. Additionally, $g(j)$ denotes the group that contains feature j.

Furthermore, we define the set with all observations as $I = \{1, 2, ..., N\}$ with $I_+ \subset I$ as the subset for all observations with a positive label (pass) and $I_- \subset I$ the subset for observations with negative labels (fail). Finally, consider the set of decision nodes $K = \{1, 2, ..., |K|\}$ and the set of leaf nodes $B = \{1, 2, ..., |B|\}$. Günlük et al. (2019) decided to force the observations to be classified as either positive or negative at each final pair of leaf nodes, where in leaf nodes of the subset $B_+ \subset B$ the observations are classified as postive and in B_- as negative. Without loss of generality, we let B_+ contain even and B_- odd indices. This does not restrict the possible solutions, as the structure of each final decision node with its pair of leaf nodes is symmetrical since the leaf nodes can be switched places by replacing the test in the final decision node with its opposite. Now, we will look at the MILP itself.

$$\max \quad \sum_{i \in I_+} \sum_{b \in B_+} c_b^i + C \sum_{i \in I_-} \sum_{b \in B_-} c_b^i \tag{1}$$

$$\text{s.t.} \quad \sum_{g \in G} v_g^k = 1 \quad \forall k \in K \ , \tag{2}$$

$$z_j^k \leq v_{g(j)}^k \quad \forall j \in J, \ \forall k \in K \ , \tag{3}$$

$$\sum_{b \in B : K^L(b) \ni k} c_b^i \leq L(i, k) \quad \forall i \in I, \ k \in K \ , \tag{4}$$

$$\sum_{b \in B : K^R(b) \ni k} c_b^i \leq R(i, k) \quad \forall i \in I, \ k \in K \ . \tag{5}$$

In (1) we have the function to maximize, which is the sum of observations with a positive label routed to positive leaf nodes plus the sum of observations with a negative label routed to negative leaf nodes. This second sum is multiplied by a weighting factor C for scenarios in which correctly classifying observations with a negative label is of a different importance than observations with a positive label. Constraint (2) makes sure that at each decision node k, exactly one group is chosen to split upon. Here, we use binary variables $v_g^k \in \{0, 1\}$ to indicate which group is indeed chosen. To ensure that only features that belong to the selected group g are candidates to be used at decision node k, we have Constraints (3). Here, binary variables $z_j^k \in \{0, 1\}$ indicate for every feature j if it is used for splitting at decision node k or not. Furthermore, Constraints (4) and (5) ensure that each observation is routed to the correct leaf node. To see this, we need to understand each part of the constraint. First, consider $L(i, k) = \sum_{j \in J} a_j^i z_j^k \quad \forall k \in K, \forall i \in I$, where a_j^i equals 1 if observation i has feature j and 0 otherwise. So, $L(i, k)$ denotes whether observation i meets the test at node k and therefore goes to the left, without loss of generality. Note that observations flow to "the left" if they meet the test, which is a structural choice since every test is symmetric in nature. Consequently, $R(i, k) = 1 - L(i, k) \quad \forall k \in K, \ \forall i \in I$. Furthermore, $K^L(b) \subset K$ is the set of decision nodes where the left branch is followed to reach leaf node b and, similarly, $K^R(b)$ for the right branch. Observe that decision variables c_b^i indicate a valid routing of observation i to leaf node b, as they take the value 1 if observation i is routed to b and 0 otherwise. This is because $L(i, k)$ and $R(i, k)$ take the values 0 or 1 and every c_b^i is maximized to this value due to the maximization of the objective function (1).

Enhancements

After having this formulation, we can speed up the optimization process by using two enhancements, as Günlük et al. (2019) show. First, we consider anchoring features. As mentioned before, the DTs are effectively symmetric. This will result in considering extra identical trees (except for switching "left" and "right") as possible solutions to the problem, whereas only considering one of them is required in the process of finding the optimal solution. Therefore, for every group g, we select the first feature in $J(g)$, j_{anchor}, as the *anchor feature*. For every decision node k that has no leaf node directly attached to it and that has symmetric subtrees on the left and the right, we add constraint:

$$z_{j_{anchor}}^k = v_g^k \ . \tag{6}$$

Second, we relax the formulation by redefining every binary variable z_j^k that is not directly attached to a leaf node and every binary variable v, such that they are continuous on the interval $[0, 1]$. This is because the formulation already ensures that, even if not declared as integers, these variables take on the value 0 or 1 in an optimal solution. This reduction of declared integer variables makes the optimization process faster (Günlük et al., 2019).

Three variations

Upon the presented formulation, Günlük et al. (2019) provides three extra variations: *controlling overfitting due to combinatorial branching*, *handling numerical features* and *maximizing sensitivity/specificity.*

First of all, Günlük et al. (2019) introduce two sets of constraints that can be used to restrict the maximum number of features to us in one split. This keeps the tests 'simple' and thereby should prevent the DT from becoming too specific for the training dataset, leading to *overfitting* and thus having a significantly lower performance on the test dataset. For every $k \in K$, decision variable m_g^k is required to decide whether the split with at most *max.card* features makes the observations flow to the left or the right. So, we have

$$\sum_{j \in J(g)} z_j^k \leq max.card + (|J(g)| - max.card)(1 - m_g^k) \ , \tag{7}$$

$$\sum_{j \in J(g)} z_j^k \geq (|J(g)| - max.card)(1 - m_g^k) \ . \tag{8}$$

Second, as mentioned before, numerical variables are just treated as categorical variables. In this case, a possible problem might be that a numerical group can take on many different values. Therefore, one should bin several numerical values together to reduce the number of features belonging to the numerical group. Also, counterintuitive tests might arise. (For example, whether a student missed 4, 6 or 13 classes or not.) These tests might help in making good distinctions for the training dataset, but are prone to making the DT too specific for this training set and thereby *overfitting*, leading to a lower performance on the test dataset. To prevent this and ensure interpretable DTs, we can enforce tests of the form "less/greater than or equal to". To this end, we introduce binary decision variables w_g^k that enforce all z_j^k in group k up to and including a certain feature to take on the value 1 or 0 and the remaining features the other value. Hence, replacing the categorical structure with a numerical structure for particular variables. This leads to the following sets of constraints:

$$z_j^k \geq z_{j+1}^k - w_g^k \quad \forall j, j+1 \in J(g) \ , \tag{9}$$

$$z_j^k \geq z_{j-1}^k - (1 - w_g^k) \quad \forall j, j-1 \in J(g) \ . \tag{10}$$

Finally, the ODT can be used to maximize the number of correctly classified observations with a certain label, while guaranteeing a minimum level of accuracy on the training dataset for observations of the opposite label. Heuristics such as CART cannot guarantee such minima. We define this minimum accuracy as s. When maximizing correct classifications of observations with a positive label, we can introduce constraint

$$\sum_{i \in I_-} \sum_{b \in B_-} c_b^i \geq \lceil s|I_-| \rceil \tag{11}$$

and alter the objective function to

$$\sum_{i \in I_+} \sum_{b \in B_+} c_b^i \, . \tag{12}$$

To do the same for observations with a negative label, we interchange the sums in (11) and (12).

3.4 Extension Imbalanced Data

Building upon the methodology and research from Günlük et al. (2019), we introduce our first extension. In many real world scenarios, the data to classify is unbalanced, meaning that one of the class-labels is overrepresented in the data. Therefore, it is of interest to investigate the performance and applicability of an ODT on such an imbalanced dataset. It is not insightful to focus on mean accuracy as performance measure, since the observations belonging to the underrepresented class are often the observations that are most important to correctly predict. Without loss of generality, we will define the underrepresented class as the positive class and the overrepresented class as the negative class.

Consider the real world scenario in which correctly classifying observations of the positive class is evaluated a certain times higher than classifying negative observations. For this purpose, we use the aforementioned hyperparemeter C, which puts a lower weight than 1 to the observations that belong to the negative class. We compare the resulting ODT with a CART that has this weight incorporated in its fitting procedure. We do so for different weights. In practise, the weight should be chosen that corresponds with the preferences of the DT user. The performance measure *Score (S)* we use, is as follows:

$$S = \frac{TP + C(TN)}{TP + FN + C(TN + FP)} \, , \tag{13}$$

where *TP* is the number of True Positives, the positive observations that are classified as such. *TN* is the number True Negatives, the negative observations that are classified as such. *FN* is the number of False Negatives, the positive observations that are classified as negative. *FP* is the number of False Positives, the negative observations that are classified as positive.

For each weight level, we compare the ODTs with different topologies and DTs from CART with different maximum depths based on their performance on a validation set. Using these results, we choose the best ODT and DT from CART to ultimately compare them based on their performance on a holdout sample.

Additionally, we verify if the third variation of the formulation by Günlük et al. (2019) can be used to guarantee a certain accuracy for the positive class (True Positive Rate, TPR) in a more effective way than using CART for different weights to find the weight for which the desired TPR is reached.

4 Computational Results

In order to get the solutions from the MILP problems, the code is written in Python 3.7 and solved with IBM ILOG CPLEX 12.10.0. For Tables 1 to 3 a dual core Intel i5-5200 2.2 GHz CPU is used. The rest of the results were found by using a 6 core Intel i5-8400 2.8 GHz CPU.

4.1 Replication

For the replication of the results by Günlük et al. (2019), we made use of the same 10 datasets. The dataset 'heloc' is from the FICO Community (2018) and the other nine from the UCI Machine Learning Repository (Dua & Graff, 2017). All of these datasets fit the purpose of the research, as they contain a binary variable to be classified. Small differences exist between number of observations and features corresponding to each dataset in our research and those belonging to the same datasets in the paper by Günlük et al. (2019). This has mainly to do with undocumented alterations of the raw datasets, so that we cannot reconstruct the exact same preprocessed datasets. Günlük et al. (2019) did not describe how they binned the numerical features together for the 'heloc' dataset. We tried to approach the same number of features (253) by clustering the numerical values together in bins of size 10 (resulting in 269 features). Therefore, we denote the dataset as 'heloc-clustered'. We prepocessed all datasets to have as final column the class labels with ones and zeroes to indicate a positive or negative class. The remainder of each dataset (representing the characteristics used for classifying) is altered to have values $j = \{1, 2, ..., |J|\}$. The summary description of the datasets can be found in Table 14 in Appendix B together with the summary description from the paper by Günlük et al. (2019).

Performance in terms of solving speed

Before we look at the performance of the ODT in terms of accuracy, we will investigate the running times for different formulation enhancements and sizes of the training sample, but only for topology *3*. Just as Günlük et al. (2019), we look at datasets 'a1a', 'bc', 'krkp', 'mush' and 'ttt'. First, we run with training samples of different sizes for each dataset such that a solution is found for at least one formulation within 1800 seconds (30 minutes), whereas Günlük et al. (2019)

had enough time available to run for 3 hours. The results can be seen in Table 1, where we put an asterisk (*) if no optimal solution can be found within 30 minutes. We use 5 variations of the formulation. The first (Nothing) is a formulation for which Constraints (4) and (5) do not have the sum in the constraints themselves, but an extra constraint, $\sum_{b \in B} c_b^i = 1 \quad \forall i \in I$, is used. The second (Relax) and third (Anchor) are the *strengthened* formulation as defined by (1) to (5), with the relaxation of some binary variables and anchoring of features respectively. The fourth (No Strength) is the same as the first (Nothing), but contains the two enhancements of anchoring and relaxing some integer variables. Finally, the fifth formulation is the strengthened formulation with both enhancements. We see that the strengthened formulation with enhancements is faster (4 to 11 times) than the No Strength formulation, except for when the 'mush' dataset is used. This result was also obtained by Günlük et al. (2019), though with different absolute solution times due to differences in hardware. Whether anchoring or relaxation of variables speeds up the process more, is different per dataset and for dataset 'krkp' not consistent with Günlük et al. (2019), as the Anchor formulation outperformed Relax and No Strength for us, but not for them. Overall, the strenghtened formulation and enhancements speed up the solving process, except for the 'mush' dataset, where the most basic formulation is the quickest.

Table 1: IP Strengthening for depth 3 with varying samples - solving time in seconds as table entries

Dataset	**Observations**	**Nothing**	**Relax**	**Anchor**	**No Strength**	**All**
a1a	**100**	*	*	*	*	769
krkp	**200**	*	1606	326	653	60
bc	**300**	*	532	53	309	42
mush	**500**	4	43	74	97	138
ttt	**300**	*	1503	394	1198	218

Now, we look at the relation between running time and the tree topology and size of the training set. We do so for datasets 'bc' and 'krkp' as can be seen in Table 2. The solution times tend to increase for larger trees and sample sizes for both datasets, but the magnitudes of these increments differ significantly. Though having the same number of nodes, the topology *IB* takes more time to solve than symmetric topology *3*.

Table 2: Solution times (in seconds) for **krkp and bc** without feature selection

Topology	Data set	100	200	300	400	500	600
depth2	krkp	1	3	5	8	11	19
depth2.5	krkp	7	16	30	54	67	105
depth3	krkp	58	60	100	301	371	642
depthIB	krkp	75	228	381	876	683	1545
depth2	bc	0	1	2	3	6	6
depth2.5	bc	1	40	94	226	479	707
depth3	bc	0	9	42	*	*	*
depthIB	bc	0	10	52	*	*	*

Next, we investigate the effect of deleting candidate groups that do not seem to help in correctly classifying the observations (and refer to this as *feature selection*). To do so, we first apply the CART algorithm to each dataset with a random 90% of the observations for 5 runs. Then, we delete all groups that were not used in at least 4 runs. Note that it was unclear how exactly Günlük et al. (2019) did their group selection, but the similarity is that we all used CART to identify the groups that did not seem important. We replicate Table 2, but now without these deleted groups and show the results in Table 3. In most scenarios, we see a decrease in solving time of about 50%.

Table 3: Solution times (in seconds) for **krkp and bc** using feature selection

Topology	Data set	100	200	300	400	500	600
depth2	krkp	1	2	3	3	6	8
depth2.5	krkp	7	16	35	39	57	97
depth3	krkp	36	51	77	152	239	381
depthIB	krkp	56	128	166	306	353	610
depth2	bc	0	1	1	2	3	4
depth2.5	bc	0	19	65	138	214	323
depth3	bc	0	3	67	911	*	*
depthIB	bc	0	5	26	1381	*	*

Performance in terms of accuracy

In the rest of the research, we will focus more on classification (training) and prediction (testing) accuracy as measure of performance. Note that all following results are averages from 5 runs with a solution time limit of 5 minutes and that, if not stated otherwise, training sizes of 600 observations are used. For most of the runs, except for results for topology *2*, the training phase was cut off after this 5 minute limit due to time constraints, indicating that the trees are often not optimal for the training sample. Note that even a DT that is optimal for the training sample does not have to be optimal for the testing sample.

We present the results of ODT compared to that of CART (with maximum depth of 3) in Tables 4 (without feature selection) and 5 (with feature selection, as described previously). If the highest testing accuracy of an ODT topology is at least 1% larger than that of CART, we highlight it in bold, and visa versa for the testing accuracy of CART. We define such a difference as *significant*. When not using feature selection, the ODT performed significantly better than the CART for 7 scenarios and visa versa for only 1. This seems to show that the ODT performs better than CART for similar depths. Feature selection only improved the ODT performance for three datasets (heloc_clustered, krkp, mush), but the testing accuracies only decreased for the 'ttt' dataset. In the paper by Günlük et al. (2019), feature selection improved performance for five datasets and deteriorated for three datsets. Therefore, we do share the conclusion from Günlük et al. (2019) that feature selection could indeed improve performance, but emphasize that it could also have a negative effect. Considering the presence of *overfitting*, only for 5 (a1a, heloc_clusterd, krkp, ttt, student) out of 10 datasets the gap between training and testing accuracy for the best performing ODT was larger than for CART-D3. Thus, we have no indication that overfitting is more of a problem for ODT in comparison to a heuristic based method like CART.

Table 4: The average training (testing) accuracy with 5 mins limit without feature selection.

Dataset	**Depth 2**	**Depth 2.5**	**Depth 3**	**Depth IB**	**CART-D3**
a1a	81.7 (79.3)	83.6 (80.1)	83.1 (77.9)	83.7 (78.1)	80.8 (79.6)
bc	96.8 (**95.7**)	97.8 (93.7)	98.2 (93.7)	98.3 (92.8)	96.1 (94.5)
heloc_clustered	73.8 (65.6)	74.9 (64.7)	74.5 (63.2)	74.4 (64.7)	68.1 (**68.9**)
krkp	87.4 (86.8)	93.8 (**93.8**)	93.8 (93.7)	93.2 (92.8)	90.4 (90.8)
mush	99.5 (99.3)	100.0 (**99.5**)	100.0 (99.5)	100.0 (99.5)	98.6 (98.1)
ttt	72.7 (66.3)	77.7 (72.3)	79.9 (72.4)	80.8 (**76.5**)	74.9 (74.2)
monks-1	78.2 (73.6)	84.2 (75.9)	89.4 (84.1)	100 (**100**)	76.5 (77.7)
votes	96.2 (94.5)	96.8 (94.1)	97.3 (94.1)	98.0 (**97.3**)	96.8 (94.5)
heart	79.4 (**76.3**)	82.8 (70.4)	82.8 (70.4)	85.3 (73.3)	81.4 (71.1)
student	77.2 (73.5)	79.4 (73.5)	80.3 (65.0)	79.0 (66.0)	76.3 (73.5)

Table 5: The average training (testing) accuracy with 5 mins limit with feature selection.

Dataset	**Depth 2**	**Depth 2.5**	**Depth 3**	**Depth IB**	**CART-D3**
a1a	81.7 (79.3)	83.6 (80.1)	83.2 (78.2)	83.8 (78.1)	80.8 (79.6)
bc	96.8 (**95.7**)	97.8 (94.9)	98.2 (93.3)	98.4 (93.3)	96.1 (94.5)
heloc_clustered	73.8 (66.1)	75.5 (64.0)	75.7 (61.8)	75.3 (62.9)	68.1 (**68.9**)
krkp	87.4 (86.8)	93.8 (93.8)	93.8 (93.7)	94.3 (**94.1**)	90.4 (90.8)
mush	99.5 (99.4)	100.0 (**99.8**)	100.0 (99.8)	100.0 (99.8)	98.6 (98.1)
ttt	72.7 (66.3)	77.7 (72.3)	80.0 (72.3)	80.3 (**75.7**)	74.9 (74.2)
monks-1	78.2 (73.6)	84.2 (75.9)	89.4 (84.1)	100 (**100**)	76.5 (77.7)
votes	96.2 (94.5)	96.8 (95.5)	97.3 (93.6)	98.0 (**97.3**)	96.8 (94.5)
heart	79.4 (**76.3**)	82.8 (70.4)	82.8 (70.4)	85.3 (73.3)	81.4 (71.1)
student	77.2 (73.5)	79.4 (73.5)	80.3 (65.0)	79.0 (66.0)	76.3 (73.5)

Additionally, we look at the effect of the size of the training sample on the testing accuracy. The results are shown in Table 6. For dataset 'a1a' no conclusion can be drawn on the effect of increasing sample size, as this effect differs per topology. From the results for datasets 'krkp' and 'heloc_clusterd' we learn that the effect of training size on testing accuracy can change. E.g. for 'heloc_clusterd', we see that testing accuracy drops when going from 600 observations to 1200, but increases again when going from 1800 to 2400. Testing accuracies stay the same among all trainings sizes for 'mush'. More importantly, these results show that large training samples are often not required to achieve high testing accuracy and, for a given topology, testing accuracy can substantially differ per training size.

Table 6: Comparison of training (testing) accuracy across training data sizes with 5 minutes limit and feature selection

Dataset	**Topology**	**600**	**1200**	**1800**	**2400**
a1a	2	82.0 (78.0)	81.9 (79.9)	-	-
krkp	2	86.8 (87.6)	86.8 (87.0)	86.6 (87.4)	86.6 (87.8)
mush	2	99.4 (99.3)	99.5 (99.3)	99.4 (99.3)	99.4 (99.4)
heloc_clustered	2	68.7 (69.3)	72.1 (67.8)	70.7 (68.2)	70.2 (68.9)
a1a	2.5	82.4 (79.9)	82.5 (79.5)	-	-
krkp	2.5	82.5 (82.1)	93.7 (93.9)	93.6 (94.1)	93.7 (94.2)
mush	2.5	99.9 (99.9)	100.0 (99.9)	99.9 (99.9)	99.9 (99.9)
heloc_clustered	2.5	68.6 (69.5)	71.9 (68.0)	69.6 (67.9)	68.6 (68.4)
a1a	3	81.2 (79.9)	82.0 (79.6)	-	-
krkp	3	85.5 (85.2)	93.7 (93.9)	92.2 (92.4)	88.9 (89.2)
mush	3	100 (100)	100.0 (99.9)	100.0 (100.0)	100.0 (100.0)
heloc_clustered	3	69.0 (69.2)	70.2 (67.3)	68.9 (67.0)	68.7 (68.3)
a1a	IB	81.2 (75.5)	81.5 (79.3)	-	-
krkp	IB	75.3 (74.8)	91.9 (92.0)	87.9 (88.4)	73.3 (73.3)
mush	IB	100 (100)	100.0 (99.9)	100.0 (99.9)	100.0 (100.0)
heloc_clustered	IB	68.8 (69.5)	70.0 (67.4)	68.7 (68.0)	68.4 (68.4)

Throughout the research, we used four different topologies. Now, we will choose the best topology for ODT via cross-validation and then compare testing accuracy and tree size with CART. In Table 7 we present our results. Note that we first do feature selection as described before. For every dataset, we reserve 10% of the observations as holdout sample for final testing. From the remaining 90%, we run 5 validation rounds for each topology by training with 600 observations (or 90% of this 90% when the dataset is too small) and report (validation) accuracy on the remaining observations of this 90%. Then, we take the number of leaves of the best topology in each run and report the average over the 5 runs. Additionally, we find the topology that has the highest average validation accuracy over the 5 runs. Then, we use the 5 resulting trees of this topology to classify the observations in the holdout sample and report the average (testing) accuracy. For CART, Günlük et al. (2019) say they do not impose a maximum depth, train with 90% of the observations as training sample and report accuracy on the holdout sample. However, this cannot yield an average number of leaves as CART seems to be carried out only once and this number of leaves is much (around 10 times) smaller than was the case in our initial replication. Therefore, we choose to run 5 validation runs for several maximum depths with 81% as training sample (so 90% of the 90%) and 9% as validation sample, leaving the holdout sample. Then, we take the tree that showed best on average for validation and take the average number of leaves from these 5 trees (of same maximum depth). Finally, we train on the 90% with the optimal maximum depth from validation and report the accuracy of the tree applied on the holdout sample. In many validation runs for ODT, there is not one single dominant topology. Therefore, we report average number of leaves in two ways: in the fourth column for when the smaller or simpler topology is chosen in case of a tie and in the fifth column for when the larger or more sophisticated topology is chosen. From the results in Table 7 we can conclude that CART outperforms the ODTs if it is allowed to grow large and for ODT no optimal solution is found on the training sample within 5 minutes (datasets krkp, ttt, student). If such an optimal solution is found, ODT is better for some datasets (bc, monks-1, votes), but CART for others (a1a, heloc_clustered, mush, heart). Additionally, the larger ODTs perform better or equally well as the small ODTs for 7 datasets.

Table 7: Comparison of testing accuracy and size of cross validated trees vs. CART

Dataset	**ODT**	**Optimal**	**leaves (small)**	**leaves (large)**	**CART**	**ave. leaves**
a1a	77.7	Yes	5.2	6	81.2	8
bc	98.8	Yes	4.8	6.4	93.8	16
heloc_clustered	66.7	Yes	5.2	5.2	68.6	16
krkp	91.2	No	6.8	8	99.4	50.2
mush	99.5	Yes	6.4	8	100	9
ttt	76	No	8	8	94.9	90
monks-1	100	Yes	8	8	91.3	47
votes	96.8	Yes	8	8	94.9	13.4
heart	73.3	Yes	6.8	7.6	78.3	18.6
student	61	No	7.6	8	65.1	68.2

Three variations

As mentioned, Günlük et al. (2019) propose three variations of the formulations. The first is to prevent overfitting by restricting the number of features that can be tested upon at one decision node. For the comparison, we take topologies 2 and 3, datasets 'a1a' and 'mush' (as used by Günlük et al. (2019)) and three scenarios. The first (Simple) is where only one feature can be used in the test, the second (Comb-con) has *max.card* of two and the third (Comb-unc) is unconstrained, so the base case scenario. It was not mentioned by Günlük et al. (2019) that relaxing binary variables cannot be done when restricting the maximum number of features to use at a decision node, as this will sometimes result in variables z_j^k taking on values between 0 and 1. We show the results in Table 8. We question whether Günlük et al. (2019) indeed only used binary variables, as their results for Comb-unc are inconsistent with results that should be equal and are reported in other tables. For 'a1a' we see that the training accuracy for scenario Simple is the smallest, but not the testing accuracy. In combination with that the testing accuracy for Comb-con is larger than for Comb-unc, this result is in line with the expectation that unconstrained combinatorial branching leads to overfitting. For the 'mush' dataset, the results are not straightforward on which scenario is best. This is in contrast with the results of Günlük et al. (2019), mainly because of the fact that for the Simple scenario, we achieve similar testing accuracy results for topology *3*, whereas their testing accuracy for Simple is inferior to that of the other scenarios. As the results on only these two datasets are insufficient to draw conclusions, we also use datasets 'bc' and 'heloc_clustered'. For both datasets, we find that Simple yields the smallest gaps between training and testing accuracy, whereas for Comb-con the highest testing accuracies are obtained. Therefore, restricting the number of features to use in a split can indeed prevent overfitting and improve performance. For best application, it is to be investigated what is the optimal maximum number of features for each dataset and topology.

The second variation is to compare the performance of the situation in which the categorical structure of the formulation is replaced with a numerical structure for numerical variables. As can be seen from the results in Table 9, gaps between training and testing accuracy close in many cases when the numerical variation (rows with 'n') is applied. Also, this numerical structure yields higher testing accuracy than the base categorical (rows with 'c') structure. This suggests that, due to restricting the number of possible tests, the second variation can help to prevent overfitting and achieve better out of sample performance.

The third variation is about maximizing the classification of observations from a specific class, while guaranteeing a minimum accuracy on the training sample for observations of the other class. We consider the scenario in which an accuracy for observations of the positive class (so the True Positive Rate, TPR) should be guaranteed (while maximizing the True Negative Rate, TNR). From the results in Table 10, for dataset 'bc' and topologies 2 and 3, we see that gaps exist between guaranteed training TPR and testing TPR of around 4% for topology *2* and 9% for topology. This seems to imply that a 'buffer' on the guaranteed TPR should be taken into account when setting the constraint on the training sample, in order to achieve the desired TPR on the testing sample.

Table 8: The average training (testing) accuracy for combinatorial vs. simple branching using depth 2 and depth 3 trees

	Depth 2			**Depth 3**		
Dataset	**Simple**	**Comb-con**	**Comb-unc**	**Simple**	**Comb-con**	**Comb-unc**
a1a	80.6 (80.8)	81.4 (80.5)	81.7 (79.2)	81.6 (79.4)	82.2 (79.5)	82.8 (79.0)
bc	94.7 (95.2)	96.4 (97.1)	96.8 (95.7)	96.1 (95.4)	97.5 (96.1)	98.0 (93.5)
heloc_clustered	68.6 (67.3)	70.9 (68.5)	73.8 (65.5)	68.8 (68.0)	70.6 (68.6)	73.9 (65.0)
mush	97.3 (96.9)	99.5 (99.3)	99.5 (99.2)	99.7 (99.5)	100.0 (99.4)	100.0 (99.7)

Table 9: The average training (testing) accuracy/solution time with or without constraints for numerical features

Dataset	**n/c**	**Depth 2**	**Depth 2.5**	**Depth 3**	**Depth IB**
a1a	n	81.7 (79.2)	83.7 (79.6)	83.6 (79.6)	84.0 (79.1)
	c	82.0 (78.0)	82.4 (79.9)	81.2 (79.9)	81.2 (75.5)
bc	n	96.8 (96.4)	97.6 (95.7)	97.9 (94.7)	97.8 (96.1)
	c	96.8 (95.9)	97.8 (94.2)	98.1 (95.4)	98.2 (93.3)
heloc_clustered	n	70.7 (69.3)	71.4 (67.6)	70.6 (67.8)	70.3 (69.2)
	c	68.7 (69.3)	68.6 (69.5)	69.0 (69.2)	68.8 (69.5)
student	n	74.8 (72.0)	76.1 (74.0)	77.4 (71.0)	76.6 (71.5)
	c	77.2 (73.5)	79.4 (73.5)	80.3 (65.0)	79.0 (66.0)

Table 10: TPR vs. TNR, breast cancer data, depth 2 and depth 3 trees

Depth 2				**Depth 3**			
Training		**Testing**		**Training**		**Testing**	
TPR	**TNR**	**TPR**	**TNR**	**TPR**	**TNR**	**TPR**	**TNR**
100	87.6	97.1	88.9	100	96.5	91.9	96.0
99.5	91.4	95.2	94.4	99.5	97.2	90.3	97.2
99	93.6	95.1	94.4	99	97.3	93.2	96.7
98.5	94.8	96.2	96.0	98.5	98.0	90.3	97.2
98	95.7	96.5	97.1	98	98.1	88.9	97.2
97	96.5	93.2	97.3	97	98.8	86.1	97.6
96	97.0	92.5	97.4	96	99.3	85.1	97.6
95	97.4	91.8	97.6	95	99.5	85.4	97.6

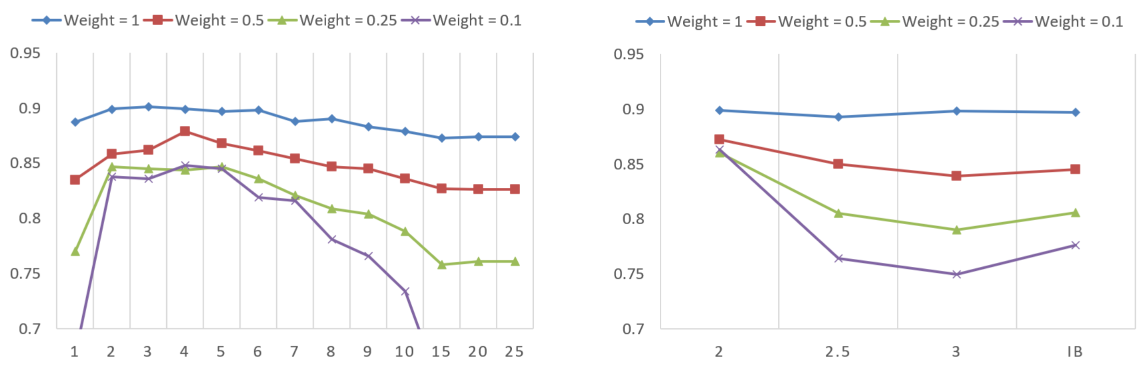
4.2 Extension Imbalanced Data

For the purpose of investigating how well the ODT can handle imbalanced data, the 'Bank Marketing Data Set' is used from the UCI Learning Repository (Dua & Graff, 2017). This 'bank' dataset has 4120 observations of which only 10% belongs to the positive class, 20 groups and a total of 209 features. We cluster features into 10 bins of the same size for the groups 'age', 'duration' and 'euribor3m', as these take on many different numerical values. Furthermore, we do feature selection using CART as described before. We take 25% as holdout sample, 50% for training and 25% for validation. In order to find the best topology for ODT and best maximum

Table 11: Testing scores for best ODT versus best CART

Weight	**ODT**	**CART**
1	0.906	0.913
0.5	0.881	0.874
0.25	0.863	0.845
0.1	0.842	0.828

depth for CART, 5 validation runs are carried out with a random 90% of the training sample for different topologies and maximum depths. In Figure 2, the resulting average validation 'Score' measure for different trees can be seen. For CART, we find that the best maximum depths obtained are 3, 4, 2 and 4 for weights 1, 0.5, 0.25 and 0.1 respectively. For ODT, we observe that topology *2* is best for all weights. Using these topologies and maximum depths for the corresponding weights, we find Scores by applying the trees on the holdout sample. Again, we do 5 runs with a random 90% of the training sample and report the average Scores in Table 11. We notice that ODT scores higher for all 3 scenarios with weights deviating from 1, suggesting that using weights C in the optimization process yields better performance than using the same weights in CART.



(a) CART for different maximum depths (b) ODT for different topologies

Figure 2: Validation scores for different trees

Besides looking at the differences in Score between ODT and CART, we also consider the possibility of ODT to guarantee a minimum TPR. In contrast with solving for only one ODT, guaranteeing this minimum TPR for CART implies finding the correct tree by running validation runs for several candidate trees. We consider minimum TPRs of 80, 85, 90 and 95. Again, we do 5 runs using the aforementioned training and validation sample, for which we report average results. In Appendix D, we show the tables used for finding the weights and maximum depths for which CART achieves the minimum validation TPR. In Table 12, we present the validation TPR and TNR (in brackets) for different topologies and guaranteed TPRs on the

Table 12: TPR (TNR) for different trees and guaranteed training TPRs on the bank dataset

	ODT				**CART**
Training TPR	**topology *2***	**topology *2.5***	**topology *3***	**topology *IB***	**[depth, 1/weight]**
95	91.8 (79.7)	92.2 (51.0)	91.2 (51.4)	94.4 (52.7)	96.8 (67.5) [3,15]
90	85.3 (84.9)	87.1 (59.5)	86.9 (63.9)	90.4 (63.0)	91.5 (74.3) [4,17]
85	80.9 (88.4)	77.6 (87.2)	81.9 (69.5)	84.4 (72.3)	85.8 (83.7) [3,6]
80	75.5 (90.1)	76.3 (87.4)	77.2 (76.6)	79.2 (78.4)	81.9 (86.8) [2,2]

training sample, as well as for the best resulting validation CART with its maximum depth and weight in square brackets. For ODT, we see very low testing TNR or large gaps between the guaranteed training TPR and achieved validation TPR. Consequently, one should also search for the optimal guaranteed TPR for the training sample, just as such a search should be carried out for the weights for CART. However, for validation TPRs of at least 90, 85 and 80, the ODT with topology *2* scores higher TNRs (79.7, 84.9, 88.4) than CART (74.3, 83.7, 86.8). Therefore, we conclude for this particular imbalanced dataset that ODT achieves higher performance for a given TPR than CART, though a search for the correct guaranteed training TPR is also required in order to achieve the desired TPR on a holdout sample.

4.3 Extension dealing with many numerical values

As mentioned by Günlük et al. (2019), the computation time of the ODT depends on the size of the dataset if it contains continuous variables. Namely, if the size of the dataset grows, the number of unique numerical values for the continuous variables also tends to grow and thereby the number of features to be considered in the solving process. Günlük et al. (2019) already mention binning these continuous values into 10 features of equal size, whereby the number of features remains constant if the dataset grows. We will test this method of creating a constant number of bins per numerical group on the 'heloc' and 'bank' dataset by comparing the Score (see 13) for 5, 10, 15 and 20 bins and our previous approach to just generate bins of size 10. We use weight $C = 1$ for the 'heloc' dataset and $C = 0.5$ for 'bank'. In Table 13 we see the results for topologies 2 and 3 and CART (with a maximum depth chosen after validation). When looking at the four ways of creating the same amount of bins per variable, we see that 15 bins performs best for 'heloc' and 10 bins for 'bank'. It seems that too few bins does not allow for sufficient *flexibility* for the DT to split and too many bins leads to *overfitting*. Interestingly, our unsupported approach of just creating bins of size 10 performs best for 'heloc', but not for 'bank'.

Table 13: Training (testing) score for different feature binning approaches

Dataset	**feature binning**	**Depth 2**	**Depth 3**	**CART**
heloc	5 bins	0.614	0.592	0.661
	10 bins	0.619	0.603	0.671
	15 bins	0.635	0.608	0.676
	20 bins	0.602	0.595	0.663
	bins of size 10	0.650	0.642	0.698
bank	5 bins	0.831	0.831	0.842
	10 bins	0.830	0.841	0.844
	15 bins	0.829	0.829	0.836
	20 bins	0.829	0.831	0.838
	bins of size 10	0.811	0.799	0.820

5 Conclusion and Discussion

Throughout the research, we have been verifying the results from Günlük et al. (2019) on the use of an Optimal Decision Tree (ODT). Our conclusions matched on the relative differences in terms of running times between the different topologies, training sample sizes and formulation variants. We see that, with 2020 hardware standards of consumer PCs, ODTs can be found within 5 minutes for a small (4 leaf nodes) topology and well performing Decision Trees (DTs) for larger topologies. This shows that the usage of MILP formulation in generating DTs is useful in practise, as it yields good DTs in reasonable time. In terms of accuracy, MILP generated DTs with a maximum of 8 leaves perform better than or equally well as equally sized trees generated by CART, for 9 out of the 10 datasets used. However, if the CART is allowed to grow larger, the MILP DT can often not match its testing accuracy, in particular for datasets on which an optimal tree on the training sample can not be found within the 5 minutes time limit. The difference in testing accuracy for different training sizes did not seem substantial, which gives the important insight that the MILP formulation can generate a good DT without needing a large training size. However, the optimal training size is dataset-specific and large differences ($> 1\%$) between testing accuracies of different training sizes might exist for a given topology. Therefore, it could be worthwhile to generate DTs with different training sizes and use the training size that yields best accuracy on a validation set. Analogously to Günlük et al. (2019), we saw that restricting the number of features to use in a single test can prevent overfitting and improve performance. In further research, recommendations could be developed on the maximum number of features to use in different scenarios. Also for the variation of imposing a 'less/greater than or equal to' structure for numerical variables, we saw that the variation can help to prevent overfitting and achieve higher out of sample performance. However, this is dataset and topology specific. Therefore, in practice the numerical structure should first be validated on a validation set to check whether it indeed is an improvement compared to the basic categorical structure of the formulation. The possibility for ODT to impose a minimum accuracy for observations of one particular class unfortunately has the restriction that this minimum accuracy can only be guaranteed for the training sample, so that a gap between training and testing accuracy should be accounted for.

We saw that using weight parameter C can be used to achieve high accuracy for the underrepresented observations in an imbalanced dataset. Additionally, for our 'bank' imbalanced dataset, ODT performed better than the best possible tree generated by CART in terms of weighted accuracy. By using the guaranteed minimum accuracy (True Positive Rate, TPR) on the training sample, a slightly lower ($\approx 4\%$) TPR for the out of sample observations can be achieved. Therefore, such a gap between training TPR and testing TPR should be accounted for. Additionally, this approach of guaranteeing the TPR (or TNR) and maximizing the TNR (or TPR) yields a higher overall accuracy for ODT than when using the tree generated by CART that has the highest TNR while yielding the required minimum TPR. Hence, the ODT can be used in practice to replace CART for imbalanced datasets, though its superiority should be verified on more datasets. As mentioned by Günlük et al. (2019), a larger dataset with continuous variables will contain more numerical features and so the complexity of the MILP problem to solve is not independent of the sample size. Therefore, we compared our unsupported approach to deal with many numerical features with the approach to create a fixed number (5, 10, 15 or 20) of equally sized bins with features from each numerical group, after which these bins are used as the features in the formulation. This approach yields a number of features that is independent of the sample size. No common conclusion could be drawn from the results for the 'heloc' and 'bank' dataset. For now, the optimal binning strategy seems data specific, but further research could possibly yield rules of thumb on the ratio between the number bins to use and numerical features present.

We share the conclusion from Günlük et al. (2019) that Decision Trees constructed by using MILP formulation outperform trees of the same size produced by CART, but not larger trees. Due to limited time available, we had to impose time limits on solving the MILP formulations. It would be of interest to see how well the ODT performs if enough time is available to indeed solve to optimality. Additionally, we showed that the MILP approach can yield superior results for imbalanced datasets in comparison with CART, although this should be verified for more imbalanced datasets.

References

Aghaei, S., Gómez, A., & Vayanos, P. (2020). Learning Optimal Classification Trees: Strong Max-Flow Formulations. *ArXiv*, *abs/2002.09142*. Retrieved 01/07/2020, from https://arxiv.org/abs/2002.09142

Bertsimas, D., & Dunn, J. (2017). Optimal Classification Trees. *Machine Learning*, *106*(7), 1039-1082.

Blanquero, R., Carrizosa, E., Molero-Río, C., & Romero Morales, D. (2019). Optimal Randomized Classification Trees. In *Conference proceedings : 2nd Spanish Young Statisticians and Operational Researchers Meeting* (p. 53). Madrid: Universidad Complutense de Madrid * Servicio de Publicaciones.

Breiman, L., Friedman, J. H., Olshen, R. A., & Stone, C. J. (1984). *Classification and Regression Trees.* Taylor and Francis.

Dua, D., & Graff, C. (2017). *UCI Machine Learning Repository.* Retrieved 30/06/2020, from http://archive.ics.uci.edu/ml

FICO Community. (2018). *Explainable Machine Learning Challenge.* Retrieved 18/06/2020, from https://community.fico.com/s/explainable-machine-learning-challenge

Gambella, C., Ghaddar, B., & Naoum-Sawaya, J. (2019). Optimization Models for Machine Learning: A Survey. *ArXiv*, *abs/1901.05331*. Retrieved 01/07/2020, from https://arxiv.org/abs/1901.05331

Günlük, O., Kalagnanam, J., Menickelly, M., & Scheinberg, K. (2019). Optimal Decision Trees for Categorical Data via Integer Programming. Retrieved 21/06/2020, from http://www.optimization-online.org/DB_HTML/2018/01/6404.html

Hara, S., & Hayashi, K. (2016). Making Tree Ensembles Interpretable. *ArXiv*, *abs/1606.05390*. Retrieved 01/07/2020, from `https://arxiv.org/abs/1606.05390`

Misic, V. V. (2019). Optimization of Tree Ensembles. *ArXiv*, *abs/1705.10883*. Retrieved 01/07/2020, from `https://arxiv.org/abs/1705.10883`

Palczewska, A., Palczewski, J., Robinson, R. M., & Neagu, D. (2013). Interpreting Random Forest Models Using a Feature Contribution Method. In *2013 IEEE 14th International Conference on Information Reuse Integration (IRI)* (p. 112-119).

A Topologies by Günlük et al. (2019)

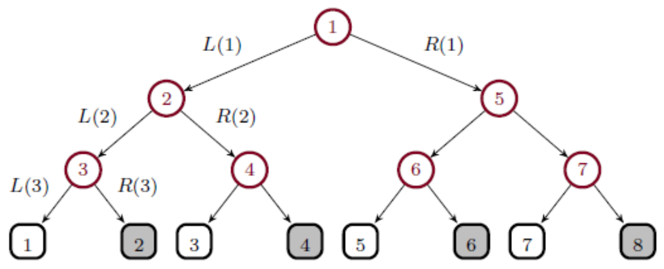


Figure 3: Topology 3 (Figure 2 on page 8)

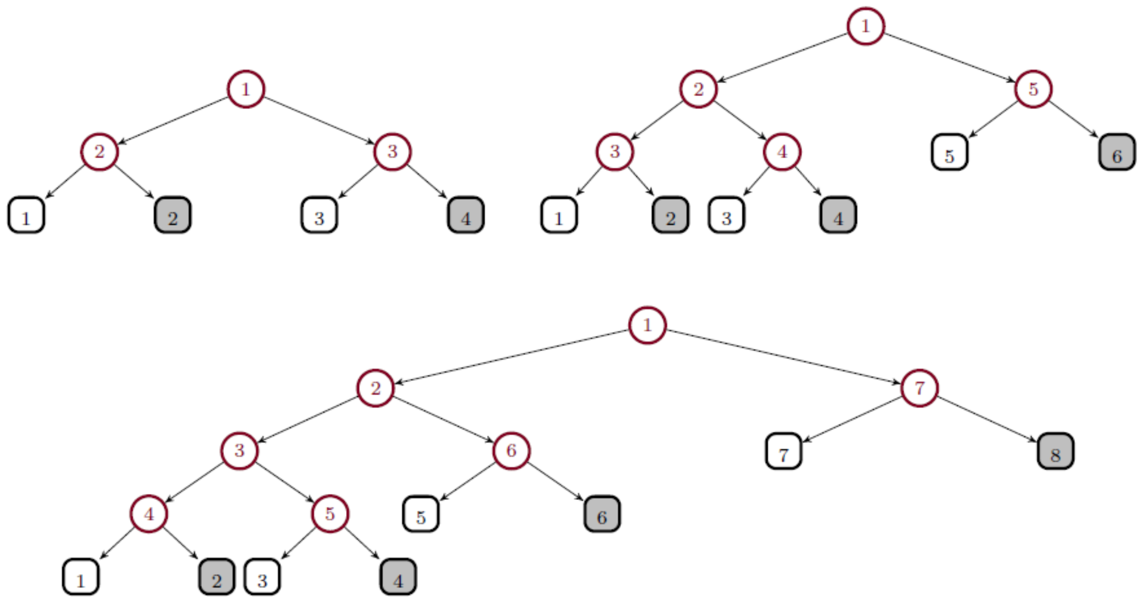
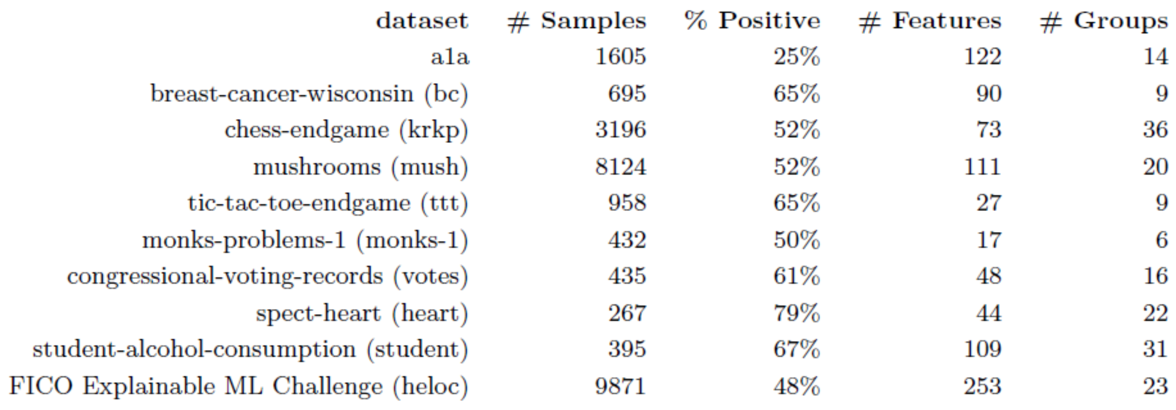


Figure 4: Topologies 2, 2.5, IB (Figure 3 on page 10)

B Summary description of the datasets

Table 14: Summary description of the datasets

Dataset	# Observations	% Positive	# Features	# Groups
a1a	1478	26%	112	14
breast-cancer-wisconsin (bc)	683	65%	89	9
chess-endgame (krkp)	3196	52%	73	36
mushrooms (mush)	8124	52%	117	22
tic-tac-toe-endgame (ttt)	958	65%	27	9
monks-problems-1 (monks-1)	432	50%	17	6
congressional-voting-records (votes)	435	61%	48	16
spect-heart (heart)	267	79%	44	22
student-alcohol-consumption (student)	395	67%	137	30
FICO Explainable ML Challenge (heloc)	10459	48%	269	23



dataset	# Samples	% Positive	# Features	# Groups
a1a	1605	25%	122	14
breast-cancer-wisconsin (bc)	695	65%	90	9
chess-endgame (krkp)	3196	52%	73	36
mushrooms (mush)	8124	52%	111	20
tic-tac-toe-endgame (ttt)	958	65%	27	9
monks-problems-1 (monks-1)	432	50%	17	6
congressional-voting-records (votes)	435	61%	48	16
spect-heart (heart)	267	79%	44	22
student-alcohol-consumption (student)	395	67%	109	31
FICO Explainable ML Challenge (heloc)	9871	48%	253	23

Figure 5: Table 1 from the paper by Günlük et al. (2019) on page 17: Summary description of the datasets

C Programming

In order to generate the Decision Trees using Mixid-integer Programming Problem (MILP) or CART, we have made use of Python 3.7. We used the DOcplex package (IBM Decision Optimization CPLEX Modeling for Python) to let CPLEX solve the MILP and the scikit-learn package to perform CART. Our code is too long to add in this appendix and is therefore available in a compressed ZIP-file.

D Extension Imbalanced Data

Table 15: TNR for CART with guaranteed training TPR > 80

1/Weight	**Maximum depths**									
	1	**2**	**3**	**4**	**5**	**6**	**7**	**8**	**9**	**10**
1										
2		**86.8**								
3		86								
4		86								
5		86	82.8							
6		85.1	83.7	84.7	85					
7		85.1	82.1	84.5	85.3					
8		84.4	81.7	84.4	84.4	84.5				
9		83.2	82.1	82.8	80.6	82.3				
10		83.2	78.3	81.1	80.4	82.2				
11		83.2	78.4	79.4	80.4	80.6				
12		83.2	76.9	80.9	79.5	80.7	82.5			
13		83.2	73.9	80.7	79.3	80	80.9			
14		83.2	67.3	77.5	79.3	78.5	80.8			
15		83.3	67.5	77.3	79.3	78.4	80.6			
16	35.5	83.3	67.5	77.3	79.3	78.4	80	81.5		
17	0	83.3	67.5	74.3	79.3	77.1	80	81.5		
18	0	83.3	67.5	71.8	78.2	76.3	80.2	81.8		
19	0	83.3	67.5	71.2	77.7	76.4	80.1	80.9		
20	0	83.3	67.5	71.2	77.7	76.4	80.1			

Table 16: TNR for CART with guaranteed training TPR > 85

1/Weight	**Maximum depths**									
	1	**2**	**3**	**4**	**5**	**6**	**7**	**8**	**9**	**10**
1										
2										
3										
4										
5			82.8							
6			**83.7**							
7			82.1							
8			81.7							
9				82.8	80.6					
10			78.3	81.1	80.4					
11			78.4	79.4	80.4					
12			76.9	80.9	79.5					
13			73.9	80.7	79.3	80				
14			67.3	77.5	79.3	78.5				
15			67.5	77.3	79.3	78.4				
16			67.5	77.3	79.3	78.4				
17	0		67.5	74.3	79.3	77.1				
18	0		67.5	71.8	78.2	76.3				
19	0		67.5	71.2	77.7	76.4				
20	0		67.5	71.2	77.7	76.4				

Table 17: TNR for CART with guaranteed training TPR > 90

1/Weight	**Maximum depths**									
	1	**2**	**3**	**4**	**5**	**6**	**7**	**8**	**9**	**10**
1										
2										
3										
4										
5										
6										
7										
8										
9										
10										
11										
12										
13			73.9							
14			67.3							
15			67.5							
16			67.5							
17	0		67.5	**74.3**						
18	0		67.5	71.8						
19	0		67.5	71.2						
20	0		67.5	71.2						

Table 18: TNR for CART with guaranteed training TPR > 95

1/Weight	**Maximum depths**									
	1	**2**	**3**	**4**	**5**	**6**	**7**	**8**	**9**	**10**
1										
2										
3										
4										
5										
6										
7										
8										
9										
10										
11										
12										
13										
14			67.3							
15			**67.5**							
16			67.5							
17	0		67.5							
18	0		67.5							
19	0		67.5							
20	0		67.5							