Using Optimal Decision Trees via Integer Programming in Business Applications

ERASMUS UNIVERSITY ROTTERDAM

Erasmus School of Economics

Bachelor Thesis Double Degree BSc^2 in Econometrics and Economics

Author

Lourens Vale (447398)

447398lv@student.eur.nl

Supervisor
prof. dr. I. Birbil

Second Assessor
dr. O. Karabag

Date Final Version: July 5, 2020

Abstract More and more interest is shown in interpretable machine learning methods because these are often more trusted in practise. One of the most widely considered tools in this area are decision trees, but popular heuristic decision tree methods still have their limitations, such as a high risk of overfitting. This paper shows optimal decision trees can be constructed using integer programming for binary classification problems. It will be analysed whether this method is ready to be used in practise by comparing its prediction performance and solving times with heuristic methods. In addition, the formulation will be extended to be able to construct its own topology while keeping it as small as possible to ensure interpretability and less chance of overfitting. For small trees and datasets we do find promising results. For larger datasets optimality is not always reached, but sub-optimal solutions seem sufficient as well. Still, many improvements have to be made before it can be widely applied in practise.

Key Words Decision Trees · Integer Programming · Machine Learning · Binary Classification

Contents

1 Introduction

Over the past few decades, the use of machine learning techniques in business applications has increased dramatically. Chui et al. (2018) estimated that the use of these new applications have the potential to create an additional $5T in value by 2020, mostly in marketing and sales and supply chain management. This huge gain can be achieved because machine learning techniques are able to discover new patterns and relations in data which can not be defined using traditional statistical techniques. The new insights can be used for improved decision making in the business processes which in turn can increase the competitive advantage of a firm (Bose & Mahapatra, 2001). This creates a continuous drive for firms to keep using and improving these new methods.

However, not all methods are as useful as they seem. Many popular machine learning models are classified as *black box* models, like *neural networks*, as their internal decisions and functioning are unclear to the end-user. For many business applications it is crucial that their models have a good interpretability because this leads to trusting the models in practise, or as Doshi-Velez and Kim (2017) stated: "The problem is that a single metric, such as classification accuracy, is an incomplete description of most real-world tasks." Furthermore, it is expected that a business analyst is more likely to implement a machine learning model if its decisions can be translated into understandable business terms (Kotsiantis, 2013). Consequently, one of the most used models nowadays are *decision trees* (DT), because they are easy to understand and their good performance on the classification of categorical data. This is because DTs consist of a sequence of decisions, comparable to human reasoning. At each decision point, the input data is split into separate groups by looking at comparable features of the data. This means that, at the end of the sequence, you end up with several groups of data characterised by similar features which are then given a label. Thereafter, a new data point simply follows the paths through the process that resemble the features of this data point and finally ends up at a label which classifies this data point. Additionally, it is possible to make a graphical representation of a DT in the form a flow chart, another advantaged leading to increased comprehensibility.

Nonetheless, finding an optimal DT which best classifies the data using the best possible splits is known as an NP-hard problem[1] (Laurent & Rivest, 1976), meaning that until this time no one has found an algorithm efficient enough to solve the problem. As such, many *heuristic* DT algorithms are developed over time which try to find a solution as close as possible to the optimum. It are mainly these methods which are now being widely used in practise, such as CART[2] (Breiman et al., 1984) or C4.5 (Quinlan, 1993). A heuristic method loses some accuracy because it is a sequential procedure which builds the DT step-wise (local) without looking at future decisions (global) (Bertsimas & Dunn, 2017). This leads to several disadvantages for using DTs in practise. At first, heuristic methods tend to create over-complex DTs to maximise its accuracy at the cost of its interpretability. This also explains the fact that DTs have the tendency to *overfit*. If there is a dominant feature present in the training data, the DT will

[1] "A problem is NP-hard if an algorithm for solving it can be translated into one for solving any NP-problem (nondeterministic polynomial time) problem." Weisstein, Eric W. *NP-Hard Problem.*

[2] CART stands for Classification And Regression Trees.

have a bias towards this feature, leading to poor out-of-sample prediction. A solution would be to construct a smaller tree but this might lose predictive power. Another option, and widely used, are more sophisticated methods such as *random forests* and *boosting*. These approaches consists of producing multiple trees which are later combined into a single solution with a better predictive power than single tree methods, but at the cost of interpretability (James et al., 2013).

In theory, an *optimal DT* could be obtained by constructing the complete tree at once whilst using full information of the other decisions made in the tree. Over the years, the rapid innovations in computational power have resulted in several attempts to actually find such methods. Recently, Günlük et al. (2018) provided a method which, with careful modelling, is able to solve an *mixed integer linear program* (MILP) for constructing a DT to optimality. Their method is devised for binary classification problems including both categorical and numerical data. They do find evidence that in some cases their algorithm is able to outperform CART, but in order to determine whether their method is ready to be implemented in day-to-day operations, more analysis and improvements are needed. For many business applications, the most practical way to use DTs would be a tree of small size, such that it can be easily interpreted and that it has a low risk of overfitting. Besides, it should be computed within reasonable time with an accuracy not significantly less than other methods. The computation time is of large importance because in many cases prediction models require frequent updating such that the models are constantly up to date with the newest available data. Chui et al. (2018) showed in a study that in 34% of the applications the models are refreshed at least monthly, of which 23% at least weekly. The question is, whether the method of Günlük et al. (2018) is able to compete with other popular methods based on the above mentioned points. This leads us to the following research question:

Is the method for constructing optimal decision trees via integer programming more useful in practise compared to popular heuristic methods in terms of accuracy and solving times?

In order to answer this research question, we will first analyze the feasibility of the method by Günlük et al. (2018) in terms of the computation times and solving processes. Second, we will compare the prediction performance of the optimal DTs with popular heuristic methods such as CART, random forests and boosting algorithms. Next to the practical relevance to determine whether it is preferable to use the optimal DTs in business applications, we will also focus on a scientific extension. A rescrition of the method by Günlük et al. (2018) is that the structure of the tree has to be determined beforehand and is fixed. Meaning that it is a confirmatory process where one can only compare several options of trees, but it is unable to find the best possible tree. Therefore, we will also try to extend their algorithm to a model which can construct the topology itself.

We did find similar results as Günlük et al. (2018) showing that the method is able to solve to optimality for small trees and datasets and outperforms modest trees constructed with CART. Additionally, we extended their work on the following points:

* Visualising and analysing the solving process to determine point of code termination.
* Trying a warm start to speed up the process.
* Tested the influence of an imbalance parameter in the objective function.
* Compared their method with random forests and boosting.
* Extended the formulation to be able to construct its own topology while keeping the tree as small as possible.

Here, we did find that for large datasets and DTs little solving progress is made after 10 minutes and a warm start did not seem to help. In terms of accuracy we found that random forests and boosting are performing better than the optimal DTs and that the imbalance parameter does not add extra value. The self constructing extension works well and results in smaller trees with accuracies comparable, or even better, to the fixed topologies.

The rest of this paper is structured as follows. We start with an extensive literature review in Section 2, where the first part contains an economic point of view of using DTs in business applications and the second part a summary of existing research into DTs. Next, the descriptive statistics and preparation of the data to fit the methodology are discussed in Section 3. In Section 4, we discuss the methodology of constructing optimal DTs and present our extension. Subsequently, the results are presented in Section 5 and we end with a discussion and conclusion in Section 6.

2 Literature Review

Economic Point of View Bose and Mahapatra (2001) reviewed a large number of computer science journals to examine which techniques and applications are being used across different industries. They find that over 48% of the applications at that time used a form of DTs. Furthermore, almost 50% of the applications was rooted in finance and marketing. A more recent study containing more than 400 use cases by Chui et al. (2018), shows the same results in terms of the broad usage of DTs. Compared to the previous study, it is interesting to see that nowadays machine learning techniques are being applied in almost every industry. Within firms the functions that show the largest applicability of these techniques are marketing and sales, supply chain management, and risk.

Next to the positive impact on the performance of firms, the use of machine learning and other applications of *artificial intelligence* (AI) is changing the way firms do business. Brynjolfsson and Mcafee (2017) noted that there are three levels at which changes occur, namely tasks and occupations, business process and business models. The first relates to freeing up workers on simple tasks that can be performed by machine learning models such that the workers can focus on critical problems. Secondly, business processes are changed due to optimisation algorithms which change the workflow in term of efficiency. Lastly, machine learning systems can create new opportunities for business models or outcompeting current business models. This shows that the use of machine learning applications is not expected to replace all of our human tasks, at least

not in the near feature. On the contrary, the systems will mostly act as a tool to complement human activities. Likewise, Bose and Mahapatra (2001) explained that new applications hardly operate as standalone systems but rather as an integrated system in other already existing systems. It can therefore be expected that clear insights in the decision making process of machine learning algorithms, such as DTs, creates a more valuable collaboration between the two. Because many systems lack the ability to give a clear explanation of its choices, this can lead to several problems as noted by Brynjolfsson and Mcafee (2017). First, an algorithm can create a bias towards a feature in the data which strongly influences the outcome. A popular example is a job applicant system which predicts whether an applicant is a good match for the firm. Here, it is possible that the system will select candidates based on their race or gender which of course goes against ethical norms. Moreover, when a systems makes errors it will be very difficult to identify where it goes wrong. In some cases it might therefore be preferable to opt for an interpretable model with less accuracy instead of a less interpretable model with a higher accuracy.

Bear mind that not all machine learning systems require interpretability, as pointed out by Doshi-Velez and Kim (2017). They explain that there is no explanation needed if unacceptable results do not lead to harmful consequences or if the system is completely integrated and validated such that the system is trusted. On the other hand, they mention that the need for interpretability arises from an incompleteness in the model or problem. These incompletenesses include the desire for scientific understanding, safety measures, ethics, mismatched objectives and multi-objective trade-offs, with match the problems as pointed out above by Brynjolfsson and Mcafee (2017). In addition, more explanation can lead to easier judgement on attributes such as fairness, privacy, reliability, causality and trust.

Decision Tree Development Although machine learning techniques such as DTs seem to be quite new, its first algorithms were developed around 50 years ago. Build upon these first algorithms, Breiman et al. (1984) constructed the method of Classification And Regression Trees (CART) which became the leading work for DT methods. CART can be seen as a heuristic or greedy method because of its *top-down* approach. The algorithm starts at the top and for each split further in the tree a separate optimisation problem is solved. After the complete tree is constructed, the tree is then *pruned* to a smaller size for generalisation purposes. By doing this, the chance of overfitting on the training data is reduced. Breiman et al. (1984) already noted that this top-down approach might result in a poorer accuracy than an algorithm which forms the complete tree in a single step but due to the lack of computational power this was not feasible back at the time. Later, many other methods were developed such as C4.5 by Quinlan (1993), which basically follows the same approach as CART but uses a different function to determine the splits.

Next to these single tree methods, various other algorithms were developed over time which consists of multiple trees with the goal to reduce the imperfections of single tree methods. By the *Law of Large Numbers* one can expect that multiple trees result in less variance and therefore

less chance of overfitting. One very well-known method is called *random forest*. The most simple approach includes multiple trees which are all trained on a random selection of the data and than later combined into a single prediction (Quinlan, 1996). Currently, many more random forest procedures exists which all boil down to generating a large number of trees and than perform some type of voting for final prediction (Breiman, 2001). Another popular method is *boosting*. Here, several weak models are combined into one strong model (Quinlan, 1996). Whereas random forests is a parallel learner, boosting can be classified as a sequential learner. One well-known boosting algorithm is called *AdaBoost* (Freund, Schapire, et al., 1996). Here, the DT is build iteratively where in each iteration the training data is weighted according to their previously predicted label. If a data sample is inaccurately classified, it gets a higher weight in the new model, created in the next iteration, to increase its probability on an accurate classification. Predictions are again made by means of voting on all the trained models. Although both random forest and boosting algorithms have a higher predictive power as single tree methods, they lack interpretability.

Since the last 30 years, when the computational power started to increase, there have been various attempts to find methods that construct single optimal DTs. Still, many relaxations had to be made in order to obtain a solution. For example, Bennett and Blue (1996) tried a method using linear optimisation and Norouzi, Collins, Johnson, Fleet, and Kohli (2015) using stochastic gradient decent but both neglected the integer nature of the problem. Apart from Günlük et al. (2018), the method that comes close to a model for optimal DTs is the work of Bertsimas and Dunn (2017). Still, they do not consider categorical variables and only treat real-valued data. However, they do propose a way to construct a multi-class classification DT whereas Günlük et al. (2018) solely focused on binary classification. Next to that, their algorithm is able to construct the structure of the tree itself, given a maximum dept. At each node will be determined whether to stop branching or to continue. If stopped branching, all data points will end up with the same leaf node. This was also the inspiration for the extension included in this paper on the method of Günlük et al. (2018).

3 Data

We will test the method of Günlük et al. (2018) by using several different datasets. The datasets *Student* and *Adult* are retrieved from Kaggle, *Heloc* is retrieved from FICO Explainable Machine Learning Challenge and the remaining seven from UCI Machine Learning Repository. All datasets are selected based on their binary classification framework. In order to fit the methodology, the datasets have to be translated into a set of binary vectors which will be explained in Section 3.2.

3.1 Descriptive Statistics

The descriptive statistics of the datasets are presented in Table 1. Here, the *#Groups* represent the number of different categorical or numerical variables (i.e. columns in a dataset) and

#Features represent the number of features contained in each group. For example, one of the groups in *Student* is the student's sex which consists of two categorical features, male and female. Lastly, *%Positive* represents the fraction of positive labels and *Type* whether the data contains categorical or numerical features. We do note that several datasets contain missing values. However, in order to be able to compare and extend the results of Günlük et al. (2018) in the best possible way, we use the same procedure of data processing as they used. Besides, the main goal of this paper is to test the algorithm of optimal DTs and not to provide the best model for these datasets. In business applications, data formatting is of course an essential part of building a model.

Table 1
Descriptive statistics of datasets.

Dataset	**#Samples**	**#Groups**	**#Features**	**%Positive**	**Type**[a]
Adult	48842	14	136	24%	c/n
Cancer	699	9	90	66%	n
Chess	3196	36	73	52%	c
Heart	267	22	44	79%	c
Heloc	10459	23	177	48%	c/n
Monks	432	6	17	50%	c
Mushroom	8124	22	117	52%	c
Student	395	32	113	67%	c/n
Tic-Tac-Toe	958	9	27	65%	c
Voting	435	16	48	39%	c

[a] Type of features: categorical (c) and/or numerical (n).

3.2 Data Preparation

The data requires a specific form to fit the methodology. Namely, each group has to be transformed into a binary unit vector where each element represents a feature of the specific group, in machine learning often referred to as *one-hot encoding*. In the case of numerical groups, it is possible to *bin* the range of values into K segments which than can be considered as K categorical features. In this case, deciles will be used as thresholds such that each numerical group contains up to 10 possible values. The set of groups will be indexed as $G = \{1, ..., |G|\}$, the total set of features as $J = \{1, ..., |J|\}$ and the sample set as $I = \{1, ..., N\}$, which can be split into a set containing positive labels I_+ and negative labels I_-. Additionally, $g(j) \in G$ represents the group containing feature $j \in J$ and $J(g)$ represents the set of features that are contained in group g. To give an understanding of this setting have look at the example shown in Tables 2 and 3.

Table 2
Example of data transformation (one-hot encoding) into the correct format.

Original[a]					*Transformed*				
I	**Target**	**Group X**	**Group Y**	**Group Z**	I	**Label**	**1**	**2**	**3**
1	Good	B	E	H	1	I_+	(0,1,0)	(0,1)	(0,0,1)
2	Bad	A	E	G	2	I_-	(1,0,0)	(0,1)	(0,1,0)
⋮	⋮	⋮	⋮	⋮	⋮	⋮	⋮	⋮	⋮
N	Good	C	D	F	N	I_+	(0,0,1)	(1,0)	(1,0,0)

[a] Assume that the features shown for each group represent all possible features for this group.

Table 3
Format overview of the groups and features of the data.

Notation	**Data**							
Group	X	X	X	Y	Y	Z	Z	Z
Feature	A	B	C	D	E	F	G	H
j	1	2	3	4	5	6	7	8
g	1	1	1	2	2	3	3	3
a^1	0	1	0	0	1	0	0	1

Here, we see that each group g is transformed into a binary unit vector with length $|J(g)|$, where the total number of features is equal to $|J| = 8$, with indices $J = \{1, 2, ..., 8\}$ representing the total set of features: {A,B,...,H}. To explain the notation and expressions have a look at the following examples: $J(3) = \{6, 7, 8\}$ as the group with index 3 contains features with indices 6,7 and 8, and $g(4) = 2$ as the feature with index 4 (in the original dataset feature D) belongs to the group with index 2 (in the original dataset Group Y). Finally, each data sample, $i \in I$, can be represented as a single binary vector containing all information, namely $a^i \in \{0, 1\}^{|J|}$, where a^i_j represents each element in this vector. For the first data sample in Table 2 this would imply $a^1 = \{0, 1, 0, 0, 1, 0, 0, 1\}$ with $a^1_0 = 0$, $a^1_1 = 1$, etc, as displayed on the last row of Table 3. These vectors are used in the formulation to construct optimal DTs.

4 Methodology

In this section, we will first introduce the needed terminology on DTs and explain the notation used throughout this paper. Secondly, the basic integer problem will be constructed step-by-step, followed by several possible improvements to the model in terms of computational performance and two extra constraints for the handling of numerical features and combinatorial branching. Lastly, we will introduce our extension to the formulation of Günlük et al. (2018).

4.1 Terminology and Notation

DTs are a form of *supervised* models which implies that the algorithm is trained using labelled input data (*training set*) containing the desired output (James et al., 2013). The algorithm tries to construct a model such that most of the input data is correctly labelled. While doing this, the

model learns from comparing the correct label to the label given by the algorithm. Finally, the constructed model can be used to predict new unlabelled data points (*test set*). DTs can either be used for *regression* and *classification* problems. In this paper we solely focus on (binary) classification problems which means that the models predict either a 0 or a 1. In the case of regression problems the DT predicts real valued outcomes.

Figure 1 contains an example of a basic DT. The circles in the tree are called *decision nodes* which are indexed by $K = \{1, ..., |K|\}$. At these points the data is split into separate classes according to a test on the features of the data. The decision nodes 2 and 3 are called the *children* of node 1. Decision node 1 contains all the available data and is called the *root node*. The squares represent *leaf nodes* and are indexed by $B = \{1, ...|B|\}$. We can split the set of leaf nodes into $B_+ = \{2, 4, ...\}$ and $B_- = \{1, 3, ...\}$ which represent subsets containing positive (white) and negative (grey) labels respectively. By fixing the labels of the leaf nodes beforehand, we reduce the problem size. The complete structure of the tree, i.e. the number of decision nodes, splits and leaf nodes, is called the *topology* of the tree. As this tree has two levels of decisions, the *depth* of the tree is equal to two. Lastly, the tree below is *symmetric*, because each node has similar *subtrees* hanging at both *branches*.

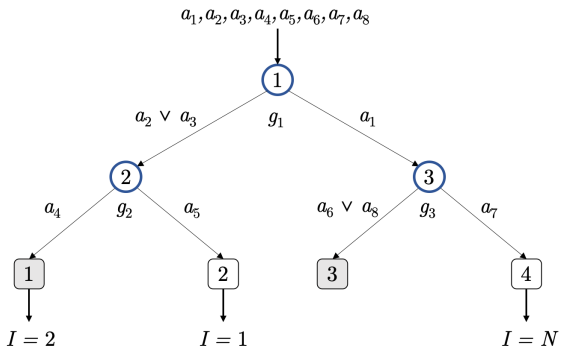


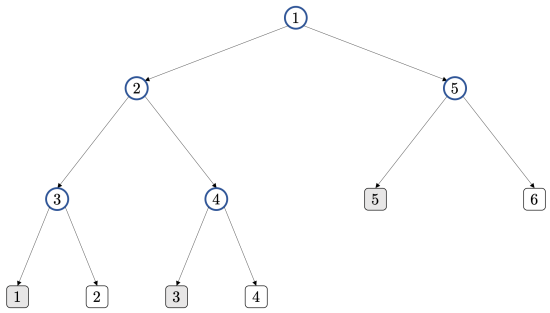
Figure 1. Basic DT with three decision nodes and four leaf nodes.

Let us now briefly describe the working of a DT by using Figure 1 together with the example data from Section 3.2. At the root node we branch on group 1, meaning the split will be made only based on the features contained in this group. In this case, if a_2 or a_3 are equal to 1 (either the feature B or C) the data sample will follow the left path and if a_1 is equal to one (feature A) the right path. The splits at nodes 2 and 3 are made in a similar way. Following this logic, the first and last data sample are labelled positive and the second sample negative. The *accuracy* of a model implies the fraction of data samples that are correctly classified, here three out of three. Another common way of measuring the prediction performance is using *sensitivity* and *specificity*. Sensitivity is also called the true positive rate (TPR), which can be calculated by dividing the number of positively classified data samples by the total number of positively labelled data samples in the dataset. Specificity, or true negative rate (TNR), implies the opposite with negative data samples. In practise, for some applications it is desirable to guarantee a level of TNR while maximising the TPR. One of the most used examples includes testing for a certain disease. One would like to maximise the number of correctly classified

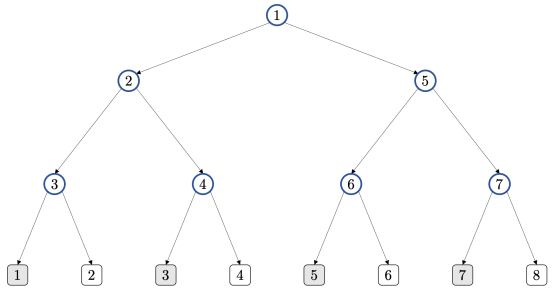
positive tests (TPR), as identifying a sick person as not sick, can have serious consequences. Here, the TNR is of less importance as identifying a person as sick which is not sick, will probably be corrected in the future.

4.2 Basic Model

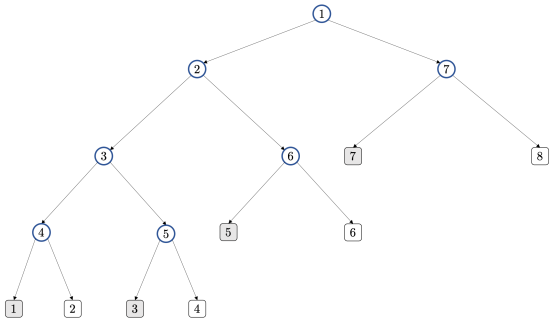
As explained earlier, in this paper optimal DTs will be constructed using an integer programming formulation. Below we present and explain the constraints needed for the basic model as formulated by Günlük et al. (2018). Important to notice is that the topology of the tree is predetermined, such that the number of leaf nodes and decision nodes is fixed. The topologies which are used throughout this paper are presented in Figure 2 together with the depth 2 tree from Figure 1.



(a) Topology: Depth3X



(b) Topology: Depth3



(c) Topology: Depth4X

Figure 2. Topologies used throughout this paper. X represents an imbalanced tree.

Notice that every solution of a subtree of a bigger tree is also a solution of the bigger tree. When looking at the example from Figure 1, we can extend this depth 2 tree to a depth 3 tree by first turning the leaf nodes into decision nodes and than simply routing all remaining data points left at $B = \{1, 3\}$ and right at $B = \{2, 4\}$, which yields the same objective value. Knowing this, an optimal solution of a bigger tree is therefore always at least as good as the optimal solution of a smaller subtree, because it can simply add more splits to the solution of the smaller tree. We do test different topologies because smaller trees tend to be more representative for testing data, i.e. less chance of overfitting, and larger trees are generally more difficult to solve.

4.2.1 Group and Feature Selection

First of all, for each decision node we have to impose a constraint that exactly one group is selected for branching. To do so, we introduce the first binary variable v_g^k, which is equal to 1 if group g is selected for branching at node k and else 0. In order to select exactly 1 group at each decision node, the sum over these variables should equal 1. This leads us to the first set of constraints:

$$\sum_{g \in G} v_g^k = 1 \quad \forall k \in K\,, \tag{1a}$$

$$v_g^k \in \{0,1\} \quad \forall k \in K\,, \forall g \in G\,. \tag{1b}$$

Secondly, once a group is selected for branching, we have to impose a constraint that only the features belonging to this specific group can be selected. This requires a second binary variable z_j^k, which is equal to 1 if feature j is selected for branching at decision node k and else 0. In this formulation, we mean with *selection* that a data sample with one of the selected features follows the *left* branch. Returning to the example from Section 4.1, we can see that at the root node feature 2 and 3 are selected for branching because they lead samples with these features to the left ($z_2^1 = z_3^1 = 1$) and feature 1 is not selected because this leads to the right ($z_1^1 = 0$). To check whether a feature belongs to a certain group, we can now use the previously defined function $g(j)$ for the next set of constraints:

$$z_j^k \leq v_{g(j)}^k \quad \forall j \in J\,, \forall k \in K\,, \tag{2a}$$

$$z_j^k \in \{0,1\} \quad \forall j \in J\,, \forall k \in K\,. \tag{2b}$$

Note that now z_j^k can only be 1 if $v_{g(j)}^k$ is equal to 1. As constraint (2a) is defined separately for each feature, it is possible that all features follow the same branch at a particular node. In this case, all data samples follow the same branch at this point in the tree, say left, and the subtree hanging at the right branch will not be used anymore.

4.2.2 Routing Data to Leaf Nodes

Now that we can select the groups and features to split on at each decision node, we have to tie the data to the tree. Therefore, we need a third, and last, binary variable c_b^i which is equal to 1 if data sample i is routed to leaf node b and else 0. For data sample i to end up at this particular leaf node b, it should satisfy all the splits made at the previous decision nodes in such a way that is follows the exact path to end up at leaf node b. Therefore, at each decision node k we have to check whether the data sample i contains the selected feature j, such that it goes left if $a_j^i = 1$ and otherwise right. To do so, we construct the expression

$$L(i,k) = \sum_{j \in J} a_j^i z_j^k \quad \forall k \in K\,, \forall i \in I\,, \tag{3}$$

which is equal to 1 if and only if $a_j^i = z_j^k = 1$, which indicates that data sample i follows the left path at decision node k, and else 0. Analogously, the expression

$$R(i,k) = 1 - L(i,k) \quad \forall k \in K\,, \forall i \in I\,, \tag{4}$$

indicates whether data sample i follows the right path at decision node k. Next, we have to tie these expressions to the binary variable c_b^i. For this purpose, we first have to define the unique paths leading to each leaf node. We can do this by creating a subset for each leaf node containing the decision nodes where the left branch is followed, $K^L(b)$, and a subset containing the decision nodes where the right branch is followed, $K^R(b)$, such that $K^L(b) \cup K^R(b) = K(b) \subset K$. Again, to make these expressions clear, we refer to the example from Section 4.1 and take leaf node 2 for demonstration. Here, $K^L(2) = \{1\}$, $K^R(2) = \{2\}$ and $K(2) = \{1, 2\}$, because to end up at leaf node 2, a data sample has to go left at node 1 and right at node 2. Now, if the value of $L(i,k)$ for data sample i is equal to 1 for all decision nodes in $K^L(b)$, this means that the data sample follows all the left paths necessary to end up in leaf node b. Similar reasoning holds for $R(i,k)$, $K^R(b)$ and right paths. Now we can construct the last set of constraints for the basic formulation:

$$c_b^i \leq L(i,k) \quad \forall i \in I\,, \forall b \in B\,, \forall k \in K^L(b)\,, \tag{5a}$$

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

$$\sum_{b \in B} c_b^i = 1 \quad \forall i \in I\,, \tag{5c}$$

$$c_b^i \in \{0,1\} \quad \forall i \in I\,, \forall b \in B\,. \tag{5d}$$

From (5a) and (5b) we can see that a data sample only ends up at leaf node b ($c_b^i = 1$) if all the correct paths are followed, i.e. all $L(i,k) = 1$ and $R(i,k) = 1$ for $k \in K(b)$. Constraint (5c) is needed to ensure that each data sample i ends up in exactly 1 leaf node.

4.2.3 Objective Function

The only part that is left is the objective function of the integer formulation. To recall, the goal of a DT is to correctly classify as much data samples as possible. We can simply adhere to this goal by maximising the sum of correct classifications through

$$\sum_{i \in I_+} \sum_{b \in B_+} c_b^i + C \sum_{i \in I_-} \sum_{b \in B_-} c_b^i, \tag{6}$$

where the left summation corresponds to positive labels and the right summation to negative labels. The constant C represents a parameter to adjust for *class imbalance*, which by default is equal to 1. In the case the number of positive and negative labels of a dataset are not equal, such as in *Adult*, it might be wise to adjust C resulting in equal weights to both sets of labels. The constraint sets (1), (2) and (5) together with the objective from expression (6) form a complete

formulation for the construction of DTs, as presented in Appendix A - Basic Formulation.

Günlük et al. (2018) also provided a different objective function, in combination with a new set of constraints, which maximises sensitivity while guarantying a certain level of specificity. This is a very useful addition to the formulation for many practical applications, especially because this cannot be addressed directly by CART or many other heuristic methods. The constraints that should be added are

$$\sum_{i \in I_-} \sum_{b \in B_-} c_b^i \geq \lceil LB \times |I_-| \rceil \quad \forall i \in I_- \, , \forall b \in B_- \, , \tag{7}$$

where $0 \leq LB \leq 1$ represents the guaranteed level of specificity. Secondly, the objective function should be changed to

$$\sum_{i \in I_+} \sum_{b \in B_+} c_b^i, \tag{8}$$

which now solely focuses on classifying positive labels. Of course, it is also possible to reverse this model, such that the specificity is maximized while guaranteeing a certain level of sensitivity. In that case all minus and positive signs from (7) and (8) should be exchanged.

The most common way to solve the integer problem, and also used by modern solvers, is by using the *branch and bound* algorithm (Wolsey, 1998). Branch and bound is an efficient method to solve (mixed) integer problems by using a tree search. The method first relaxes the problem to an LP problem and checks whether the solution is integer valued. If not, it tries to find a new integer valued solution by starting a tree search.

4.3 Reducing Problem Size

Günlük et al. (2018) noted that the basic formulation can be improved in terms of computational performance. Many binary variables can be relaxed to continuous variables and it contains more constraints than needed. Below we will discuss three different improvements that can be made through relaxation of variables, strengthening of the constraints for a tighter formulation and breaking of symmetry. In the results section the effect of these changes will be tested. Apart from these changes in the formulation, the objective from expression (6) and constraints (1a) and (2a) remain unchanged. A complete formulation with all the improvements included can be found in Appendix A - Improved Formulation.

4.3.1 Relaxation

In this non-polynomial optimisation problem, computational difficulty increases in the number of integer variables. Hence, one should try to reduce as much integrality as possible. In that case, the formulation becomes a *mixed* integer problem as it then contains binary and continuous variables. First of all, we can relax the integrality of c_b^i because we use this variable to maximise the objective function. Hence, an optimal solution will always choose the upper-bound of the

variable, which is 1. Furthermore, Günlük et al. (2018) show that all the extreme point solutions of the optimisation problem are integral, even when also relaxing the integrality of all v_g^k variables, and z_j^k variables for decision nodes adjacent to leaf nodes, indicated by K^L [Proposition 3, p. 13]. These relaxations imply that we can exchange constraints (1b), (2b) and (5d) with:

$$0 \leq v_g^k \leq 1 \quad \forall g \in G\,, \forall k \in K\,, \tag{9a}$$

$$0 \leq c_b^i \leq 1 \quad \forall b \in B\,, \forall i \in I\,, \tag{9b}$$

$$0 \leq z_j^k \leq 1 \quad \forall j \in J\,, \forall k \in K^L\,, \tag{9c}$$

$$z_j^k \in \{0,1\} \quad \forall j \in J\,, \forall k \in K \backslash K^L\,. \tag{9d}$$

4.3.2 Strengthening

Next, we focus on constraints (5a), (5b) and (5c). In the case when $L(i,k) = 0$ for some data sample i at decision node k, this implies that the sample is routed to the right. This also means that the leaf nodes which can only be reached by going left at this particular decision node, will never be reached. It is therefore unnecessary to include these leaf nodes in the constraint and allows us to form a tighter formulation. Therefore, we can replace constraint (5a) and (5b) with:

$$\sum_{b \in B : K^L(b) \ni k} c_b^i \leq L(i,k) \quad \forall i \in I\,, \forall k \in K\,, \tag{10a}$$

$$\sum_{b \in B : K^R(b) \ni k} c_b^i \leq R(i,k) \quad \forall i \in I\,, \forall k \in K\,. \tag{10b}$$

Here, the iterator $b \in B : K^L(b) \ni k$ includes the leaf nodes which are reachable when going left at decision node k and the same holds for the iterator in constraint (10b) when going right. This also implies that we can drop (5c) from the formulation, which reduces the total number of constraints by I.

4.3.3 Breaking Symmetry

In the case of a *symmetric* integer program where the variables can be alternated without changing the structure, the search space is increased exponentially which is a problem for solvers. This problem arises for every decision node which has symmetric subtrees on both branches. From Figure 1 we can see that simply changing the left branch with the right branch yields the same objective value but a different solution. To prevent this, we can select an *anchor feature*, $j(g) \in J(g)$, for each group $g \in G$. In the case group g is selected for branching, the anchor feature $j(g)$ will always be selected to follow the left branch. This leads to an extra constraint to the formulation but decreases the search space significantly:

$$z_{j(g)}^k = v_g^k \quad \forall g \in G\,, \forall k \in K^*\,. \tag{11}$$

Note that this constraint should not hold for decision nodes adjacent to leaf nodes because then this would always route data samples with the anchor feature to a negative leaf node. Therefore, let K^* denote the decision nodes that are not adjacent to a leaf node and have symmetric subtrees hanging on the right and left branch.

4.4 Enhancing Prediction Performance and Interpretability

Until now we have presented a complete and efficient formulation to construct a DT. However, it is still possible to increase the interpretability of the tree. We propose two ways to do this, first by taking into account the ordinal nature of numerical variables in Section 4.4.1 and secondly by restricting combinatorial branching in Section 4.4.2. It is expected that these additions also result in a higher testing accuracy.

4.4.1 Numerical Features

As explained before, it is possible to bin a numerical group into K segments, and use these segments as categorical features. However, the previously defined constraints do not take into account the ordinal nature of numerical features. For example, when simply using bins as categorical features, a possible solution can be to select features belonging to the 3^{rd} and 6^{th} quantile to go left, which is of course hard to interpret. Therefore, we can consider to add an extra set of constraints which enforces to split numerical groups in an ordinal way, either "less than or equal to" or "greater than or equal to". These extra constraints are presented below:

$$z_j^k \geq z_{j+1}^k - w_g^k \qquad \forall j, j+1 \in J(g)\,, \forall k \in K\,, \forall g \in G^*\,, \tag{12a}$$

$$z_j^k \geq z_{j-1}^k - (1 - w_g^k) \qquad \forall j, j-1 \in J(g)\,, \forall k \in K\,, \forall g \in G^*\,, \tag{12b}$$

$$w_g^k \in \{0, 1\} \qquad \forall k \in K\,, \forall g \in G^*\,. \tag{12c}$$

Here, constraints (12a) and (12b) are defined separately to check either the higher or lower bin. Because this split can be made in two ways, this also requires an extra binary variable w_g^k, which is equal to 1 if the branching condition is of the form "greater than or equal to" and 0 when "less than or equal to". Furthermore, G^* represents the set of numerical groups.

4.4.2 Combinatorial Branching

Another potential risk leading to overfitting of the DT is combinatorial branching. For groups with a large number of features, i.e. high $|J(g)|$, heuristic methods have the inclination to select many features for branching which is hard to interpret. Besides, these decisions are only based on the training data and might not give a good representation of new data samples. We can therefore restrict the formulation to only make splits with a maximum number of features, *max.card*. This can be implemented using the following constraints:

$$\sum_{j \in J(g)} z_j^k \leq max.card + (\,|J(g)| - max.card\,) \times (1 - u_g^k) \qquad \forall k \in K\,, \forall g \in G^{**}\,, \tag{13a}$$

$$\sum_{j \in J(g)} z_j^k \geq (\,|J(g)| - max.card\,) - (\,|J(g)| - max.card\,) \times u_g^k \quad \forall k \in K\,, \forall g \in G^{**}\,, \tag{13b}$$

$$u_g^k \in \{0, 1\} \qquad \forall k \in K\,, \forall g \in G^{**}\,. \tag{13c}$$

Constraint (13a) can be seen as the upperbound and (13b) as the lowerbound. As each split consists of going either left or right, the branching condition also consists of two forms: "at most $max.card$" or "at least $(|J(g)| - max.card)$". Again, this requires a new binary variable u_g^k which is equal to 1 if the branching condition is of the form "at most $max.card$" or 0 if of the form "at least $(|J(g)| - max.card)$". G^{**} represent the categorical groups for which holds that $|J(g)| > max.card$. Moreover, caution has to be taken with the relaxation constraints from Section 4.3.1. When adding the constraints for combinatorial branching, an optimal solution will not always be integer valued, meaning that we cannot use relaxation and combinatorial branching constraints at the same time.

4.5 Extension: Pruning Decision Tree

Günlük et al. (2018) stated that their formulation is not able to construct the topology itself, which when looking only at the structure and number of decision and leaf nodes is true. But given the fact that all features are allowed to follow the same branch at some particular node, or in other words, not performing a split, means that this node can be removed (pruned). We used this logic to define a reward for the objective function in the case a decision node does not branch, which forces the formulation to train the DT as small as possible because this simply results in a higher objective value. Note that a maximum allowed depth of the tree is needed in order to define the search space. To implement this reward in the previously defined formulation, we defined an indicator variable t_g^k, which is equal to 1 if all features of group g are routed either left or right at decision node k and otherwise 0, such that:

$$t_g^k = \begin{cases} 1, & \text{if } (\,\sum_{j \in J(g)} z_j^k = |J(g)| \wedge v_g^k = 1\,) \vee (\,\sum_{j \in J(g)} z_j^k = 0 \wedge v_g^k = 1\,) \\ 0, & \text{otherwise} \end{cases}. \tag{14}$$

Here, the first part, $\sum_{j \in J(g)} z_j^k = |J(g)| \wedge v_g^k = 1$, holds when all features of group g, denoted by $J(g)$, are selected to follow the left path at decision node k. Similarly, the second part, $\sum_{j \in J(g)} z_j^k = 0 \wedge v_g^k = 1$, holds when all features go right. Both conditions imply that this particular decision node is not used for branching and therefore t_g^k is set equal to 1[3]. Now, we can add t_g^k to the objective function through

[3]In IBM-CPLEX this can be implemented as follows: `model.add(t[g,k]==(nb_split==group_sizes[g]+v[g,k]==1+nb_split==0)==2)`. This sets t_g^k equal to 1 if two conditions are defined `True`, which is equivalent to formulation (14).

$$\textit{objective } (6) + T \sum_{g \in G} \sum_{k \in K} t_g^k = \sum_{i \in I_+} \sum_{b \in B_+} c_b^i + C \sum_{i \in I_-} \sum_{b \in B_-} c_b^i + T \sum_{g \in G} \sum_{k \in K} t_g^k. \tag{15}$$

The objective value now increases in case we do not use a decision node, such that the formulation tries to find a solution with the highest accuracy as possible, using the smallest possible tree. Given the fact that only one group can be selected for branching, the expression $\sum_{g \in G} t_g^k = \{0, 1\}$ holds for each decision node k. The parameter T gives a weight to the *pruned decision nodes*. In case $T = 1$, pruning 1 decision node is equal to classifying 1 data sample correct. As datasets differ a lot in size, it might be more useful to set the weight of T relative to the size of the dataset. In that case, we define the objective function as

$$\textit{objective } (6) + T \times \frac{N}{100} \sum_{g \in G} \sum_{k \in K} t_g^k \,, \tag{16}$$

where now pruning 1 decision node is equal in weight to T percent of correctly classified data samples. The weight T is an important factor because it can be seen as a parameter which controls the trade off between high accuracy and small trees. A low T results in high accuracy and a high T in a small tree. Because this constraint tries to construct a tree as small as possible, this also results in more interpretable trees with less risk of overfitting on the training set. Besides, instead of constructing the initial four topologies and comparing them for the best fit, it is now possible to let the formulation choose its own structure, and therefore relaxing the confirmatory nature of the process.

It is also possible to use this extension to the formulation in combination with the anchoring and strengthening constraints from Sections 4.3.3 and 4.3.2 respectively or with the sensitivity / specificity maximising objective function. The only issue is the degree of integrality as this extension only works when t_g^k is binary, plus relaxing the integrality of z_j^k, as in Section 4.3.1, is not possible because in modern solvers only integer variables can be used in equivalence constraints (14).

5 Results

In this section the optimal DT formulation will be thoroughly tested. All the results are obtained using IBM ILOG CPLEX 12.6.1 in Python 3.7 on a *MacBook Pro 2.9 GHz 2-Core Intel Core i5*. The first part consists of testing the feasibility of the model, i.e. solving times and finding the most efficient formulation to solve the problem. In the second part, the accuracy will be tested and compared with CART, random forest and boosting. Lastly, the extension from Section 4.5 will be tested.

5.1 Feasibility

When considering to use this DT method in day-to-day operations it should of course solve in reasonable time. In this section it will be analyzed how feasible this method actually is. Several

tests will be performed to give an as clear as possible view on the use of this method including the effect of different topologies and samplesizes. We also test the usage of a warm start to speed up the solving process.

5.1.1 Selecting the Most Efficient Formulation

In the methodology we proposed three computational improvements to the basic formulation. First, we will test to what extend these improvements have effect on the solving time of the problem. For each dataset we will run the following models: without all improvements (*Nothing*), without relaxing (*NoRelax*), without anchoring (*NoAnchor*), without strengthening (*NoStrenght*) or with all improvements (*All*). Here, we use a depth 2 tree and a training size of 200. If after 5 minutes the optimal solution is still not found, the solve is terminated and the best found sub-solution at that moment will be returned. The results are presented in Figure 3 from which we can derive that, overall, the model using all improvements is able to find the optimal solution in the smallest amount of time[4]. Furthermore, we can see that in most cases *NoStrength* results in longer computation times compared to *NoRelax* and *NoAnchor*, meaning that strenghtening the constraints (Section 4.3.2) has the largest effect. We can also derive from this figure that there exists large differences in the solving time per dataset. The datasets *Adult*, *Student* and *Heloc*, of which the latter is not included in the graph because it reached the time limit for each model, seem to be more difficult to solve than the others.

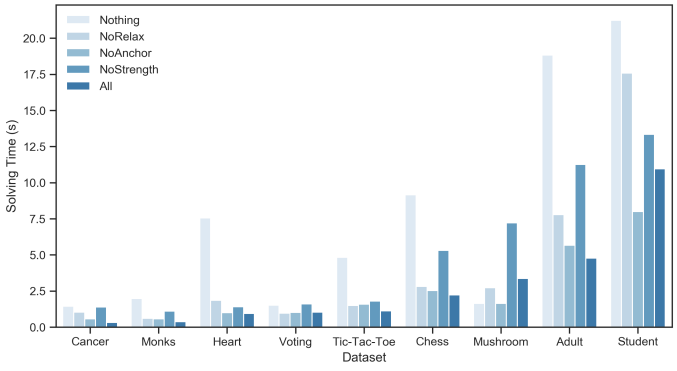


Figure 3. Solving times for depth 2 DTs, a training size of 200 and a time limit of 5 minutes.

5.1.2 Analyzing Influence of Feature Selection, Topology and Samplesize

We observed quite some differences in solving times between the datasets, which could be explained by the total number of features. Because some constraints are defined for each feature and therefore more features lead to an increase in the size of the problem. A solution to this would be to make a pre-selection of features which have the most explainable power and use only these features in the model. One way to do this is to first construct the DT using CART,

[4]From this moment onwards, all results are obtained using all improvements (relaxing, strengthening, anchoring) unless differently indicated.

and than only use the groups which show the largest importance for the MILP[5]. Throughout the rest of the paper, this will be referred to as *feature selection*. To test this effect, we solve a depth 2 model for each dataset with and without feature selection as presented in Table 4. This confirms that feature selection has a very large influence on the solving times[6]. However, interesting to see is that for the difficult datasets, the improvement is still rather small, which indicates that the difficulty also lies in other areas.

Table 4
Solving time in seconds using a samplesize of 200, depth 2 tree and a time limit of 5 minutes.

Selection[a]	Adult	BC	Chess	Heart	Heloc	Monks	Mush	Stud	TTT	Vote
False	3.52	1.24	2.41	0.89	41.74	0.25	0.21	9.82	1.09	1.24
True	2.85	0.33	0.53	0.29	10.52	0.03	0.09	3.85	0.26	0.12
Faster	×1.2	×3.8	×4.6	×3.1	×4.0	×8.3	×2.3	×2.6	×4.2	×10.3

Note: BC = Breast Cancer, Mush = Mushroom, Stud = Student and TTT = Tic-Tac-Toe.
[a] If *True* the feature selection procedure from Footnote 5 is used, if *False* all groups and features are used to construct the DT. *Faster* represents how much faster *True* solves than *False*.

Next, we test the influence of different samplesizes and topologies on the solving time, as presented in Figure 4. For these results the datasets *Cancer*, *Chess*, *Mushroom*, *Monks*, *Voting* and *Tic-Tac-Toe* are used and the averages are reported in the figure. The other datasets either contain smaller sample sizes or reached the time limit of 5 minutes for each model. It is clearly visible that the deeper the tree, the longer it takes to solve the problem, which can be explained by the fact that a larger tree contains a much larger search space. Furthermore, increasing the size of the training set shows a similar, exponential, trend.

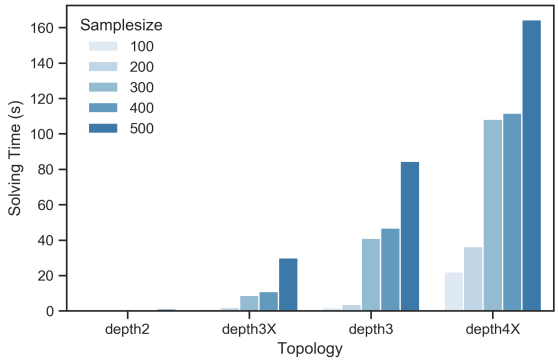


Figure 4. Average solving times using different topologies and samplesizes with a time limit of 5 minutes.

These results indicate that the optimal DT formulation is only solvable within reasonable time for small (max depth 4) DTs trained with a small (< 500) sample size. However, for larger samplesizes and more difficult trees and datasets, we can investigate to what extend it harms to terminate the solution after a few minutes and take the sub-optimal solution instead, which

[5]We used the function `.feature_importances_` from `sklearn` and took the average of the sum of squared feature importances for each group and than selected the top 50% most important groups for the MILP. The unused groups can simply be removed by setting the corresponding v variables to 0.
[6]From this moment onwards, all results are obtained using feature selection unless differently indicated.

was not addressed by Günlük et al. (2018). To test this, we solve the 'difficult' *Adult* dataset for samplesizes 100, 300 and 1000, a depth 3 tree and a time limit of 1 hour. During the solve, we keep track of the relative gap between the best integer objective and the objective of the best node remaining, as can be seen below in Figure 5.

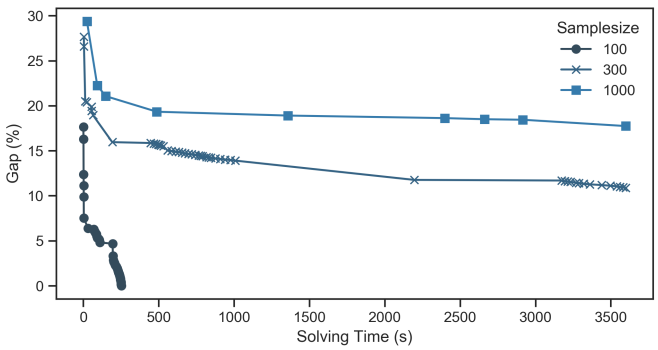
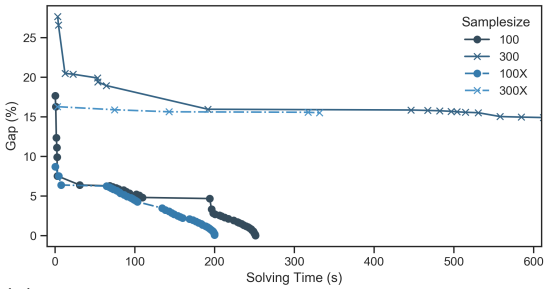


Figure 5. Solving time against optimality gap for dataset *Adult* with a tree of depth 3, feature selection and a time limit of one hour.

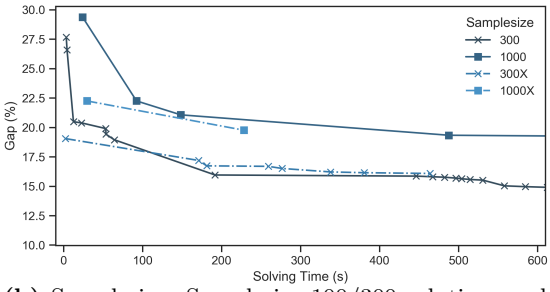
When the gap reaches 0% the optimal solution is found, as for the model trained using a samplesize of 100. For the larger training sets we see that at some point very little improvement is made. Both the solutions of the models using a training size of 300 and 1000 do not seem to improve after 5-10 minutes. This indicates that terminating the code after 5-10 minutes rather than after 1 hour or even longer does not seem to result in large differences[7]. This graph also shows that some optimal DTs, even with the formulation of Günlük et al. (2018), are still unable to solve to optimality, at least not on a normal laptop. But this does not mean that the sub-optimal solutions are useless. It is still possible that this 'almost' optimal solution outperforms current heuristic methods. We will see this in the next section. We also checked what will happen if we blow up the training size to 40.000. In that case the solver was not able to find even a single sub-solution in 30 minutes time. Hence, this might indicate that for very large datasets this formulation cannot be used.

Another potential way to speed up the solving process is to give the MILP a *warm start*. In this case, you provide a possible solution to the problem to start with, i.e. giving the problem a direction where to start searching. The property from Section 4.2 allows us to use a smaller tree as a *warm start* for a bigger tree. The other observation we made is that increasing the size of the training data also increases the solving time. Thus, we can also use the solution of a tree trained with a smaller training set as a *warm start* for a bigger training set. Both options are tested and presented in Figures 6a and 6b respectively. As we saw earlier that only little improvement is made after 10 minutes, we used this as the time limit for these results.

[7]From this moment onwards, all results are obtained using a time limit of 5 minutes (due to time constraints), unless differently indicated.



(a) Topology: Depth 2 solution used for solving a depth 3 tree.



(b) Samplesize: Samplesize 100/300 solution used for solving a tree with samplesize 300/1000.

Figure 6. Solving times for dataset *Adult* using *warm starts*, indicated by X.

From these figures we can see that the warm starts do have a positive effect on the solving process. For the model with training size 100 we see that using the warm start resulted in one minute less solving time. However, in the cases where the optimal solution was initially not found after one hour as shown in Figure 5, the warm start only has an effect on the short term but on the long term (> 5 minutes) it follows almost the same path as without using the warm start. Overall, keeping in mind that the warm start itself also takes some time to solve, this process does not seem to speed up the solving process.

5.2 Accuracy

Now that we know how the solving process reacts to different constraints, samplesizes and topologies, we can focus on the accuracy. Although the goal of DTs is to maximize the accuracy of the training set, the testing accuracy is often of much larger importance as this represents the actual prediction performance of new data samples. Therefore, most conclusions regarding the accuracy of DTs will be drawn based on the testing accuracy.

5.2.1 Prediction Performance

First, we compare the accuracy of the optimal DTs with the accuracy obtained by construction a DT with CART. To make sure the CART DT stays interpretable and for comparison purposes, we select a maximum depth of 3. In order to reduce selection bias in our results, we generate for each dataset three random training samples by using $min\{500, 80\%\}$ of the data and report the averages of the training and testing accuracy[8]. For CART no time restrictions hold, which allows us to use the average accuracy of 5 random samples of 80% of the data. At the same time, we also test the effect of using *feature selection* on the accuracy. All these results can be found in Table 5, where for each dataset the highest testing accuracy is presented in bold.

[8]From this moment onwards, all results are obtained using these three random samples trained with $min\{500, 80\%\}$ of the data, unless differently indicated.

Table 5
The average training (testing) accuracy with and without feature selection.

Dataset	FS[a]	Depth2	Depth3X	Depth3	Depth4X	CART-D3
Adult	*Yes*	82.3 (81.2)	84.5 (79.3)	84.5 (78.5)	84.5 (78.4)	-
	No	83.1 (80.5)	84.7 (79.9)	84.7 (79.9)	83.8 (80.6)	82.3 (**82.0**)
Cancer	*Yes*	96.3 (94.5)	97.7 (93.8)	98.2 (93.3)	98.1 (94.3)	-
	No	96.4 (93.6)	97.7 (94.0)	98.0 (94.0)	98.3 (**95.1**)	95.0 (92.9)
Chess	*Yes*	87.5 (86.8)	94.1 (**93.8**)	94.1 (93.8)	94.6 (93.8)	-
	No	87.5 (86.8)	94.1 (93.8)	92.3 (91.5)	94.3 (93.0)	90.5 (90.2)
Heart	*Yes*	80.0 (74.7)	82.3 (74.7)	82.3 (74.7)	84.4 (75.3)	-
	No	80.0 (74.7)	82.3 (**75.3**)	82.6 (74.7)	85.1 (73.5)	81.6 (74.8)
Heloc	*Yes*	72.9 (**68.4**)	73.4 (66.3)	74.5 (66.2)	72.8 (65.4)	-
	No	73.0 (68.3)	72.5 (67.1)	73.3 (64.9)	70.1 (67.9)	67.2 (66.8)
Monks	*Yes*	78.6 (74.3)	85.1 (76.2)	89.9 (85.1)	100.0 (**100.0**)	-
	No	78.6 (74.3)	85.1 (76.2)	89.9 (85.1)	100.0 (100.0)	78.4 (77.9)
Mushroom	*Yes*	99.3 (99.4)	99.9 (**99.8**)	100.0 (99.8)	100.0 (99.6)	-
	No	99.3 (99.4)	99.9 (99.5)	99.9 (99.2)	100.0 (99.7)	98.5 (98.4)
Student	*Yes*	92.5 (91.6)	93.9 (**93.2**)	93.9 (91.6)	93.8 (90.3)	-
	No	92.5 (92.0)	93.6 (92.8)	94.1 (88.6)	94.0 (91.1)	87.8 (86.1)
Tic-Tac-Toe	*Yes*	71.9 (68.8)	76.4 (73.3)	77.5 (73.4)	78.3 (**76.1**)	-
	No	71.9 (68.8)	77.8 (72.5)	78.5 (73.7)	77.0 (70.7)	75.3 (74.2)
Voting	*Yes*	95.7 (96.9)	96.5 (96.2)	97.0 (95.8)	97.7 (95.8)	-
	No	95.7 (96.9)	96.6 (95.8)	96.9 (**97.3**)	97.2 (96.9)	96.7 (96.6)

[a] FS = Feature Selection. If *yes* the feature selection procedure from Footnote 5 is used, if *no* all groups and features are used to construct the DT.

We can see that the optimal DTs in all cases outperform the depth 3 DTs of CART, except for *Adult*. Between the different topologies only small differences occur, with an exception for *Monks*, where a depth 4X tree seems to perfectly describe both training and testing data. We do see that bigger trees are always able to achieve a higher training accuracy than smaller subtrees, as stated before in Section 4.2. Furthermore, we can see that feature selection has little effects on the accuracy but a clear trend is not visible. We do see, however, that on average both training and testing accuracy improved slightly. This can be explained by the fact that feature selection results in a faster solving process as we saw earlier. When reaching the time limit, the model with feature selection is probably in some cases few steps ahead of a model without using feature selection. Hence, for both computational speedup and accuracy purposes feature selection is recommended. Note that in some cases, such as for *Monks*, the accuracy with and without feature selection is the same, which is possible when both models solve within the time limit and therefore both reach the same solution.

Many solves from the previous table reached the time limit and therefore sub-optimal solutions were reported. This proves that sub-optimal solutions after only 5 minutes are already able to outperform CART-D3. As we saw earlier in Figure 5, MILP problems for difficult or large datasets seem to never find an optimal solution. We can create a similar figure to investigate the effect of longer computation times on the accuracy, as presented in Figure 7. This again shows

that at some point, also in terms of accuracy, very little improvement is made. It is therefore unnecessary to keep your model running for more than 10 minutes.

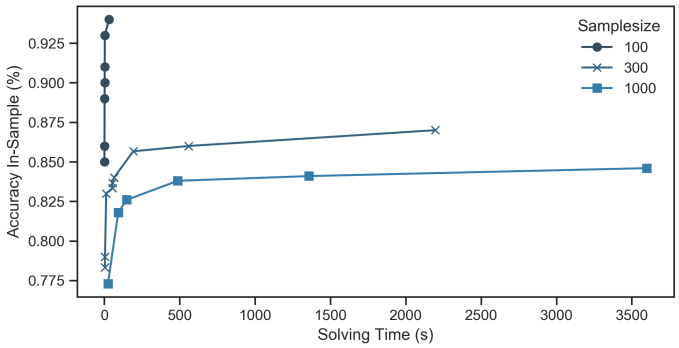


Figure 7. Solving time against in-sample accuracy for dataset *Adult* with a tree of depth 3, feature selection a time limit of one hour.

This figure also shows that using a larger samplesize results in a lower training accuracy. To analyse the effects of a larger sample size on both the training and testing accuracy we solve each model for a samplesize of 100, 200 and 400 using a depth 3 tree. The results can be found in Figure 8, where the light-blue and dark-blue bars represent testing and training accuracy respectively. Overall, we can see that increasing the training size results in a smaller gap between training and testing accuracy. This effect can be explained by noting that a larger sample has presumably a better representation of the total population and therefore increases the testing accuracy. Consequently, this also means that it is less likely to overfit on the training data and therefore reduces the training accuracy. Another effect that is visible is that a larger samplesize might increase the size of the problem and therefore takes longer to solve and in turn results in a lower training and testing accuracy, such as for *Monks* and *Voting*.

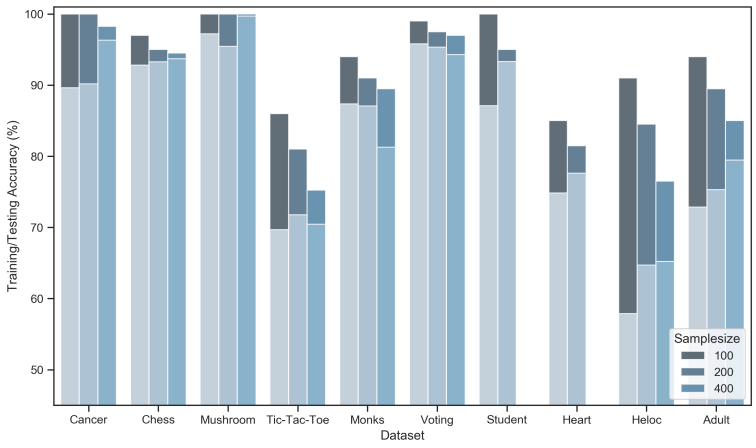


Figure 8. Results of increasing the samplesize on training and testing accuracy for depth3 trees. *Light blue* is testing accuracy and *Dark blue* is training accuracy.

5.2.2 Effect of Numerical Features, Combinatorial Branching and Imbalance Parameter

As explained in the methodology, the risk of overfitting can be reduced by constraining the number of features to branch on at each decision node. We can do this by adding the previously defined constraints (13) for combinatorial branching to the formulation. We test this effect on the datasets *Adult*, *Mushroom* and *Student* as these are the only datasets with groups containing more than 5 features. Three different branching possibilities are considered, namely where $max\_card = 1$, $max\_card = 2$ and the unconstrained version (*None*) similar to the results in Table 5. The results are presented in Table 6. As explained in Section 4.4.2, in this case no relaxation of the variables is used. We do see that constraining to $max\_card = 1$ for depth 2 trees might in some cases be too strong as it reduces both training and testing accuracy compared to unconstrained branching. On the other side, for $max\_card = 2$ and depth 3 trees we do see improvements in terms of testing accuracy compared to using the unconstrained branching rule. However, due to a lack of datasets with many features per group we cannot draw a clear conclusion whether combinatorial branching results in less overfitting and consequently a better prediction performance.

Table 6
Results of different combinatorial branching rules on training (testing) accuracy.

	Depth2			*Depth3*		
Dataset	**1**	**2**	**None**	**1**	**2**	**None**
Adult	81.5 (82.0)	81.6 (81.0)	82.3 (81.2)	82.7 (77.9)	82.3 (79.0)	84.5 (78.5)
Mushroom	96.9 (96.4)	99.3 (99.4)	99.3 (99.4)	99.8 (99.7)	100.0 (99.8)	100.0 (99.8)
Student	92.5 (91.1)	92.5 (92.4)	92.5 (91.6)	93.2 (92.0)	93.8 (92.0)	93.9 (91.6)

Next, we analyse the effect of using the extra constraints (12) for the ordinal nature of numerical features. To do so, we construct DTs for each dataset with numerical features with and without the extra constraints and report the training and testing accuracy in Table 7. We see that in most cases the testing accuracy improved while the training accuracy decreased. It is therefore recommended not to neglect the ordinal nature of numerical features as it improves the testing accuracy and the interpretability of the DT. Although adding constraints (12) to the formulation, the solving time reduced slightly compared to the formulation without these extra constraints. One explanation can be the possible reduction of the search space as with the numerical constraints included, less combinations of features to split on are possible.

Table 7
Effects of extra numerical constraints on training (testing) accuracy.

Dataset	c/n[a]	Depth2	Depth3X	Depth3	Depth4X
Adult	c	82.3 (81.2)	84.5 (79.3)	84.5 (78.5)	84.5 (78.4)
	n	82.3 (81.2)	84.5 (81.3)	84.0 (80.2)	83.8 (80.5)
Cancer	c	96.3 (94.5)	97.7 (93.8)	98.2 (93.3)	98.1 (94.3)
	n	95.7 (93.0)	96.9 (94.6)	97.2 (94.0)	97.3 (95.8)
Heloc	c	72.9 (68.4)	73.4 (66.3)	74.5 (66.2)	72.8 (65.4)
	n	72.3 (68.5)	73.2 (67.8)	73.2 (67.6)	71.8 (66.8)
Student	c	92.5 (91.6)	93.9 (93.2)	93.9 (91.6)	93.9 (90.3)
	n	91.7 (93.2)	93.2 (89.9)	93.0 (89.5)	92.7 (92.4)

[a] Here, c stand for categorical, such that constraints (12) are not used. n stands for numerical, such that constraints (12) are used.

Lastly, we examine the influence of the imbalance parameter C included in the objective function (6) which was not checked by Günlük et al. (2018). We compare the default value of $C = 1$ with $C = c$, where $c = \frac{|I_+|}{|I_-|}$[9] such that the positive and negative labels have equal weight. The results are shown in Table 8. We observe that in all cases both the training and testing accuracy decreases, except for *Cancer*. Defining the objective with the default $C = 1$ is therefore recommended. The only difference is that with choosing $C = c$, the fraction of correctly classified positive and negative labels is probably more equal, whereas in the default case the tree tends to focus more on the superior group.

Table 8
Effect of imbalance parameter C on average training (testing) accuracy.

		Depth2		*Depth3*	
Dataset	c [a]	$C = 1$	$C = c$	$C = 1$	$C = c$
Adult	0.32	82.3 (81.2)	81.7 (80.6)	84.5 (78.5)	83.4 (79.8)
Cancer	1.94	96.3 (94.5)	96.3 (95.7)	98.2 (93.3)	98.0 (94.3)
Heart	3.76	80.0 (74.7)	76.6 (66.1)	82.3 (74.7)	81.2 (69.7)
Student	2.03	92.5 (91.6)	91.6 (93.7)	93.9 (91.6)	92.7 (90.3)
Tic-Tac-Toe	1.86	71.9 (68.8)	68.6 (65.4)	77.5 (73.5)	74.1 (70.6)
Voting	0.64	95.7 (97.0)	93.9 (95.4)	97.0 (95.8)	96.8 (95.8)

[a] The presented value represents the value for the total dataset. For the accuracy the fractions of the random samples are used, which can slightly differ from the total dataset.

5.2.3 Maximising Sensitivity / Specificity

We also check the working of the other objective function (8) which maximises sensitivity (specificity) while guaranteeing a certain level of specificity (sensitivity). As this approach is highly applicable to health related datasets, we test it using the *Cancer* dataset. In this dataset the goal is to identify as much malignant breast tumors as possible, which are labelled negative. We can implement this into the formulation by maximising the specificity (TNR) while guaranteeing

[9] Caution has to be taken that this value should represent the fraction of the random sample taken from the total dataset. This can slightly differ from the value of the complete dataset.

a certain level (LB) of sensitivity (TPR), which includes identifying benign tumors. Note that we here use the reversed order as presented in Section 4.2.3. The results for depth 2 and depth 3 trees using several values of LB are presented in Table 9.

Table 9
The average training (testing) accuracy, TPR and TNR for dataset Cancer. The TNR is maximised, while the TPR is constraint to a lowerbound (LB).

		Training			*Testing*		
Topology	**LB$_{\text{TPR}}$**[a]	**TNR**	**TPR**	**Acc**	**TNR**	**TPR**	**Acc**
Depth2	0.95	98.2	95.3	96.3	95.4	95.0	95.1
	0.96	96.4	96.2	96.3	95.4	96.3	96.0
	0.97	93.8	97.2	96.1	93.6	97.1	95.8
	0.98	91.2	98.2	95.9	88.8	96.8	93.8
	0.99	86.5	99.1	94.9	84.7	98.1	93.1
	1.00	76.9	100.0	92.3	75.1	98.9	90.1
Depth3	0.95	99.8	95.3	96.8	94.5	95.2	95.0
	0.96	100.0	96.2	97.5	95.7	94.4	95.0
	0.97	98.8	97.2	97.7	92.7	95.0	94.1
	0.98	97.4	98.2	97.9	89.6	97.1	94.3
	0.99	94.4	99.1	97.5	83.6	98.1	92.8
	1.00	91.5	100.0	97.1	83.7	97.1	92.1

Note: TPR = sensitivity, TNR = specificity and Acc = Accuracy.
[a] Guaranteed level of sensitivity (TPR) implemented in the constraints (7).

From this table we can see that, for low values of LB, the tree is able to identify more malignant tumors than benign tumors, which is what we desired. For the depth 3 tree with $LB = 0.96$ even a training TNR of 100% is reached. As the LB increases towards 1, we see that guaranteeing this constraint goes at the expensive of the TNR. However, we also see that the guaranteed level of TPR is not reached in the testing set. To make this more visible we refer to Figure 9, where we see that for values of $LB > 0.95$, the guaranteed level of TPR is almost never reached in practise. We do see that the depth 2 trees more closely maintain the desired level of LB than depth 3 trees. The testing accuracy is also in most cases higher than the testing TNR, which might imply that setting $LB \geq 0.95$ is too strict.

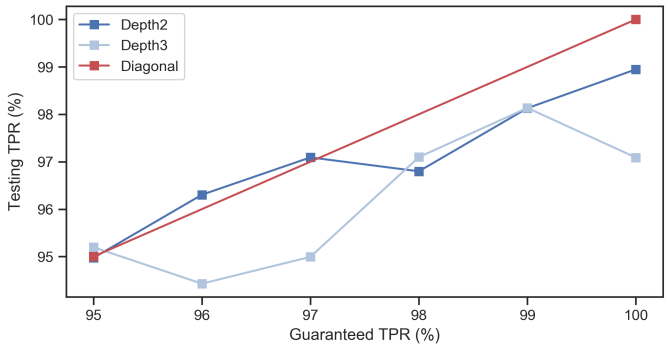


Figure 9. Testing sensitivity for *Cancer* dataset against several guaranteed specificity values.

5.2.4 Comparing with CART, Random Forest and Boosting

In Table 5 we saw that the MILP is able to construct (sub) optimal DTs which outperform CART-D3 trees. However, what if we neglect the interpretability of the solution and solely focus on maximising the testing accuracy. It is also of high importance to see if the optimal DTs can compete with well performing methods such as random forest and boosting. We therefore compare the best found optimal DT solution from previous tests with unconstrained CART, random forest and boosting (*AdaBoost*), see Table 10. In comparison to the optimal DT, the heuristics solve in matter of seconds and therefore do not need a small fraction of the data to train on. Therefore, we used 5 random samples of 80% of the total data.

Table 10
MILP vs. Heuristics (default settings) in terms of training (testing) accuracy.

	MILP		*Heuristics*		
Dataset	**Accuracy**	**Topo**	**CART**	**RF**	**Boost**
Adult	81.5 (81.9)	D2	96.5 (78.4)	95.5 (81.6)	83.9 (**83.5**)
Cancer	96.3 (94.5)	D2	100.0 (93.4)	99.7 (**96.1**)	99.7 (95.1)
Chess	94.6 (93.8)	D4X	100.0 (**99.6**)	100.0 (98.7)	97.0 (96.4)
Heart	84.4 (75.3)	D4X	94.7 (71.9)	94.4 (79.3)	86.6 (**81.5**)
Heloc	72.9 (68.4)	D2	97.4 (62.5)	96.0 (69.3)	72.4 (**72.5**)
Monks	100.0 (**100.0**)	D4X	100.0 (87.8)	99.9 (91.5)	75.0 (74.9)
Mushroom	100.0 (99.8)	D3	100.0 (**100.0**)	100.0 (100.0)	100.0 (100.0)
Student	93.9 (**93.2**)	D3X	100.0 (87.8)	99.8 (87.3)	98.2 (88.9)
Tic-Tac-Toe	78.3 (76.1)	D4X	100.0 (93.8)	100.0 (**95.0**)	84.7 (84.6)
Voting	95.7 (96.9)	D2	100.0 (94.0)	99.7 (97.0)	98.4 (**97.7**)

Note: Topo = Topology, RF = Random Forest and Boost = Boosting.

Overall, we see that CART, random forests and boosting are still much stronger prediction methods as they jointly outperform the MILP in eight out of ten datasets, however at the cost of interpretability. Note that *Monks* and *Student* reach an optimal solution for the MILP and probably therefore outperform the heuristic methods. For the other datasets, either the solved depth 2 model might be too simple to represent the whole dataset or the program was terminated and therefore no optimal solution was found. We see that CART is tempting to overfit on the training data as for many datasets a training accuracy of 100% is reached. However, for datasets with little variety such as *Mushroom* and *Chess*, this is no problem because then the training set is always representative for the testing set, hence resulting in a similar testing accuracy.

5.3 Extension: Pruning Decision Tree

At last, we check the results when using the extended formulation from Section 4.5, where we let the formulation choose the best and small as possible topology, using a maximum depth of 4. We compare four different values of T, using the objective function from equation (16). Again we use the same random splits as before and a time limit of 5 minutes. As explained earlier, the relaxation constraints (9) are turned off. The results are shown in Table 11.

Table 11
Average values of training (testing) accuracy and decision nodes for pruned DTs.

Dataset	T	Accuracy	Nodes	Time[a]
Chess	0.1	94.1 (93.9)	5.0	*
	0.5	93.9 (93.7)	4.7	*
	1	93.1 (92.6)	4.3	*
	2	92.7 (91.4)	3.3	*
Heloc	0.1	67.0 (66.4)	4.7	*
	0.5	61.3 (58.6)	1.7	*
	1	69.2 (65.2)	2.7	*
	2	68.3 (65.6)	2.0	*
Monks	0.1	100.0 (100.0)	6.0	6.8
	0.5	100.0 (100.0)	6.0	5.3
	1	100.0 (100.0)	6.0	7.5
	2	100.0 (100.0)	6.0	9.6
Mushroom	0.1	100.0 (99.9)	5.3	*
	0.5	99.2 (99.1)	1.7	*
	1	98.9 (98.8)	1.3	266.1
	2	98.5 (98.5)	1.0	253.5
Tic-Tac-Toe	0.1	76.9 (70.6)	7.3	*
	0.5	77.2 (72.3)	5.7	*
	1	75.3 (72.0)	3.7	*
	2	75.0 (71.6)	3.3	*
Voting	0.1	98.0 (96.2)	9.0	*
	0.5	95.5 (97.7)	1.7	*
	1	95.0 (98.1)	1.0	*
	2	95.0 (98.1)	1.0	75.2

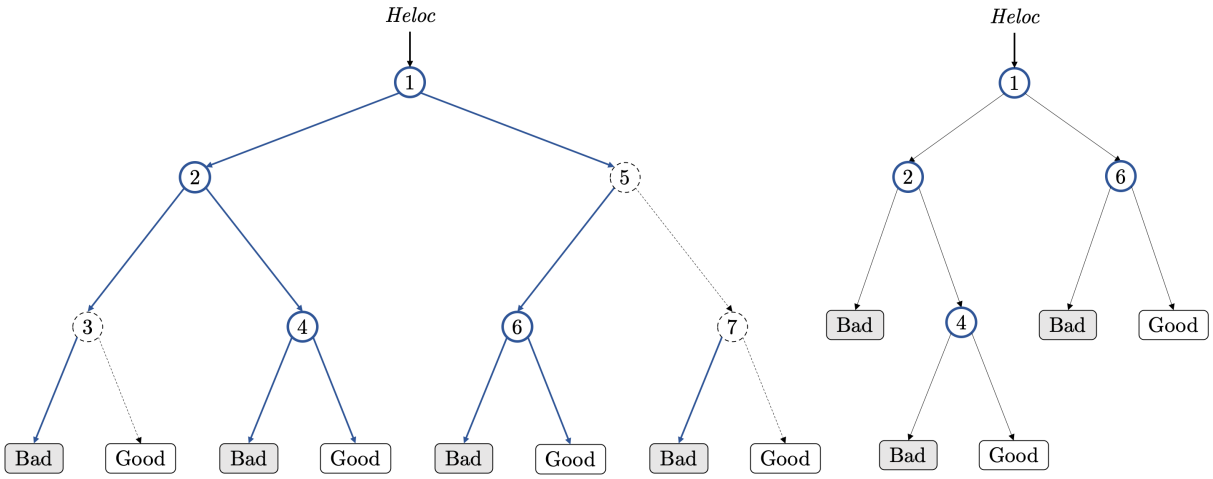
[a] A * denotes the time limit of 5 minutes is reached.

From this table we can derive that setting T equal to 0.1, which means that pruning one decision node is equal in weight to 0.1% of correctly classified data samples, already results in DTs quite similar to the best found trees from Table 10. We see that for *Monks*, *Chess*, *Voting* and *Mushroom* even smaller DTs exists with the same or even better testing accuracy as found before. *Heloc* and *Tic-Tac-Toe* belong to the more difficult datasets and perform slightly less as before. This is probably because now the search space is much bigger, however the results do show that smaller DTs exist with a testing accuracy close to the previously found. Besides, instead of running four different topologies for 5 minutes we can now try to solve only one problem in 20 minutes, which might result in even better DTs. Lastly, setting T equal to 2 or larger might be too strong as for some datasets in this case only 1 decision node is chosen to branch on.

It is also possible to combine this extension with the sensitivity / specificity maximising objective function and numerical features constraint. When constructing a tree for the *Cancer* dataset using a maximum depth 3, $T = 0.1$, $LB = 0.95$, numerical features and a time limit of

10 minutes a testing TNR of 98% is reached. This is higher than any value of testing TNR from Table 9.

To give a demonstration of how the topology construction works, we have provided a solution of dataset *Heloc* in Figure 10 below. In order to obtain this result, a maximum depth of 3 was selected and a samplesize of 500 was used. Furthermore, relaxing was turned off, numerical features were turned on and the time limit of 10 minutes was reached. Figure 10a shows the complete tree where the bold arrows represent the branches that are included in the solution. This means that node 3,5 and 7 are not used and can therefore be pruned, such that we end up with the topology presented in Figure 10b.



(a) Solution where nodes 3,5 and 7 are not used for branching. **(b)** Reduced topology.

Figure 10. Construction of topology for *Heloc* dataset using pruning constraint.

This simple interpretable tree with just four decision nodes reached a training accuracy of 71% and testing accuracy of 69%. For comparison, the tree constructed by CART (with default settings) from Table 10 has an average depth of 33 and is therefore not interpretable at all, plus its testing accuracy is significantly lower. We saw that boosting reached a higher testing accuracy of 72.5% but this solution does also not provide us with a clear explanation. The only alternative which can be easily interpreted is the CART-D3 tree from Table 5. However, this reached a testing accuracy of 67% and consisted of 7 decision nodes.

When linking the actual data to Figure 10b we get the tree as presented in Figure 11. This tree provides you with a lot of extra information other heuristic methods are not able to give. As stated on the website of FICO: "A HELOC is a line of credit typically offered by a bank as a percentage of home equity. The fundamental task is to use the information about the applicant in their credit report to predict whether they will repay their HELOC account within 2 years. This prediction is then used to decide whether the homeowner qualifies for a line of credit and, if so, how much credit should be extended." The value *Bad* indicates that a consumer at least once was past due on a payment and the value *Good* indicates all payments were made on time. With this DT we now know that the months a consumer is included in the file of a credit extender

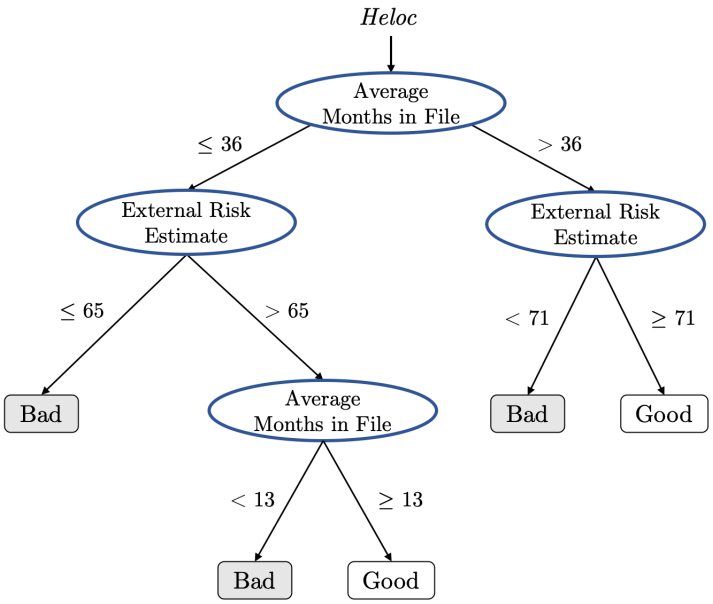


Figure 11. Interpretable decision tree for *Heloc* dataset.

and the external risk estimate of the consumer are important variables in the decision making process. With the actual splits, a credit extender can identify *Bad* and *Good* consumers more easily. For example, if consumer has an external risk estimate of ≤ 65, this already shows quite some evidence that this consumers can be labelled *Bad*.

6 Conclusion, Discussion and Further Research

In this paper, the method proposed by Günlük et al. (2018) to construct optimal DTs using integer programming is thoroughly tested and extended. Next to the basic formulation, several other additions were given, such as the handling of numerical features and combinatorial branching. The goal was to test to what extend optimal DTs can be used in practical applications and if they are competitive enough with popular heuristic methods.

We obtained similar results as Günlük et al. (2018) on the first set of tests. This included the observation that the MILP is solvable in reasonable time (< 5 minutes) for small datasets and DTs with a maximum depth of 3. These optimal trees showed to outperform DTs constructed with CART, restricted to maximum depth of 3, based on their testing accuracy. We also demonstrated that increasing the size of the training set and increasing the difficulty of the topology, both largely increase the size of the problem to a point where no optimal solution is found. Nevertheless, the sub-solutions still outperformed CART-D3.

Our extensions were mainly based on testing the practical applicability and the attempt to make the formulation more useful in practise. At first we investigated what the consequences are of terminating the solve after a specific amount of time. This showed that after 5-10 minutes very little improvements to the solutions are made. Hence, keeping the solve running for hours does not result in better solutions. Secondly, we tried using a solution of a simpler (smaller training set or smaller topology) model as a warm start for a more difficult model. This showed

to have a positive effect on the short term but on the long term the same little progress as before was observed. Therefore, using a warm start does not result in better solutions. Another important finding showed that this method is not able to find any solution in 30 minutes time when using a samplesize of 40.000, which of course is a big problem for firms with large datasets.

Next, when loosening the interpretability of the heuristic methods we do see another pattern. In almost all cases the testing accuracy of solutions from random forests and boosting algorithms outperform the testing accuracy of the MILP instance. This shows that the optimal DTs are not able to compete with other widely used methods, yet.

Our most promising extension included the newly added ability of the formulation to construct the topology of the DT itself, while keeping it as small as possible. With a tuning parameter it can be determined how strong the formulation should focus on pruning decision nodes or on reaching the highest accuracy. Almost all solutions obtained with this extension consisted of fewer decision nodes with similar, or in some cases even better, testing accuracies as found before. This extension is also time saving as now only one model can be solved to optimality instead of trying four different topologies. Furthermore, as smaller trees are rewarded, this subsequently leads to less risk of overfitting on the training data and a higher interpretability.

There are, however, some points of attention when analyzing these results. First of all, due to time limitations a time limit of 5 minutes was chosen for each solve, whereas the results showed there is still considerable progress made until 10 minutes after the start of the solve. To add to that, there appeared to exist quit some difference in the prediction metrics of the random data splits meaning that using 5 instead of 3 random splits might give a better representation of the actual results. Secondly, the obtained results are highly dependent on the computational power used to solve the problems. However, this also implies that there exist the potential to obtain even better results as now only a 'normal' laptop is used. It might be interesting to see how this method performs when making use of cloud computing or other strong computation sources. These points of attention also explain the small differences between our results and those of Günlük et al. (2018).

There are several ways further research can contribute to the improvement of this method. First of all, the method is now only applicable for binary classification problems and can be generalised to also accept multivariate classification problems. Also, more research is needed to point out which value of T results in the perfect balance between accuracy and interpretability. Another possibility lies in the area of technical development, which includes increasing computational power.

All in all, this method does show to have the potential to become leader in interpretable forecasting methods but before it can be widely used in practise significant improvements in terms of the solving process have to be made such that it can handle larger datasets and solves much faster.

References

Bennett, K. P., & Blue, J. A. (1996). Optimal decision trees. *Rensselaer Polytechnic Institute Math Report*, *214*, 24.

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

Bose, I., & Mahapatra, R. K. (2001). Business data mining - a machine learning perspective. *Information & Management*, *39*(3), 211–225.

Breiman, L. (2001). Random forests. *Machine Learning*, *45*(1), 5–32.

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

Brynjolfsson, E., & Mcafee, A. (2017). The business of artificial intelligence. *Harvard Business Review*, 1–20.

Chui, M., Manyika, J., Miremadi, M., Henke, N., Chung, R., Nel, P., & Malhotra, S. (2018). Notes from the AI frontier: Applications and value of deep learning. *McKinsey Global Institute Discussion Paper, April*.

Doshi-Velez, F., & Kim, B. (2017). Towards a rigorous science of interpretable machine learning. *Last access date: 15-06-2020*.

FICO Explainable Machine Learning Challenge. (n.d.). Retrieved 15-05-2020, from https://community.fico.com/s/explainable-machine-learning-challenge?tabset-3158a=2

Freund, Y., Schapire, R. E., et al. (1996). Experiments with a new boosting algorithm. In *ICML* (Vol. 96, pp. 148–156).

Günlük, O., Kalagnanam, J., Menickelly, M., & Scheinberg, K. (2018). Optimal decision trees for categorical data via integer programming. *Last access date: 04-05-2020*.

James, G., Witten, D., Hastie, T., & Tibshirani, R. (2013). *An Introduction to Statistical Learning* (Vol. 112). Springer.

Kaggle. (n.d.). Retrieved 15-05-2020, from https://www.kaggle.com/datasets

Kotsiantis, S. B. (2013). Decision trees: a recent overview. *Artificial Intelligence Review*, *39*(4), 261–283.

Laurent, H., & Rivest, R. L. (1976). Constructing optimal binary decision trees is np-complete. *Information Processing Letters*, *5*(1), 15–17.

Norouzi, M., Collins, M., Johnson, M. A., Fleet, D. J., & Kohli, P. (2015). Efficient non-greedy optimization of decision trees. In *Advances in Neural Information Processing Systems* (pp. 1729–1737).

Quinlan, J. R. (1993). *C4. 5: Programs for Machine Learning*. Morgan Kaufmann Publishers.

Quinlan, J. R. (1996). Bagging, Boosting, and C4.5. In *In Proceedings of the Thirteenth National Conference on Artificial Intelligence* (pp. 725–730). AAAI Press.

UCI Machine Learning Repository. (n.d.). Retrieved 15-05-2020, from https://archive.ics.uci.edu/ml/index.php

Wolsey, L. A. (1998). *Integer Programming* (Vol. 52). John Wiley & Sons.

A MILP Formulations

Basic Formulation

Basic integer programming formulation from Section 4.2:

$$
\begin{aligned}
max \quad & \sum_{i \in I_+} \sum_{b \in B_+} c_b^i + C \sum_{i \in I_-} \sum_{b \in B_-} c_b^i \\
s.t. \quad & \sum_{g \in G} v_g^k && = 1 && \forall k \in K\,, \\
& z_j^k && \leq v_{g(j)}^k && \forall j \in J\,, \forall k \in K\,, \\
& c_b^i && \leq L(i,k) && \forall i \in I\,, \forall k \in K\,, \forall k \in K^L(b)\,, \\
& c_b^i && \leq R(i,k) && \forall i \in I\,, \forall k \in K\,, \forall k \in K^R(b)\,, \\
& \sum_{b \in B} c_b^i && = 1 && \forall i \in I\,, \\
& v_g^k \in && \{0,1\} && \forall g \in G\,, \forall k \in K\,, \\
& z_j^k \in && \{0,1\} && \forall j \in J\,, \forall k \in K\,, \\
& c_b^i \in && \{0,1\} && \forall b \in B\,, \forall i \in I\,.
\end{aligned}
$$

Improved Formulation

Mixed integer programming formulation with improvements from Section 4.3.1, 4.3.2 and 4.3.3:

$$
\begin{aligned}
max \quad & \sum_{i \in I_+} \sum_{b \in B_+} c_b^i + C \sum_{i \in I_-} \sum_{b \in B_-} c_b^i \\
s.t. \quad & \sum_{g \in G} v_g^k && = 1 && \forall k \in K\,, \\
& z_j^k && \leq v_{g(j)}^k && \forall j \in J\,, \forall k \in K\,, \\
& z_{j(g)}^k && = v_g^k && \forall g \in G\,, \forall k \in K^*\,, \\
& \sum_{b \in B, k \in K^L(b)} c_b^i && \leq L(i,k) && \forall i \in I\,, \forall k \in K\,, \\
& \sum_{b \in B, k \in K^R(b)} c_b^i && \leq R(i,k) && \forall i \in I\,, \forall k \in K\,, \\
& 0 \leq v_g^k \leq 1 && && \forall g \in G\,, \forall k \in K\,, \\
& z_j^k \in \{0,1\} && && \forall j \in J\,, \forall k \in K \backslash K^L\,, \\
& 0 \leq z_j^k \leq 1 && && \forall j \in J\,, \forall k \in K^L\,, \\
& 0 \leq c_b^i \leq 1 && && \forall b \in B\,, \forall i \in I\,.
\end{aligned}
$$