ERASMUS UNIVERSITY ROTTERDAM

Erasmus School of Economics

Bachelor Thesis Econometrics & Operations Research

Optimal classification trees and the influence of time limits on the accuracy of optimal classification trees

Name student: Anna van Leeuwen

Student ID number: 544960

Supervisor: R.M. Badenbroek

Second assessor: T.A.B. Dollevoet

Date final version: 03-07-2022

Abstract

To create a decision tree, often heuristics are used that use a top-down approach where the choices of splits are made separately without looking at the impact on the rest of the tree. This results in decision trees that are not optimal, because they cannot account for all the underlying characteristics of the dataset. A solution to this problem could be to use an optimal decision tree that creates the entire tree at once to reach optimality. Decision trees are widely used to create classification models, because of their easy interpretability and visualisation. The increase of computational power of solvers in mixed-integer optimization (MIO) combined with algorithmic improvements in integer optimization are the starting point of *optimal classification trees.* We present an MIO formulation where the hierarchical structure of a decision tree is created using mixed-integer constraints. By using real-world datasets of the UCI machine learning repository, we show that optimal classification trees can perform better than state-of-the-art heuristics. Because the drawback of using optimal classification trees over the top-down heuristics is time, we investigated what influence a time limit has on the obtained solution. We found that setting a small time limit on the solver still gives solutions that approach the optimal solution, such that we can obtain fast solutions that are better than those of the heuristics.

Contents

1 Introduction **3**

2 Literature review **4**

3 Methodology **6**

3.1 CART 7

3.2 MIO formulation 8

3.3 Solving MIO problem using warm starts 12

3.4 Tuning the hyper-parameters 13

3.5 OCT model with time limits 15

4 Results **15**

4.1 Data 15

4.2 Out-of-sample accuracies of CART and OCT 16

4.3 OCT model with time limits 19

5 Conclusion **21**

6 Appendix **23**

6.1 Trade-off time and accuracy 23

6.2 Programming 24

1 Introduction

A decision tree is one of the common approaches to construct a classification model. A classification model seeks to predict the class a data point falls into. An advantage of using a decision tree for this is the easy interpretability and the visualisation of the problem. A decision tree consists of nodes in a hierarchical structure, where the nodes apply a split to determine to which side of the tree a point continues, till the points reaches one of the nodes at the bottom of the tree. These nodes are all assigned to a specific class, and this class will be the class prediction of the points that fall into this node.

For applications of decision trees where a simple and precise interpretation is necessary, it is useful to have trees that are as shallow as possible. These trees are smaller and therefore easier to understand and interpret. This research is thus not only scientifically relevant, but also interesting for people who use decision trees in their work, such as doctors, engineers and lawyers. The easier the tree, the more useful it is. This is of course only the case when the tree still contains enough information such that a good classification of a point can be made.

To create a decision tree, it is possible to use a top-down approach, where splitting rules are made sequentially, starting at the root of the tree. CART (Classification and Regression Trees) (Breiman et al., 1984) can be used to construct a tree this way. However, this method will not always lead to the optimal tree, since the impact of a choice at the top of the tree on the rest of the tree is not taken into account. Therefore, we want to compare the performance of CART to a decision tree that is created in one step, resulting in an optimal decision tree. These *optimal classification trees* (OCT) can take more time to create, but will lead to higher out-of-sample accuracy compared to a top-down method such as CART. In this research, it is investigated how well the OCT model performs in comparison to the CART model, as in Bertsimas & Dunn (2017). Because time can be a disadvantage of using OCT over CART, we also investigate what influence smaller time limits have on the performance of trees created by an OCT model. When the accuracy of these trees with a time limit is still better than that of CART, the possible disadvantage of time of using OCT reduces.

In order to compare the performance of CART and OCT, we use different real-world datasets from the UCI machine learning repository that all contain information about a certain number of features and a classification. The most important features are used to construct a decision tree that predicts the class best. In both models, the goal is to minimize the misclassification error to obtain the most accurate decision tree. For the OCT model, we construct an MIO problem, where the hierarchical structure of a tree is created using constraints. To solve this model to optimality, we have to tune some hyper-parameters and therefore we use an adapted version of the OCT as intermediate step before obtaining the final tree. After the optimal tree is found for every dataset, we look at the trade-off between the time limit and the out-of-sample accuracy. Small steps in increasing the time limit are taken, and after the time limit has reached, the best obtained solution so far is tested.

We found that for a group of datasets, OCT does perform better than CART, as we expected. For these datasets, the average improvement of out-of-sample accuracy of OCT over CART is around 1-2% for decision trees with depth $D = 2$. For the trade-off between the time limit and the accuracy of the OCT model, we found that for small trees with depth $D = 1$, the optimal solution was already found after 25% of the maximum running time for one third of the datasets. For models with depth $D = 2$, the optimal value was reached within 50% of the maximum running time for 80% of the datasets. Even though we cannot know if the optimal tree is found if we do not look further than the solution obtained at a small time limit, we did find that this optimal value is often reached in a short time span. The drawback of using an MIO formulation over a state-of-the-art heuristic is time, but this is reduced now that we know that we can use shorter time limits to obtain a solution that is already better than that of the heuristic.

In the remaining part of the research, the sections are structured as follows. In Section 2 we present the theoretical background of decision trees and the methods to solve these. Then, in Section 3, the detailed methodology of the problem is described. Furthermore, it is specified how the model is solved my an MIO solver, and how the hyper-parameters are tuned. In Section 4, the data used for the experiments is described and the results are presented, followed by a conclusion in Section 5.

2 Literature review

In this section, a background of the use of decision trees is given. Furthermore, the top-down approach of making a decision tree is explained in detail and the idea of coming up with an optimal decision tree is shown. Also, different MIO solvers and their development over the years is presented. Lastly, some investigations published after (Bertsimas & Dunn, 2017) are shortly described.

Decision trees are among the most popular techniques to build classification models (Kotsiantis, 2013). These trees are subsequent models, which combine the outcome of different simple tests to obtain the classification of a point. Decision trees are clear and easy to interpret. Without having complete knowledge of a certain topic, it is still possible to follow the logical rules of a tree to end at a leaf node with a certain label. Decision trees do not always have the highest accuracy, but the easy interpretation of the trees can give a higher preference for decision trees in many applications, such as health care.

The most common method for decision tree methods in classification is CART (Breiman et al., 1984). Here, a top-down approach is used to determine the partitions at the nodes. One of the main decisions here is the choice of a splitting rule. While starting at the root node, a split is chosen based on the outcome of an optimization problem. This is done before continuing to do this repeatedly on the two child nodes that are created. This top-down approach is not only used for CART models, but also for C4.5 (Quinlan, 1993) and ID3 (Quinlan, 1986). In such a top-down approach, each split in the tree is determined without taking into consideration the possible shock this can lead to in further splits of the tree. This can result in trees that are not too accurate when it is used for classifying points in the future, because a strong split can be dominated by a weak split, by means of the sequential choices for splits. Furthermore, methods that use a top-down approach usually require pruning. This is needed because a top-down approach cannot handle penalties on the complexity of the tree while the tree is growing. Therefore, pruning needs to be done afterwards to avoid overfitting. If the penalty for complexity is too high, it can be that the most accurate tree is not found by the top-down approach. Heuristics like IDX (Norton, 1989), LSID3 and ID3-k (Esmeir & Markovitch, 2007) aim to solve the problem of strong splits being dominated by weaker splits by looking further than the current leaf when determining the splits. In these methods, the decision of splits is made when looking ahead at deeper trees starting at the current node. Nonetheless, it is unclear whether these heuristics avoid the look-ahead pathology of decision tree learning (Murthy & Salzberg, 1995a).

A more accurate way of forming a decision tree is to create the entire decision tree in one single step. This way, each split can be determined with full knowledge of all the other splits in the tree. We can then omit the situation where trees do not capture well the underlying characteristics of the dataset, but this happens at the cost of a longer running time. The resulting decision tree will be the optimal one according to the training data used. This idea was already suggested by Breiman et al. (1984). The top-down approach with pruning in CART was not chosen because they believed this was the best technique, but because of the practical drawback of time. Constructing an optimal binary decision tree is namely NP-hard (Hyafil & Rivest, 1976), so a heuristic was needed to solve these problems faster for large instances. Constructing an optimal tree can thus increase the accuracy of the tree, but it usually takes more time to construct a tree, so a trade-off between accuracy and time must be made when deciding on which method to choose.

In the past, much research has been done to find heuristics that create a decision tree in one step, for example with linear optimization (Bennett, 1992), dynamic programming (Cox et al., 1989; Payne & Meisel, 1977) and continuous optimization (Bennett & Blue, 1996). The goal of these heuristics was to create optimal univariate decision trees. However, all these methods were unable to generate optimal decision trees within a practical time period. Building a decision tree contains several discrete decisions, like splitting a node in the tree and which variable to split on. Furthermore, there are also discrete outcomes, like which leaf node a points falls into and if the point is accurately labeled. Such a problem can be formulated as an MIO problem. Since the past 40 years, many statistical problems use MIO formulations (Arthanari & Dodge, 1981). However, within the machine learning community is it believed that MIO formulations are unmanageable even for small problems. This was indeed the case in the early 1970s, when MIO solvers were not as developed as now. The increase of computational power of MIO solvers in the past decades has ensured that problems with MIO formulations can now be solved within a reasonable time period.

CPLEX, GLPK, MOSEK and GUROBI are some examples of currently developed MIO solvers. In Bixby (2012), the speedup of MIO solvers was measured by testing twelve consecutive versions of CPLEX on the same set of MIO problems. The difference in speed between the versions was measured. The versions of CPLEX that were tested were those between the first in 1991 and the one in 2007, named CPLEX 1.2 and CPLEX 11, respectively. Between all these versions, a newer version resulted in an improvement compared to the previous ones. Between the first and last version, a total speedup factor over 29,000 was found (Bixby, 2012; Nemhauser, 2013). As a result of the improvement of the MIO solvers, the use of MIO formulations can be more useful than before. The theoretical qualities of the MIO formulations are now supported by the practical benefits that the problems can be solved more quickly.

Another important benefit of using MIO is the completeness obtained by the modeling framework. MIO can contain univariate and multivariate decision trees, where the multivariate splits are often much stronger than the univariate ones. A multivariate decision tree is a tree where splits can be made with multiple variables at a time, while a univariate decision tree only splits on one variable at the same time.

After the research on optimal classification trees by Bertsimas & Dunn (2017), new researches have been published about optimizing decision trees. For example, there is an approach that learns optimal tree-based prescription policies directly from the data (Amram et al., 2022). This method can produce optimal, interpretable policy trees that can handle both discrete and continuous treatments. After some experiments on data, it is shown that these trees perform best-in-class. Furthermore, survival tree methods adapt the tree-based models such that censored outcomes can be analysed (Bertsimas et al., 2022). This is useful for medical data, where these censored outcomes often appear. This Optimal Survival Trees algorithm that builds on the MIO problem can improve on the accuracy of existing survival tree methods.

In this research, the outcomes of CART and OCT are compared. In both models, the goal is to minimize the misclassification error to obtain the most accurate decision tree. When we want to create a decision tree in one step as in OCT, we need to formulate an MIO problem that can be solved quickly. In the next section, an MIO formulation is used to solve the decision tree problem. Moreover, evidence of the success of this approach is given. First, we present the exact formulation of this new classification method, *optimal classification trees* (OCT). We then compare the outcomes of OCT to the state-of-the-art CART method on a sample of datasets from the UCI machine learning repository. The out-of-sample accuracies of the methods are compared. Then, we look at the trade-off between the time limit and the best obtained solution after that time limit is reached.

3 Methodology

In this section, the optimal decision tree formulated as an MIO problem will be introduced. First, the general processes of CART and OCT are given, and afterwards the exact variables and constraints are shown. Then we give a detailed algorithm to implement this OCT MIO problem.

Each dataset contains n observations $(\boldsymbol{x}_i, y_i)$, $i = 1, ..., n$, each with p features $\boldsymbol{x}_i \in \mathbb{R}^p$ and a label

$y_i \in \{1, ..., K\}$ that indicates which of the K labels corresponds to this data point. All values of each dimension in a dataset are normalized to the 0-1 interval, such that $\boldsymbol{x}_i \in [0,1]^p$. In the optimal tree that is found in the end, we make a distinction between two kind of nodes:

- Branch nodes: nodes that apply a split with parameters a and b. These parameters are used to determine which path of the decision tree a point will follow. For a given point i, if $\boldsymbol{a}^T\boldsymbol{x}_i < b$, the point will go on to the left branch from the node, and otherwise it goes to the right branch. In this paper, we look at univariate decision trees, where a split is only made for one variable at a time.

- Leaf nodes: nodes at the bottom of the tree. These nodes correspond to a class where the point is classified when it ends at this leaf node. The class is chosen as the most common class upon all the points in the leaf node.

3.1 CART

For decision tree methods with a top-down approach like CART, ID3 and C4.5, the partitioning process goes as follows. At each step of the process, they try to find a split that separates the current region such that the splitting criterion is maximized. This criterion is often based on the label impurity of the dataset that are in the resulting regions. The algorithm repeats this partition process at the two new regions that are made by the previous split. Once a stopping criterion is met, the partitioning process stops. The stopping criteria for CART are the following (Bertsimas & Dunn, 2017):

- It is not possible to create a split where each side of the partition has at least a certain number of nodes, N_{min}.

- All points in the current node are part of the same class.

When the splitting procedure is finished, each region is linked to one of the K class labels. This class is then used to predict the class of the points that are in the region.

In order to avoid overfitting, the tree needs to be pruned. This process starts at the bottom of the tree, and then works upwards to the root node. There is a complexity parameter, α, that decides whether to prune a node or not. This parameter looks at the trade-off between the additional accuracy of a split and the additional complexity of adding the split. The higher this complexity parameter, the more nodes are pruned off and this gives thus smaller trees. This procedure of growing and pruning in two-stages is used in the CART problem. The problem can be formulated as a formal optimization problem. In total, CART contains two parameters: the complexity parameter, α, and the minimum number of points required in each leaf node, N_{min}. These parameters together with the training data containing n observations $(\boldsymbol{x}_i, y_i)$, $i = 1, ..., n$ can be used to make a tree T that solves the following problem:

$$\begin{aligned} \min \ & M_{xy}(T) + \alpha|T| \\ \text{s.t.} \ & N_x(l) \geq N_{min}, \qquad \forall l \in \text{leaves}(T), \end{aligned} \tag{1}$$

where $M_{xy}(T)$ is the misclassification of the tree T on the training data, $|T|$ is the number of branch nodes in the tree T, and $N_x(l)$ is the number of points assigned to leaf node l. This is the *optimal tree problem*, and CART is used to solve this problem. As mentioned in Bertsimas & Dunn (2017), the choice for CART is arbitrary, and other methods like C4.5 and ID3 could have been used instead. These are all methods that seek to find a tree T based on heuristics, while we try to find an optimal tree T. Experiments of Murthy & Salzberg (1995b) found that the difference in out-of-sample accuracy between CART and C4.5 is not significant, so the choice between these methods is not really important for the purpose of this research. For testing the performance of CART on real-world datasets, the R programming (R Core Team, 2015) language is used.

The solutions of CART are compared to those of OCT. In OCT, a decision tree is made in one single step, such that a optimal classification tree is created. To find such a globally optimal decision tree, an MIO formulation is used. We will start with the classic OCT model, as introduced in Bertsimas & Dunn (2017). For the OCT model, we keep track of the split applied at every node. In this univariate case, the split at each node only involves a single variable. While creating a tree, we need to make a few decisions at every step. At every node, we need to decide whether to branch or stop. If we choose to stop branching at the node, we must choose the label assigned to the created leaf node. If we choose to branch at the node, we must choose the variable to branch on. Since we look at the univariate case, this is only one variable. When classifying the training points to the leaf nodes of the tree, we must decide which leaf node the training point ends in. The advantage of using MIO instead of a top-down method is that we can make all these decisions at the same time, instead of considering them separately. This will result in an optimal tree, because the impact of a decision at the top of the tree is also measured for the rest of the tree.

3.2 MIO formulation

A formulation for (1) as an MIO problem will now be presented. This section is based on Section 2.2 from Bertsimas & Dunn (2017). We always start with trying to construct an optimal decision tree with maximum depth D. Given this depth D, a maximal number of nodes can be reached to construct a maximal tree. This number of nodes equals $T = 2^{(D+1)} - 1$. All nodes have an index $t = 0, ..., T - 1$, where the node with index $t = 0$ refers to the root node. $p(t)$ stands for the parent of node t, and $A(t)$ is the set with all ancestors of node t. The latter is split up in two parts: $A_L(t)$ and $A_R(t)$, and this corresponds to the ancestors whose left branch has been followed on the path from the root node to node t, and the ancestors whose right branch has been followed on the path from the root node to node t, respectively. Next to this, there is also set n, including all observations of the dataset, indicated by $i = 1, ..., n$. Each data point must be classified in one of the K classes, indicated by $k \in \{1, ..., K\}$ classes. Per dataset, there are p different features on which splits can be made, indicated by $j \in \{1, ..., p\}$.

All the sets used in the model can be summarized as follows:

$$\begin{aligned}
&D : \text{maximum depth of the tree}\\
&T : \text{maximum number of nodes in the tree with depth } D\\
&p(t) : \text{parent of node } t\\
&A_L(t) : \text{left ancestors of node } t\\
&A_R(t) : \text{right ancestors of node } t\\
&n : \text{number of observations}\\
&K : \text{number of classes}\\
&p : \text{number of features}
\end{aligned}$$

The indices of all nodes T are split up between the branch and leaf nodes:

- Branch nodes: nodes $t \in T_B = \{1, ..., \lfloor T/2 \rfloor\}$ apply a split $\boldsymbol{a}_t^T \boldsymbol{x}_i < b_t$ for a certain point $i = 1, ..., n$. Nodes that satisfy this condition follow the left branch of the tree, and otherwise the right branch of the tree is followed.
- Leaf nodes: nodes $t \in T_L = \{\lfloor T/2 \rfloor + 1, ..., T\}$ make a class prediction of the points that end at the leaf node.

For the different branch and leaf nodes, constraints are needed to make sure the tree grows according the right structure. The first decision variables that are introduced for this are the following:

$$\begin{aligned}
&\boldsymbol{a}_t \in \mathbb{R}^p, \qquad t \in T_B,\\
&b_t \in \mathbb{R}, \qquad t \in T_B,\\
&d_t = \mathbb{1}\{\text{node } t \text{ applies a split}\}, \qquad t \in T_B,
\end{aligned}$$

where $\boldsymbol{a}_t$ and b_t are used to track the split applied at branch node $t \in T_B$. The d_t variables are used to keep track of which branch nodes apply splits. If a branch node does not apply a split, we model this by setting $\boldsymbol{a}_t = \boldsymbol{0}$ and $b_t = 0$. This way, the split constraint becomes $0 < 0$, which makes sure the point will follow the right branch, since the constraint is never satisfied. This is implemented to ensure that the tree does not stop growing when a branch node does not apply a split. Instead, all nodes are sent to the same leaf node. The following constraints are used to enforce this:

$$\sum_{j=1}^{p} a_{jt} = d_t, \qquad \forall t \in T_B, \tag{2}$$

$$0 \leq b_t \leq d_t, \qquad \forall t \in T_B, \tag{3}$$

$$a_{jt} \in \{0, 1\}, \qquad j = 1, ..., p, \quad \forall t \in T_B, \tag{4}$$

where constraints (2) make sure only one split is applied for each branch node $t \in T_B$. Constraints (3) are valid for b_t, because b_t only becomes greater than zero if node t applies a split, and thus if $d_t = 1$. Since we normalized the data points, meaning all $\boldsymbol{x}_i \in [0,1]^p$, and since $\boldsymbol{a}_t$ contains only one 1 if $d_t = 1$, it is always true that $0 \leq \boldsymbol{a}_t^T \boldsymbol{x}_i \leq d_t$ for any i and t. We therefore only have to consider values between 0 and 1 for b_t.

To keep the hierarchical structure of the tree, we need constraints that ensure that a branch node cannot apply a split if its parent does not also apply a split.

$$d_t \leq d_{p(t)}, \qquad \forall t \in T_B \backslash \{0\}. \tag{5}$$

This constraint only holds for branch nodes that are not the root node.

Constraints 2 - 5 construct the tree structure while using MIO. Now we continue with constraints to keep track of the allocation of points to the leaf nodes. First, we need new indicator variables:

$$\begin{aligned} z_{it} &= \mathbb{1}\{\boldsymbol{x}_i \text{ is in node } t\}, \\ l_t &= \mathbb{1}\{\text{leaf node } t \text{ contains any points}\}. \end{aligned}$$

We enforce all leaves, if they contain points, to have a minimum number of points, N_{min}:

$$z_{it} \leq l_t, \qquad i = 1, ..., n, \quad t \in T_L, \tag{6}$$

$$\sum_{i=1}^{n} z_{it} \geq N_{min} l_t, \qquad t \in T_L. \tag{7}$$

Constraints (6) differ from Bertsimas & Dunn (2017), because they hold for every $i = 1, ..., n$ and for $t \in T_L$ instead of $t \in T_B$. The latter holds for (7) too.

Each point needs to be assigned to one leaf node, and this is enforced in the following constraints:

$$\sum_{t \in T_L} z_{it} = 1, \qquad i = 1, ..., n. \tag{8}$$

We also need constraints to make sure that a leaf that is reached by following the left branch of the parent (so the left child) can only contain points if a split is made by their parent. If no split is made, all points need to be assigned to the right child:

$$l_t \leq d_m, \qquad t \in T_L, \quad \forall m \in A_L(t). \tag{9}$$

These constraints are not mentioned in Bertsimas & Dunn (2017), but they are necessary to avoid that the tree stops growing early because no split is made. This is now solved, because there are still child nodes, even if the parent does not apply a split. If this is the case, we saw that we set $\boldsymbol{a}_t = \boldsymbol{0}$ and $b_t = 0$ and the point will then follow the right branch.

Finally, we need constraints to enforce the splits that are required by the hierarchical structure of the tree, when points are assigned to leaves:

$$\boldsymbol{a}_m^T \boldsymbol{x}_i < b_m + M_1(1 - z_{it}), \qquad i = 1, ..., n, \quad \forall t \in T_L, \quad \forall m \in A_L(t), \tag{10}$$

$$\boldsymbol{a}_m^T \boldsymbol{x}_i \geq b_m - M_2(1 - z_{it}), \qquad i = 1, ..., n, \quad \forall t \in T_L, \quad \forall m \in A_R(t). \tag{11}$$

These constraints (10) and (11) need to be satisfied for all leaf nodes, contrary to Bertsimas & Dunn (2017), where it is said that the branch nodes need to satisfy these constraints. Furthermore, vector $\boldsymbol{a}_m$ and b_m both use the same index m, while this is different in Bertsimas & Dunn (2017). Since we can only have non-strict inequalities in a problem solved by an MIO solver, we need to convert (10) into a constraint that is not a strict inequality. For this reason, we add a small constant ϵ to the left side of equation (10):

$$\boldsymbol{a}_m^T \boldsymbol{x}_i + \epsilon \leq b_m + M_1(1 - z_{it}), \qquad i = 1, ..., n, \quad \forall t \in T_L, \quad \forall m \in A_L(t). \tag{12}$$

The value of ϵ cannot be too small, since this could cause difficulties for the MIO solver, so we want to make ϵ as big as possible without affecting the feasibility of any solution to the problem. This is done by defining a different ϵ_j for each feature j. The biggest value possible is the smallest non-zero distance between adjacent values of this feature. To find this value, we sort the values of the jth feature in non-increasing order. Then, for each feature j, the value for ϵ is calculated as follows:

$$\epsilon_j = \min \left\{ x_j^{(i+1)} - x_j^{(i)} \;\middle|\; x_j^{(i+1)} \neq x_j^{(i)}, \quad i = 1, ..., n-1 \right\},$$

where $x_j^{(i)}$ is the ith largest value in the jth feature. These values of ϵ_j are combined in vector $\boldsymbol{\epsilon}$ and can be implemented in the constraints:

$$\boldsymbol{a}_m^T (\boldsymbol{x}_i + \boldsymbol{\epsilon}) \leq b_m + M_1(1 - z_{it}), \qquad i = 1, ..., n, \quad \forall t \in T_L, \quad \forall m \in A_L(t). \tag{13}$$

Then we must specify values for the constants M_1 and M_2. We know that both $\boldsymbol{a}_m^T \boldsymbol{x}_i \in [0, 1]$ and $b_t \in [0, 1]$, such that the largest possible value of $\boldsymbol{a}_t^T (\boldsymbol{x}_i + \epsilon) - b_m$ is $1 + \epsilon_{max}$, where $\epsilon_{max} = \max_j \{\epsilon_j\}$. Therefore, $M_1 = 1 + \epsilon_{max}$. Similarly, the largest possible value of $b_t - \boldsymbol{a}_t^T \boldsymbol{x}_i$ is 1, so we can set $M_2 = 1$. This results in the following definitive constraints that enforce splits in the tree:

$$\boldsymbol{a}_m^T (\boldsymbol{x}_i + \boldsymbol{\epsilon}) \leq b_m + (1 + \epsilon_{max})(1 - z_{it}), \qquad i = 1, ..., n, \quad \forall t \in T_L, \quad \forall m \in A_L(t), \tag{14}$$

$$\boldsymbol{a}_m^T \boldsymbol{x}_i \geq b_m - (1 - z_{it}), \qquad i = 1, ..., n, \quad \forall t \in T_L, \quad \forall m \in A_R(t). \tag{15}$$

The objective is to minimize the misclassification error, so we need some parameters and variables to keep track of the misclassification:

$$\begin{aligned}
&N_{kt} : \text{number of points of label } k \text{ in node } t,\\
&N_t : \text{number of points in node } t,\\
&c_{kt} = \mathbb{1}\{c_t = k\},\\
&L_t : \text{optimal misclassification loss in node } t,
\end{aligned}$$

where c_t corresponds to the label of leaf node t. N_{kt} and N_t can be calculated using the z_{it} variables:

$$N_{kt} = \sum_{i:y_i=k} z_{it}, \qquad \forall k \in K, \quad \forall t \in T_L \tag{16}$$

$$N_t = \sum_{i=1}^{n} z_{it}, \qquad \forall t \in T_L. \tag{17}$$

The label of each leaf node t is determined as the most common label of all points assigned to node t:

$$c_t = \arg \max_{k=1,\dots,K} \{N_{kt}\}. \tag{18}$$

Only one class prediction needs to be made for the nodes that contain points:

$$\sum_{k=1}^{K} c_{kt} = l_t, \qquad \forall t \in T_L. \tag{19}$$

The misclassification at each leaf node, L_t, is calculated as the total number of points in the node less the number of points of the most common label:

$$L_t = N_t - \max_{k=1,\dots,K} \{N_{kt}\} = \min_{k=1,\dots,K} \{N_t - N_{kt}\}, \tag{20}$$

which can be linearized to the following constraints:

$$L_t \geq N_t - N_{kt} - M(1 - c_{kt}), \qquad k = 1, ..., K, \quad \forall t \in T_L, \tag{21}$$

$$L_t \leq N_t - N_{kt} + Mc_{kt}, \qquad k = 1, ..., K, \quad \forall t \in T_L, \tag{22}$$

$$L_t \geq 0, \qquad \forall t \in T_L, \tag{23}$$

where M is a sufficiently large constant that makes the constraint inactive depending on the value of c_{kt}. We take $M = n$ in this case.

The total misclassification cost is calculated as $\sum_{t \in T_L} L_t$, and the complexity of the tree is equal to the number of splits, which is calculated as $\sum_{t \in T_B} d_t$. Like in CART, we normalize the misclassification against the baseline accuracy $\hat{L}$. This is obtained by predicting the most occurring class in the entire dataset. For example, if we have a dataset with $n = 100$, and the most popular class occurs in 60 of the 100 points, we set $\hat{L} = 60$. This way, the impact of α is independent of the size of the dataset. The objective of problem (1) can then be written as:

$$\min \frac{1}{\hat{L}} \sum_{t \in T_L} L_t + \alpha \sum_{t \in T_B} d_t. \tag{24}$$

This objective (24) combined with constraints 2 - 9, 14 - 17, 19 and 21 - 23 results in the MIO formulation of problem (1), named the OCT model.

This model can be solved by any MIO solver, and in this paper, CPLEX will be used to obtain the solutions. The three hyper-parameters, maximum depth D, minimum leaf size N_{min} and complexity parameter α, still need to be specified to solve the model. In Section 3.4, an effective method to tune these parameters is presented.

3.3 Solving MIO problem using warm starts

MIO solvers can solve faster when a feasible solution is given as warm start before the process starts to solve. As in Bertsimas & Dunn (2017), we use warm starts when running the adapted OCT model with the penalty constraint, where the variable added in this constraint is increased. The stronger the warm start, the faster the new problem can be solved. The warm start provides a strong upper bound, from which the solver can build to find a better solution. In this case, a better solution is a solution with a lower combination of misclassification error and complexity of the model. We can use solutions of CART as a warm start for the MIO model, or we can use previously obtained solutions from the OCT model as warm start. Namely, if we have a solution for depth D, this is a feasible warm start for depth $D+1$. In Section 3 of Bertsimas & Dunn (2017), it is mentioned that they start building trees from depth $D_{max} = 2$, and use warm starts then. It is thus also not exactly specified if they use the solution from $D = 1$ as warm start. Due to time limitations, the solutions of CART and of lower depths are not used as warm starts in this problem. We do use warm starts for the different problems within a certain D, as described in Section 3.4.

3.4 Tuning the hyper-parameters

In this section, the full implementation details of the algorithm to tune the hyper-parameters to create the optimal decision tree using the OCT MIO model are presented. First, we take a look into parameter D, the maximum depth of the tree. This parameter is important, because the depth of the tree is of great influence for the complexity of the tree and thus of the problem. In this research, we mainly look at trees with depth $D = 1$ and $D = 2$. When we investigate the influence of a time limit on the resulting tree, we also look at trees with depth $D = 3$ for some of the datasets.

Furthermore, we have to tune the complexity parameter α. Because α is continuous-valued, we could discretize the possible values and then test each value for the MIO problem. However, this is an inefficient approach, because some values will give the same solution and this will thus lead to unnecessary runs. We need to ensure to have a sufficiently fine-grained discretization, because otherwise good values of α are missed. On the other side, we ideally want to search only between the "critical" values α. To determine the best value of α, we rewrite the objective of the original OCT model:

$$\min \quad \frac{1}{\hat{L}} \sum_{t \in T_L} L_t + \alpha \sum_{t \in T_B} d_t, \tag{25}$$

into an objective and a penalty constraint:

$$\begin{aligned} \min \quad & \frac{1}{\hat{L}} \sum_{t \in T_L} L_t \\ \text{s.t.} \quad & \sum_{t \in T_B} d_t \leq C, \end{aligned}$$

where C is a constant corresponding to the value of α. This objective and constraint in combination with the other unchanged constraints is the adapted OCT model. In this new constraint, C is the maximum number of splits in the decision tree. The number of splits is an integer, so we only have to test integer values of C. We search through values $C = 1, ..., C_{max}$, where $C_{max} = 2^D - 1$ is the maximum number of splits, given depth D. We can use the solution of the problem with C as a warm start for the problem with $C+1$ maximum splits. When comparing all solutions for the different values of C, it can occur that one of those solutions is not optimal for any value of α with respect to objective (25). In this case, the solution is dominated by other values of C. Then we do not want to take into account this dominated solution, so we can remove such solutions in a post-processing step to only end with possible optimal solutions. Given a depth D, we have a number of optimal solutions obtained by running the model for all possible values of C. For each of these runs, a time limit of 30 minutes is applied (except for two datasets, specified in Section 4.2). We then run these solutions on a validation set and identify the best performing solution. For this solution, we can make an interval for α for which this solution is strictly better than the other solutions. The midpoint of this interval is used as optimal value for α for the original problem. In case of equal performance between solutions, the union of the intervals can be used. In the case where all values of C generate the same optimal solution, it can be argued that the chance of overfitting (for which this tuning is done) is not great, so we can set α to zero.

The other two hyper-parameters, D_{max} and N_{min}, are not tuned for this research as this is not necessary for the computational experiments in Section 4. For D_{max}, values from 1 up to 4 are used in Bertsimas & Dunn (2017). For this research, we start with running the model for $D_{max} = 2$. The minimum leaf size N_{min} is set to 5% of the number of data points. This is done to reduce the number of computations needed. We are then left with the process of determining the optimal value for α, which can be summarized as follows, as in Bertsimas & Dunn (2017):

1. Set the maximal depth D_{max} and minimum leaf size N_{min}.
2. For $D = 1, ..., D_{max}$:
   (a) For $C = 1, ..., 2^D - 1$:
      i. Search through pool of candidate warm starts and choose the one with the lowest error.
      ii. Solve the MIO problem for depth D and C using the selected warm start.
      iii. Add the MIO solution to the warm start pool.
3. Post-process the solution pool by eliminating all solutions that are not optimal to (25) for any value of α.
4. Identify the best performing solution on the validation set, and determine the interval of α for which this solution is optimal. Use the midpoint of this interval as the tuned value of α

In Bertsimas & Dunn (2017), the solution of CART with the same N_{min} and $\alpha = 0$ is also used as a warm start for each run of D and C. In this research, this step is omitted, as it is not necessary for the specific outcome, only for the running time needed to reach a solution. This results in that not every step of the process above uses a warm start: for the values of $D = 1$ and $C = 1$, there is no warm start in step 2(a)i.

After the tuning process, the obtained α can be used to train a tree with the original OCT model on the combined training and validation set. The last 25% of the dataset can be used to test the out-of-sample accuracy of the decision tree.

3.5 OCT model with time limits

For the part of testing the influence of different time limits on the accuracy of the tree, the same OCT model is used. Per dataset, different values for the time limit are tested, while we keep track of the best obtained solution at the end of such a time limit. The time limit applies only on the final determination of the optimal tree, not on the hyperparameter tuning process. The time limit for this latter process stays 30 minutes (and two hours for two datasets), as before. The total running time needed to solve the model completely (so without time limit) differs per dataset, depending on the size, number of features and number of classes. Since some datasets can solve to optimality very fast, we start with small time limits and build up to the number of seconds that is needed to solve the model to optimality. This upper bound is thus different per dataset: some datasets can already solve in 20 seconds, others take 30 minutes, so we also look at different increases in time limit, and look at the trade-off between the percentage of accuracy and the percentage of time that was used compared to the time used to completely solve. In order to have a fair comparison between the time limits, the original dataset is shuffled once for this part. This shuffled dataset is split up into a training, validation and testing set and this same partition is used for each run with a different time limit.

4 Results

In this section, the outcomes of the experiments are presented. First, the used data is described. Then, an overview of the out-of-sample accuracies of the different datasets is presented, including a comparison between CART and OCT. Lastly, an analysis of the influence of the maximum running time on the performance of the best obtained tree in that time period is given.

4.1 Data

In order to do experiments with the models for classification trees, we use real-world datasets from the UCI machine learning repository (Lichman, 2013). The same datasets will be used as in Bertsimas & Dunn (2017). There are some datasets that cannot be used, because they cannot be found or downloaded, or deviate too much from the dataset described in Bertsimas & Dunn (2017). For example, the “Cylinder bands” dataset is described in the paper with $n = 277$ and $p = 484$, while the only dataset on the website of the UCI machine learning repository contains $n = 512$ observations, with each $p = 39$ features. The outcomes of the CART and OCT model for this dataset differ much from the results in Bertsimas & Dunn (2017). Since it is not mentioned specifically how they altered the dataset, we cannot know exactly where the differences in the dataset are, but based on the values for n and p, it is already clear that the dataset is completely different. Therefore, we cannot make a good replication of this dataset, and we made the decision to not include this dataset in the experiments. Datasets “Acute-inflammations-1”, “Acute-inflammations-2” and “Climate-model-crashes” could not be downloaded, and “Connectionist-bench” and “Pima-Indians-diabetes” are not visible on the current version of the UCI machine learning repository. Due to time limitations, the following datasets are also not included in the results: “Image-segmentation”, “Iris”, “Optical-recognition”, “Parkinsons” and “Spambase”. The remainder of the datasets as in Bertsimas & Dunn (2017) are used for the replication of this part, in total 43 datasets. The datasets from the UCI machine learning repository are often used for empirical analysis of machine learning algorithms. In this research, only datasets containing up to 1000s of points are used, because larger datasets require much more time to obtain a solution with good quality.

The datasets that contain missing values are cleaned before using. We omitted all rows that contain missing values. For all datasets except “Ozone-level-detection-eight”, this resulted in a dataset with the number of observations corresponding to the n in Bertsimas & Dunn (2017), so we assumed that the data was cleaned in the same way there. There are some datasets where the number of features p does not coincide with the number indicated in Bertsimas & Dunn (2017). The number of features p in Table 1 is followed by ** if it does not correspond to the number in Bertsimas & Dunn (2017).

In the experiments, all datasets are separated into three parts. The first part contains 50% of the data points and is used as the training set. The second and third part both contain 25% and are used as the validation and testing set, respectively. In Section 3.4 it is explained in more detail how these splitted sets are used. To make sure that the specific partition of the dataset into a training, validation and testing set does not influence the outcome too much, the splitting of each dataset is done five times. In all these five different partitions, results on the out-of-sample accuracy are obtained and these are averaged to get the final out-of-sample accuracies.

4.2 Out-of-sample accuracies of CART and OCT

In Table 1, the results for CART and OCT at depth $D = 2$ are presented. In the second to fourth column, the size of the dataset, the number of features, and the number of classes are shown. The column “Missing values?” indicates if there are missing values in the dataset, and if the value is “yes”, this means that the dataset is altered by removing all rows with missing values. The size n of the dataset is the size after cleaning the dataset. In the next two columns, the out-of-sample accuracies of CART and OCT are given, with the best performing one of the two in bold and in brackets the obtained accuracy as in Bertsimas & Dunn (2017). An asterisk next to the accuracy means the model could not solve to optimality within the time limit of two hours. The last column shows the mean improvement of OCT over CART, with a standard error of the measures.

Table 1: Results for CART and OCT at depth $D = 2$

Dataset					Mean out-of-sample accuracy		Mean improvement
Name	n	p	K	Missing values?	CART	OCT	
Balance-scale	625	4	3	no	64.8 (64.5)	**69.0** (67.1)	+4.13 ± 2.02
Banknote-authentication	1372	4	2	no	**89.2** (89.0)	52.4 (90.1)	-36.79 ± 1.24
Blood-transfusion	748	4	2	no	75.8 (75.5)	**75.9** (75.5)	+0.64 ± 1.53
Breast-cancer-diagnostic	569	30	2	no	**90.6** (90.5)	62.8 (91.9)	-27.81 ± 1.35
Breast-cancer-prognostic	194**	32**	2	yes	75.1 (75.5)	**78.3** (75.1)	+3.23 ± 2.67
Breast-cancer	683**	9	2	yes	92.4 (92.3)	**94.8** (94.5)	+2.43 ± 0.63
Car-evaluation	1728	6**	4	no	**77.1** (73.7)	76.1 (73.7)	-1.06 ± 2.26
Chess-king-rook-versus-king-pawn	3196	36**	2	no	**77.7** (78.2)	69.9* (86.7)	-7.81 ± 2.60
Congressional-voting-records	232	16	2	yes	**97.9** (98.6)	96.9 (98.6)	-1.03 ± 0.91
Connectionist-bench-sonar	208	60	2	no	**70.4** (70.4)	51.5* (71.2)	-18.85 ± 2.92
Contraceptive-method-choice	1473	9**	3	no	47.0 (46.8)	**47.6** (48.4)	+0.56 ± 1.73
Credit-approval	653	15**	2	yes	**87.4** (87.7)	84.9 (87.7)	-2.53 ± 1.33
Dermatology	358	34	6	yes	65.8 (65.4)	**69.2** (67.4)	+3.44 ± 2.74
Echocardiogram	61	8**	2	yes	**97.5** (74.7)	71.0 (77.3)	-26.5 ± 12.45
Fertility	100	9**	2	no	88.0 (88.0)	**92.0** (85.6)	+4.0 ± 4.0
Haberman-survival	306	3	2	no	73.0 (73.2)	**75.5** (73.2)	+ 2.54 ± 1.85
Hayes-roth	132	3**	3	no	45.5 (52.7)	**47.2** (45.5)	+ 1.82 ± 2.97
Heart-disease-Cleveland	297	13**	5	yes	**54.5** (54.1)	53.8 (53.6)	-0.69 ± 3.32
Hepatitis	80	19	2	yes	**82.0** (83.0)	**82.0** (83.0)	+0.00 ±0.00
Indian-liver-patient	579	10	2	no	71.6 (71.7)	**71.8** (70.9)	+0.16 ± 2.89
Ionosphere	351	34	2	no	**88.7** (87.8)	68.7 (87.8)	- 19.93 ± 2.01
Mammographic mass	830	4**	2	yes	77.8 (81.2)	**79.5** (81.2)	+1.73 ± 0.90
Monks-problems-1	124	6**	2	no	69.7 (57.4)	**84.5** (67.7)	+14.84 ± 4.78
Monks-problems-2	169	6**	2	no	60.9 (60.9)	**67.6** (60.0)	+6.69 ± 4.28
Monks-problems-3	122	6**	2	no	94.4 (94.2)	**95.3** (94.2)	+0.94 ± 2.27
Ozone-level-detection-eight	1846**	72	2	yes	93.4 (93.1)	**93.8** (93.1)	+0.33 ± 0.71
Ozone-level-detection-one	1848	72	2	yes	96.8 (96.8)	**97.1** (96.8)	+0.39 ± 0.38
Planning-relax	182	12	2	no	71.3 (71.1)	**72.0** (71.1)	+0.70 ± 4.00
Qsar-biodegradation	1055	41	2	no	**76.7** (76.4)	70.8 (76.1)	-5.87 ± 2.72
Seeds	210	7	3	no	87.0 (87.2)	**88.1** (88.7)	+1.03 ± 2.80
Seismic-bumps	2584	18**	2	no	93.5 (93.3)	**93.7** (93.3)	+0.19 ± 0.36
Soybean-small	47	35**	2	no	72.7 (72.7)	**98.3** (98.2)	+25.60 ± 1.67
Spect-heart	80	22	2	no	64.0 (64.0)	**72.0** (65.0)	+8.00 ± 4.74
Spectf-heart	80	44	2	no	**68.0** (69.0)	64.0 (72.0)	- 4.00 ± 2.74
Statlog-project-German-credit	1000	20**	2	no	69.1 (70.1)	**71.0** (70.5)	+1.84 ± 1.22
Statlog-project-landsat-satellite	4435	36	6	no	**63.1** (63.2)	42.4 (63.2)	- 20.72 ± 0.69
Teaching-assistant-evaluation	151	5**	3	no	40.5 (38.9)	**49.2** (42.2)	+8.66 ± 2.78
Thoracic-surgery	470	16**	2	no	85.9 (85.5)	**86.3** (84.6)	+0.39 ± 1.95
Thyroid-disease-ann-thyroid	3772	21	3	no	**95.2** (95.6)	92.4 (95.6)	- 2.74 ± 1.30
Thyroid-disease-new-thyroid	215	5	3	no	**91.5** (91.3)	71.3 (92.8)	-20.16 ± 3.55
Tic-tac-toe-endgame	958	9**	2	no	69.0 (68.5)	**69.9** (69.9)	+0.87 ± 2.39
Wall-following-robot-2	5456	2	4	no	**93.8** (94.0)	40.3 (94.0)	- 53.46 ± 0.55
Wine	178	13	2	no	**81.8** (81.3)	53.6 (91.6)	- 28.14 ± 9.78

* = did not solve to optimality within the time limit of two hours

** = different from Bertsimas & Dunn (2017)

Looking at the results for CART and OCT, there are some remarkable outcomes in Table 1. For some datasets, like "Banknote-authentication", "Breast-cancer-diagnostic" and "Wall-following-robot-2", the out-of-sample accuracies for OCT are significantly lower (more than 25%) than those of CART. We would expect the accuracy of OCT to be at least as high as that of CART, and it can happen that OCT performs a little less due to the samples that are taken, but the big differences for these datasets are inexplicable. The outcomes for CART are similar to the outcomes of CART in Bertsimas & Dunn (2017), so it is unlikely that it has something to do with the dataset itself. The high accuracies for CART make the low accuracies for OCT more unlikely, but at this point there is not an explanation for the uncommon values for OCT.

For two datasets, the OCT model could not solve to optimality within two hours. Therefore, the final out-of-sample accuracy for OCT is worse than that of CART. For "Chess-king-rook-versus-king-pawn", the accuracy for OCT was much higher than for CART in Bertsimas & Dunn (2017), and the gap left after two hours of solving indicates that the solver could find a significantly better solution. The difference between the accuracy for CART and OCT is even bigger for "Connectionist-bench-sonar", where the out-of-sample accuracy for OCT is almost 20% lower. The optimality gap at the end of the two hours was over 70%, so there is much space left to find a better solution. Since the number of observations $n = 208$ is not too big compared to other datasets, it is strange that the model cannot solve to optimality in two hours.

For around 60% of the datasets, we do obtain results where OCT performs better than CART as we expect, since OCT creates an optimal decision tree, and CART uses a top-down method that is not optimal.

There are some datasets where OCT performs a little less than CART, and this could be explained by the the fact that we are taking samples for the training, validation and testing set every time we run the model. On average, we expect that these differences in samples are compensated by the number of times we take a new sample (five in this case), but for small datasets this sampling can still have a great influence. For example, in "Echocardiogram", there are only 61 observations with 8 features. We observed that the out-of-sample accuracies differ greatly for the five new runs, varying between 60% and 80%. Furthermore, the accuracy for CART for this dataset is almost 100%, while this value in Bertsimas & Dunn (2017) is a lot closer to the accuracy of OCT. This uncommon behavior cannot be totally explained by the dataset.

For some datasets, for example "Monks-problems-1" and "Monks-problems-2", the OCT model performs much better than in Bertsimas & Dunn (2017). This can be caused by the differences in dataset: for these two datasets there were $p = 11$ features in Bertsimas & Dunn (2017), while there are only $p = 6$ features contained in the datasets used here.

We found that on average, OCT performs better than CART. For some datasets, OCT performs significantly worse than CART, but this is probably caused by a small mistake somewhere, because this pattern is not observed for other datasets. Some datasets could not solve to optimality within the time limit of two hours, and these also have a lower accuracy for OCT, because the optimal decision tree could not be found. There are also some datasets for which the performance of OCT is a little worse than for CART, but these small differences are partly caused by the random sampling that is made for the partition of training, validation and test set. For the other datasets, we observed that OCT performs better than CART for decision trees with depth $D = 2$.

4.3 OCT model with time limits

Next, we investigated the influence of the time limit on the accuracy of the OCT model. The time limit is increased up to the maximum running time, differing per dataset. This maximum running time is equal to the time needed to solve the model to optimality, when there is no restriction on the time. Based on the maximum time limit, we tested the model on different time limits, to see if the maximum out-of-sample accuracy is already obtained before the model is solved optimally. We observed different patterns for the different datasets: some already reach the maximum accuracy at the smallest time limit, and others' accuracy increases step-by-step when the time limit becomes larger. In Figures 1 and 2, the trade-off between time and accuracy for two datasets is presented, where the time in seconds is on the horizontal axis, and the out-of-sample accuracy on the vertical axis.

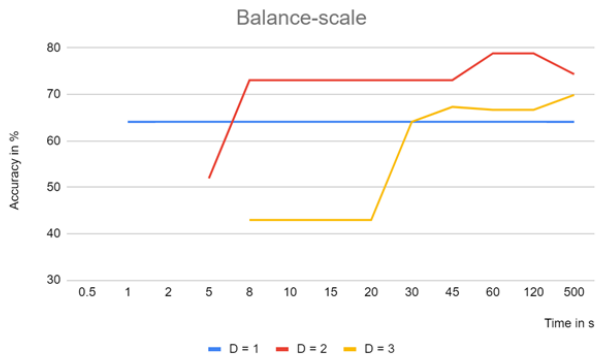


Figure 1: Trade-off between time and accuracy for "Balance-scale"

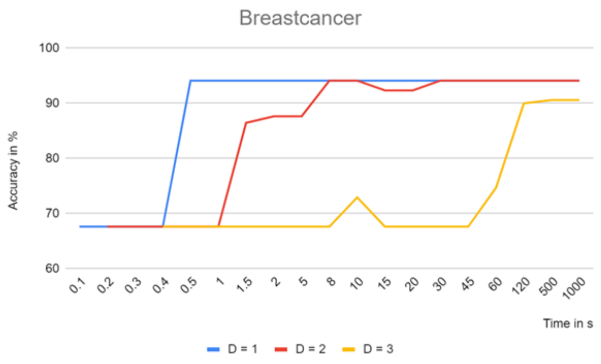


Figure 2: Trade-off between time and accuracy for "Breastcancer"

In Figure 1, we observe that the accuracy for a decision tree with $D = 1$ does not depend on the time limit, because even for a limit of one second, the OCT model reaches its maximum value. For $D = 2$ and $D = 3$, we see an upward sloping graph, where the accuracy improves over time. These lines start at $t = 5$ and $t = 8$ for $D = 2$ and $D = 3$, respectively, indicating that no solution was found for smaller time limits. Since we do not use warm starts of lower depths for higher depths, the solution for $D = 3$ starts again at a low accuracy, and not at the final accuracy obtained for $D = 2$. For $D = 2$, we can observe a decrease in the accuracy when the time limit is increased from 120s to 500s, which is illogical, because we do not expect the solution to be worse when it takes more time to run, because the previous solution would always be better then. This decrease in accuracy can only be explained by the fact that the solver does not use the exact same path to come to a solution every time it runs, but this behavior is strange for the large increase in time limit.

In Figure 2, the decision trees for $D = 1$ and $D = 2$ reach the same final accuracy of 94.1%, but the tree with $D = 1$ reaches this accuracy in 0.5 seconds, while this takes a bit longer for $D = 2$, namely 8 seconds. The smallest tree with depth $D = 1$ jumps from the first obtained solution at 0.1 seconds to the optimal solution in 0.5 seconds, without intermediate steps. The tree with depth 2 takes more steps to reach the optimal result. The highest accuracy reached for the tree with depth $D = 3$ is lower than that of $D = 1$ and $D = 2$. Again, this is caused by not using a warm start. The figures with trade-offs between time and accuracy for some other datasets can be found in Appendix 6.1.

Some other datasets that were tested for the trade-off between time and out-of-sample accuracy have a similar pattern as in Figure 3, where the out-of-sample accuracy is the same for all depths $D = 1, 2, 3$ and the only difference is the time after which a solution is obtained. For “Banknote-authentication”, the solution is found after 0.1 second for $D = 1$, after 0.5 seconds for $D = 2$ and 15 seconds for $D = 3$. The figures of the other datasets can be found in Appendix 6.1.

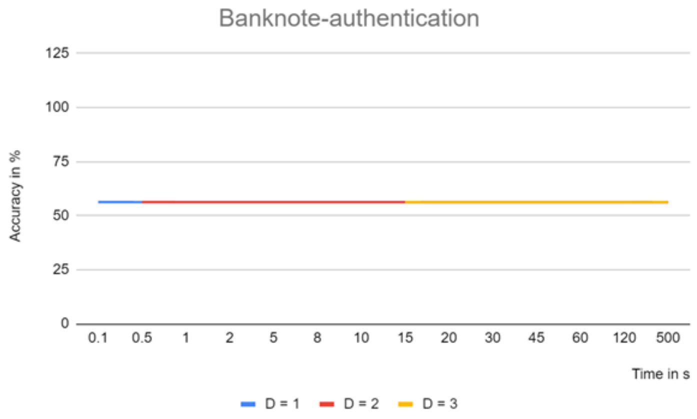
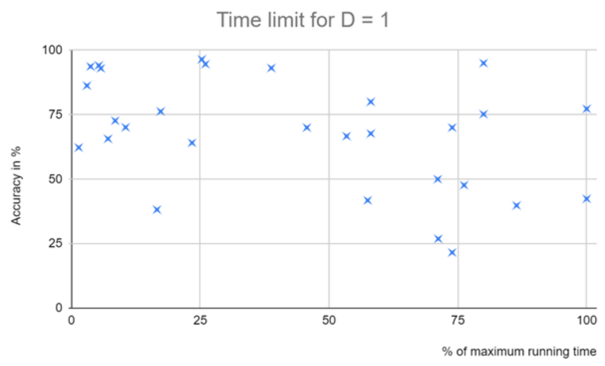


Figure 3: Trade-off between time and accuracy for “Banknote-authentication”

Furthermore, we look at the percentage of the maximum running time that is needed to reach the accuracy that is optimal for depths $D = 1$ and $D = 2$ for all datasets. The results are presented in Figures 4 and 5. On the horizontal axis, the percentage of maximum running time is plotted. The vertical axis represents the corresponding out-of-sample accuracy. For $D = 1$, in Figure 4, we observe that for all datasets but two, the optimal tree is found earlier than the maximum running time. For more than a third of the datasets, the optimal tree was found in less than 25% of the maximum running time, indicating that the optimal tree is already found in the beginning of the process, but it apparently takes some more time to guarantee that this is the optimal solution.

For the decision trees with depth $D = 2$, the results are shown in Figure 5. For a quarter of the datasets, the optimal accuracy is found after less than one percent of the maximum running time. For 80% of the datasets, this maximum running time was found in under 50% of the maximum running time. A few datasets took long to solve, and could not find a better solution earlier than the maximum running time. These were often big datasets ($n > 1000$) that required more time, and took longer to find a feasible solution at all, compared to other datasets where a non-optimal solution was already found at a very short time limit.



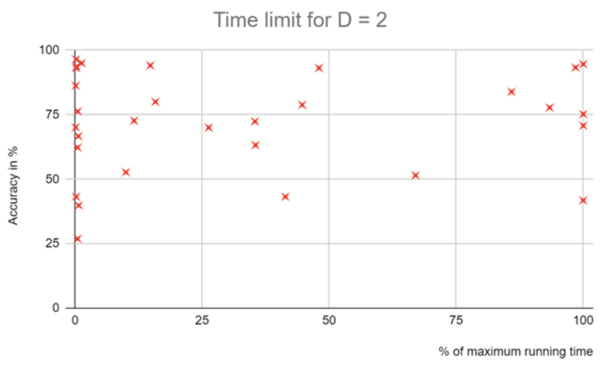


Figure 4: Plot of the percentage of the maximum running time needed to reach the accuracy that is optimal for D = 1

Figure 5: Plot of the percentage of the maximum running time needed to reach the accuracy that is optimal D = 2

For almost all datasets, for both decision trees of depth $D = 1$ and $D = 2$, the optimal solution is found faster than the maximum running time. At the cost of not knowing with certainty if the solution is optimal, we can find a very good solution in half to three-quarters of the time.

5 Conclusion

In this research, we investigated the performance of an optimal classification tree on real-world datasets. We presented an MIO formulation to create an optimal decision tree in one single step. This OCT model is compared to the top-down method CART. Furthermore, we looked at the trade-off between the maximal running time and the out-of-sample accuracy for OCT.

It is found that in most cases, OCT performs better than CART. For some datasets, OCT performs significantly worse than CART, but this is probably caused by a small mistake somewhere, because this pattern is not observed for the other datasets. Some datasets could not solve to optimality within the time limit of two hours, so these datasets also have a lower accuracy for OCT, because the optimal solution could not be found. For the other datasets, we observed that OCT performs better than CART on average, with an accuracy improvement of 1-2% for OCT on decision trees with depth $D = 2$.

For the trade-off between the time limit and the accuracy, we found that for depth $D = 1$, the optimal solution is already found after two seconds for the first 15 datasets. For 80% of these datasets, the optimal decision tree is found even after 0.5 seconds. For decision trees with depth $D = 2$, the differences in time needed to reach the optimal solution are bigger. In around 50% of the cases, the optimal tree was found after 0.5 seconds, but we need to take into account that for 40% of the datasets, the optimal solution was found directly at the first solution, which was often obtained after around 0.5 seconds. This means that the obtained solution did not improve when the time limit was increased: the first obtained solution was already the optimal one. For the other 50% of the datasets, where the optimal tree was not found after 0.5 seconds, the time needed to find the optimal tree varies from 2 to 500 seconds, with only a few cases where 500 seconds were needed. For the other datasets, the optimal tree was found in less than a minute. Even though a best solution can be found within seconds, it is not certain that this solution is optimal, when we only look at the solution obtained after, for example, 0.5 seconds. When using the OCT model to find an optimal decision tree, it is also valuable to know that you found the optimal decision tree, but this is at the cost of time.

We also looked at the percentage of maximum running time that is needed to find the optimal solution, where the maximum running time is the time needed to find the optimal decision tree when there is no time restriction. Here, we found that for depth $D = 1$ for more than a third of the datasets, the optimal solution could be found in less than 25% of the maximum running time. More than 80% of the datasets could find the optimal solution within 75% of the maximum running time. For decision trees with depth $D = 2$, the optimal solution was found in under 50% of the maximum running time for 80% of the datasets.

In both experiments with finding the optimal decision tree with OCT and looking at the different time limits, no warm starts from lower depths were used for higher depths due to time limitations. This influences the time needed to find an optimal solution, and this is of importance especially for the trade-off between the time limit and the obtained solution. In some of the experiments, the optimal solution obtained for $D = 3$ has a lower out-of-sample accuracy than that of the decision tree for $D = 2$, and this implication could be avoided when using the solution of $D = 2$ as a warm start for the problem with $D = 3$. For the solutions for $D = 1$ and $D = 2$, it is observed that the best solutions are obtained after only a few seconds in most cases, indicating that the missing warm start does not influence the best found solution much, as this would only give a lower upper bound at the beginning of solving and an optimal solution is still found fast without the warm start.

Further research is thus needed to see the more precise trade-off between time and accuracy for $D = 3$ by using warm starts. Furthermore, more different datasets could be tested to obtain a more correct conclusion on the performance of OCT compared to CART. In the datasets that are used for these experiments, uncommon behavior of low accuracy for OCT, while the accuracy for CART is a lot higher, is in many cases associated with a high number of features ($p \geq 30$). It is therefore interesting to test more datasets with a lot of features, to see if this pattern occurs in those experiments too. In this research, the comparison between OCT and CART is only made for decision trees with depth $D = 2$, and this could be expanded to trees of higher depth.

6 Appendix

6.1 Trade-off time and accuracy

In Figures 6 - 12, the trade-off between time and accuracy is shown for the datasets “Congressional-voting-records”, “Dermatology”, “Blood-transfusion”, “Breast-cancer-diagnostic”, “Breast-cancer-prognostic”, “Car-evaluation” and “Contraceptive-method-choice”.

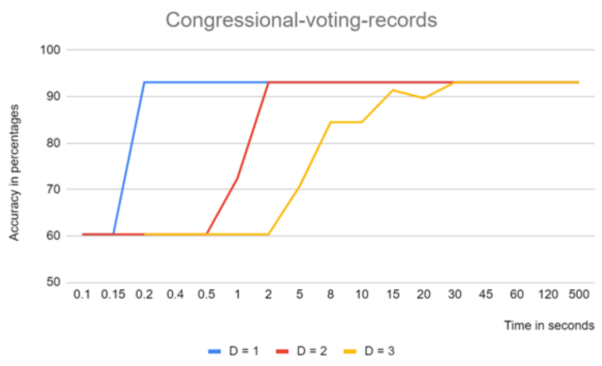


Figure 6: Trade-off between time and accuracy for “Congressional-voting-records”

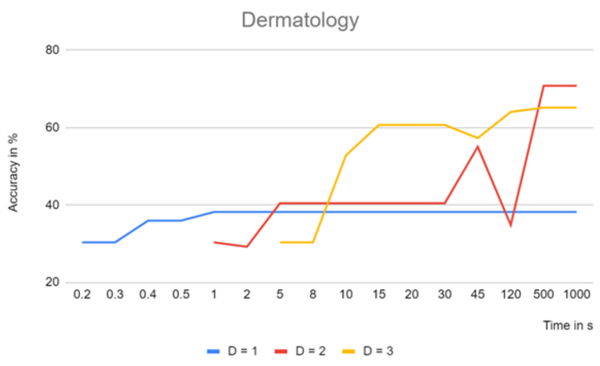


Figure 7: Trade-off between time and accuracy for “Dermatology”

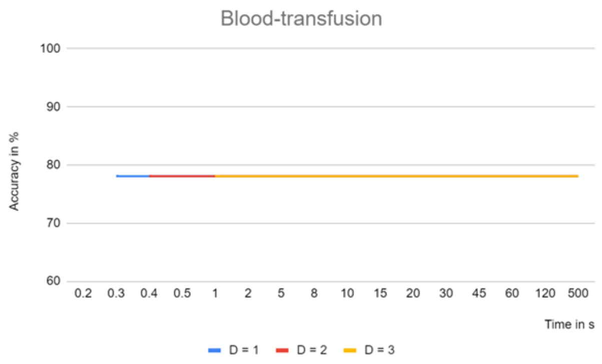


Figure 8: Trade-off between time and accuracy for “Blood-transfusion”

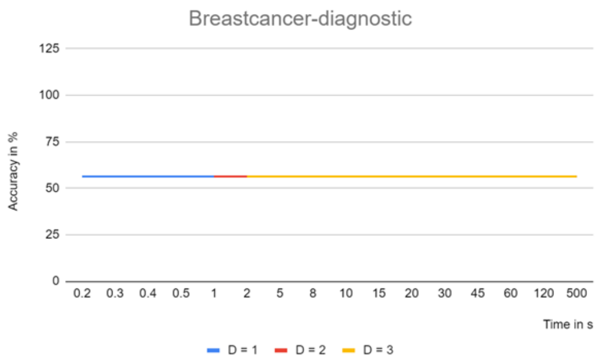


Figure 9: Trade-off between time and accuracy for “Breast-cancer-diagnostic”

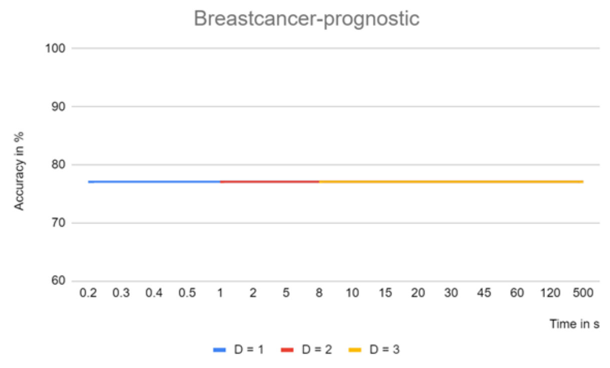


Figure 10: Trade-off between time and accuracy for "Breast-cancer-prognostic"

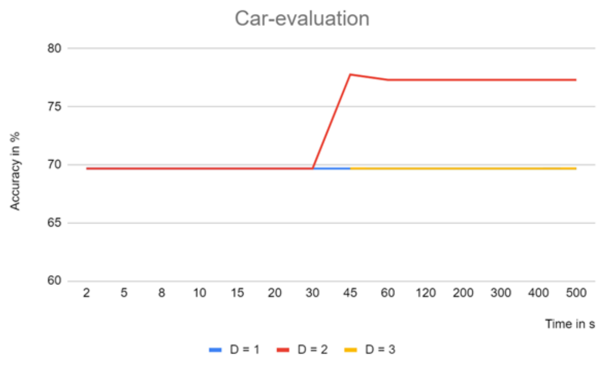


Figure 11: Trade-off between time and accuracy for "Car-evaluation"

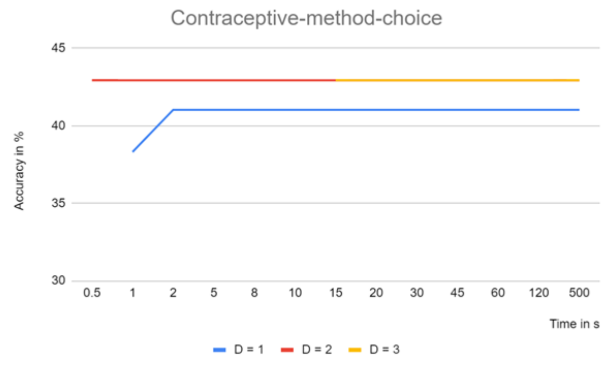


Figure 12: Trade-off between time and accuracy for "Contraceptive-method-choice"

6.2 Programming

In this section, a short description of the programming files is given. For CART, the RPART package of the R programming language is used (Therneu et al., 2015). For each run, all indices of the dataset are shuffled to generate a random training, validation and test set. With the training set, a decision tree is created. Based on this tree, the optimal value for the complexity parameter is determined. With this value of the complexity parameter, a new tree is created on the combined training and validation set with the RPART function, with the complexity parameter as input for "cp", and the value for N_{min} as input for "minbucket". The tree is pruned afterwards, to avoid overfitting. This final tree is tested on the last 25% of the dataset to obtain the out-of-sample accuracy. This whole process, including the shuffling of the data, is done five times for each dataset, and the average out-of-sample accuracy is reported.

For the OCT model, CPLEX is used in Java. For each dataset, there is a new class in Java. All classes contain the same model and methods, the only differences are some dataset related inputs, such as the number of data points and all the $\boldsymbol{x}_i$ values that need to be normalized. For some datasets, the input given to Java is already normalized, for some other datasets this is done in the class itself, when loading the data.

There are some general methods that are used to either test the data, or give outputs that are needed for some constraints. There is a method to get a specific column of a two-dimensional array, which is needed to calculate the inner product between $\boldsymbol{a}_t^T$ and $\boldsymbol{x}_i$ in (14). Related to this, there is also a method that calculates the inner product when two vectors are given. There are also methods to find the parent of a node, the ancestors of a node and the left or right ancestors. These methods return a list of integers of the indices of the parent, ancestors, left ancestors and right ancestors, respectively. The last methods are to calculate the ϵ vector in (14) , which is split up in two methods. One calculates the ϵ value for one feature, and the other summarizes all ϵ values for each feature in a vector. Then there are two methods for the OCT model: one for validation, and one for creating the tree after the value of α is determined. The validation model contains the penalty constraint of the number of splits being less or equal than C. This model is used to determine the values of α, which are related to the values of C, for which the model is optimal. Once this α is found, the OCT model can be solved on the combined training and validation set to create the final tree. This tree is then tested on the test set, to obtain the out-of-sample accuracy. There is a method that tests the accuracy of the final tree, when the values for a, b and c are given as input, together with the test set. Each run, a random training, validation and test set are generated. The whole process described above is done five times for each dataset, and averages are reported. For the trade-off between time and accuracy, one shuffled set of each dataset is used for each run with a different time limit, to obtain a fair comparison between the time limits.

References

Amram, M., Dunn, J., & Zhuo, Y. (2022). Optimal policy trees. *Machine learning*. doi: https://doi.org/10.1007/s10994-022-06128-5

Arthanari, T., & Dodge, Y. (1981).
Mathematical programming in statistics, *341*. (New York: Wiley)

Bennett, K. P. (1992). Decision tree construction via linear programming. *Proceedings of the 4th midwest artificial intelligence and cognitive science society conference*, 97-101.

Bennett, K. P., & Blue, J. (1996).
Optimal decision trees. (Rensselaer Polytechnic Institute Math Report No. 214)

Bertsimas, D., & Dunn, J. (2017, JUL). Optimal classification trees. *Machine Learning*, *106*(7), 1039-1082. doi: 10.1007/s10994-017-5633-9

Bertsimas, D., Dunn, J., Gibson, E., & Orfanoudaki, A. (2022). Optimal survival trees. *Machine learning*. doi: https://doi.org/10.1007/s10994-021-06117-0

Bixby, R. E. (2012). A brief history of linear and mixed-integer programming computation. *Documenta Mathematica. Extra Volume: Optimization Stories*, 107-121.

Breiman, L., Friedman, J., Olshen, R., & Stone, C. (1984). Classification and regression trees. *European Journal of Operational Research*, *19*(1), 144. doi: 10.1016/0377-2217(85)90321-2

Cox, L., A, J., Yuping, Q., & Kuehner, W. (1989). Heuristic least-cost computation of discrete classification functions with uncertain argument values. *Annals of Operations Research*, *21*(1), 1-29.

Esmeir, S., & Markovitch, S. (2007). Anytime learning of decision trees. *The Journal of Machine Learning Research*, *8*, 891-933.

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

Kotsiantis, S. B. (2013, APR). Decision trees: a recent overview. *Artificial Intelligence Review*, *39*(4), 261-283. doi: 10.1007/s10462-011-9272-4

Lichman, M. (2013).
UCI machine learning repository. doi: http://archive.ics.uci.edu/ml/index.php

Murthy, S., & Salzberg, S. (1995a). Lookahead and pathology in decision tree induction. *IJCAI*, 1025-1033.

Murthy, S., & Salzberg, S. (1995b). Decision tree induction: How effective is the greedy heuristic? *KDD*, 222-227.

Nemhauser, G. L. (2013). Integer programming: The global impact. *Presented at EURO, INFORMS, Rome, Italy, 2013.* doi: http://euro-informs2013.org/data/http_/euro2013.org/wp-content/uploads/nemhauser .pdf

Norton, S. W. (1989). Generating better decision trees. *IJCAI*, *89*, 800-805.

Payne, H., & Meisel, W. S. (1977). An algorithm for constructing optimal binary decision trees. *IEEE Transactions on Computers*, *100*(9), 905-916.

Quinlan, J. (1986). Induction of decision trees. *Machine learning*, *1*(1), 81-106.

Quinlan, J. (1993).
C4.5: Programs for machine learning. (San Francisco, CA: Morgan Kaufmann)

R Core Team. (2015).
R: A language and environment for statistical computing. (Vienna: R Foundation for Statistical Computing) doi: http://www.R-project.org/

Therneu, T., Atkinson, B., & Ripley, N. (2015). *rpart: Recursive partitioning and regression trees*. doi: http://CRAN.R-project.org/package=rpart ,Rpackageversion4.1-9.