Erasmus University Rotterdam
Erasmus School of Economics
Bachelor Thesis Econometrie en Operationele Research

The out-of-sample performance of an MIP formulation for constructing optimal decision trees

Merlijn Oonk (541608)

Supervisor:	Riley Badenbroek
Second assessor:	M.H. Akyuz
Date final version:	2nd July 2023

Abstract

Decision trees are used extensively for classification problems, but it remains hard to construct optimal trees. This research explores the MIP formulation of Verwer and Zhang (2017) for creating optimal decision trees. In particular the performance on out-of-sample data is of interest, as it has not been examined before. Next to the MIP formulation, we investigate the performance of the MIP formulation with warm start solutions generated by the CART heuristic (MIP_WS). We evaluate the differences in classification accuracy of in-sample and out-of-sample data and compare those results to CART. Lastly, we prune the trees using the Reduced Error Pruning algorithm to explore whether this reduces overfitting on the training data. We find that MIP_WS generally is the most accurate model on the training data, but it does not outperform CART on out-of-sample data or after pruning the trees. Since CART is faster and typically constructs sparser trees, it is preferred over the MIP models for use in practice.

1 Introduction

Decision trees are used extensively for classification problems. A trained decision tree can label new observations in an accurate manner, while the trees typically are very transparent and easy to interpret. Therefore, they are widely used in practice. The method is most popular in the medical sector: Shouman et al. (2011) state that decision tree techniques are one of the most successful methods to diagnose a heart disease, while Bonner (2001) and Podgorelec et al. (2002) show different ways how decision trees can be used in mental health clinical practice and in medicine, respectively.

Laurent and Rivest (1976) show in 1976 that the construction of an optimal decision tree is an NP-hard problem, which leads to the development of various algorithms to find good decision trees. The Classification And Regression Tree (CART) algorithm as first proposed by Breiman et al. (1984) is still widely used. CART uses a greedy top-down approach in which it performs optimization problems starting from the root node. In each decision node, the algorithm assigns the best split it can find to the node. This directly shows the disadvantage of this greedy method, since decisions in future nodes are not taken into account at the local optimization problems. This could lead to weak solutions of the algorithm. Other algorithms that are often used are ID3 (J. R. Quinlan (1986)) and C4.5 (J. Quinlan (1993)). These algorithms are similar to CART, since they also use a greedy top-down procedure to come to a solution.

Because these algorithms generally do not result in optimal solutions, Verwer and Zhang (2017) and Bertsimas and Dunn (2017) both model the decision tree classification problem as a mixed integer programming (MIP) problem in order to explore the possibilities of creating optimal decision trees. They both find that their models outperform CART when running them for a manageable amount of time. The best solutions are found when using the solution of CART as a warm start for the MIP problems. However, it remains difficult to consistently obtain optimal solutions for large datasets and growing depths of the tree.

After these findings, there has been more attention for modelling the decision tree classification problem as an MIP problem. Carrizosa et al. (2021) show the flexibility of an MIP problem as it can easily allow for inclusion and avoid discrimination. Verwer and Zhang (2019) design a linear program formulation with a much smaller number of decision variables and constraints

than in earlier formulations, and furthermore the decision variables are independent from the size of the datasets. This method gives better solutions than earlier formulations while having a shorter running time. Zhu et al. (2020) develop an MIP formulation in order to train multivariate decision trees, while Günlük et al. (2021) use an MIP formulation to deal with categorical data in decision trees.

Although the MIP formulation of Verwer and Zhang (2017) gives new insights about creating optimal decision trees, the researchers claim to have found a better method than before without testing their method on out-of-sample data. This means they cannot guarantee that their methods can handle overfitting, because they only have strong results for in-sample performance on only a few datasets.

This research will focus on further exploring the methods of Verwer and Zhang (2017) and investigate if they perform as well as they claim. To be specific, we examine whether their methods obtain similar results on new datasets and how the performance of the trees change on out-of-sample data. We then prune the trees by using the Reduced Error Pruning (REP) algorithm to evaluate whether this reduces overfitting on the training data. We investigate the performance of two methods: the MIP formulation as described by Verwer and Zhang (2017) and this same MIP formulation with the solutions of CART as a warm start. We compare all the results to the CART heuristic to assess which method performs best for use in practice. All data we use is coming from the UCI Machine Learning Repository (Dua and Graff (2017)).

On the basis of our experiments we find that the MIP problem that uses CART as a warm start is the most accurate on the training data. However, it performs similar to CART on test data both before and after pruning the trees. Since CART generally creates sparser trees and has a faster running time than the MIP formulations, it is the preferred method for use in practice.

This research contributes to the literature by further exploring MIP models that try to create optimal decision trees. In particular the analysis on the classification of test data and the performance of pruned trees expand the current knowledge of the use of MIP formulations to create decision trees.

The remainder of this paper looks as follows: in Section 2 we give a brief problem description, in Section 3 we explain the performed data transformation. After this we discuss the methodolgy in Section 4 and present the results in Section 5. Lastly we conclude our findings in Section 6.

2 Problem description

In this section we shortly explain the main problem and introduce some definitions that are used to describe certain features of a decision tree. In figure 1 we show a decision tree with numbered nodes. In a decision tree, every leaf node (in this case node 8-15) gets assigned a label. In every node except the leaf nodes (node 1-7), there is a decision rule in order to split the data. A decision rule consists of a feature variable and a threshold value. All observations that arrive in a node go right in that node if their feature value of the relevant variable is larger than the threshold, and go left otherwise. The depth of a node refers to the layer the node is in, starting at depth 0 for the root node.

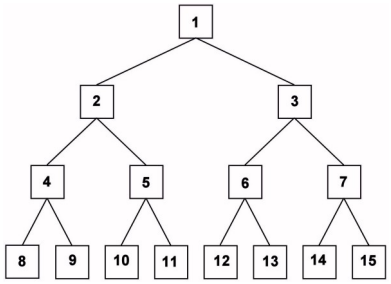

Figure 1: Decision tree with depth $k = 3$

There are two types of decision trees: classification trees and regression trees. For classification trees the goal is to correctly label as many of the observations as possible. In regression trees, the target variables are continuous variables instead of labels. The prediction of a single observation tries to come as close to the target value as possible.

Currently, decision trees often are created by heuristics such as CART. These heuristics tend to quickly find good decision trees, but the downfall is that they generally do not find optimal trees. Our goal is to find optimal decision trees by using an MIP formulation. If the method does not find optimal trees, we can still explore whether creating trees by using an MIP problem can compete with the heuristics. In practice, decision trees are often used on data that is not utilized to build the tree. It is a common problem that the trained tree is overfitting on the training data. This generally leads to a weaker performance of the tree on out-of-sample data. To avoid this, pruning methods are widely used to discard certain subtrees. This can reduce the specificity of the tree on the training data and improve the out-of-sample accuracy.

3 Data transformation

In this section we explain the transformation. We follow the methods of Verwer and Zhang (2017) and interpret it as the implementation of the 3 steps stated below. An example of the data transformation of a single feature column is shown in Table 1.

- **Step 1**. For every sorted feature column, we assign new integer values (starting from 0) to every unique feature value. This means equal feature values are mapped to the same integer. In this way all feature values are mapped to integers while the order of the feature values are intact for every column.

- **Step 2**. For every sorted column, various integer values are allowed to be taken together. If a set of successive integers all have the same target value, and there are no equal integers with different target values, all observations of this set are mapped to the same integer.

- **Step 3**. The most occurring integer is mapped to 0, while remaining the current order: for every column, find the integer value that is occurring most often. If the value of this integer is i, subtract i from all variables in the column to maintain the correct order. The result is the final feature column.

By mapping all feature values to integers in Step 1, the thresholds in the decision rules can be

forced to be integers as well. This will be used in the formulation of the MIP model. The first two columns of the example in Table 1 correspond to Step 1 of the data transformation.

In Step 2 of the transformation, certain sets of integers are mapped to a single integer value. This is allowed because these sets all have the same target value and therefore an optimal decision tree strives to group these values together. If integer values 3, 4, 5 and 6 of a certain feature q all have the same target value, then an optimal decision rule in a node will never be $q \leq 4$, because this will split these variables. This step results in a maximum feature value that is as low as possible, which is the value of the big-M that is used in the MIP formulation. An example of Step 2 of the data transformation is shown in Table 1. The second column contains the sorted integer feature values while the corresponding target values are in the last column. The third column is the result of Step 2 of the data transformation. In the second column, the feature value 2 cannot be taken together with other integers, because it has target values of both 0 and 1. The feature values 3 and 4 can be taken together because they all have the same target value. Because feature value 4 becomes feature value 3, feature value 5 becomes feature value 4 to prevent any gaps in the feature values. This results in a maximum feature value that is as small as possible.

In the last step of the transformation the most frequently occurring feature value is mapped to 0. In the example in Table 1, a feature value of 3 occurs the most in the third column and thus it should be mapped to 0. To realize this, 3 should be subtracted from all values in the third column to remain the order of the values. The result of this step is shown in the fourth column. This concludes the data transformation.

Sorted Feature Value	**Integer Feature Value**	**Merged Feature Value**	**Final Feature Value**	**Target Value**
3.2	0	0	-3	1
3.5	1	1	-2	0
3.6	2	2	-1	0
3.6	2	2	-1	1
3.7	3	3	0	1
3.7	3	3	0	1
3.9	4	3	0	1
4.5	5	4	1	0

Table 1: An example of the data transformation

4 Methodology

In this section we discuss our methodology . In Section 4.1 we describe the MIP formulation for both classification and regression trees. After this, we explain how the Reduced Error Pruning works in Section 4.2.

4.1 MIP Formulation

The MIP formulation we use is the formulation proposed by Verwer and Zhang (2017). We now explain the full model and mention some small adjustments we make. In Table 2 we show an overview of the notation that is used in the formulation for classification trees. Note that the domain of certain variables changes in the encoding for regression trees.

Symbol	**Definition**
R	Set of all data rows
Q	Set of all features
U	Set of all decision nodes
L	Set of all leaf nodes
Y	Set of all possible target values
LF	Lowest feature value in the data
UF	Highest feature value in the data
k	Maximum depth of the tree
$d(u)$	The depth of node u, with depth of the root node equal to 0
a_{rq}	Data value of row r, feature q
$t(r) \in Y$	Target value of row r
$p_{ly} \in \mathbb{B}$	Prediction for leaf l and target y, 1 if prediction of leaf l is target value y
$e_r \in \mathbb{B}$	Prediction error for row r, 1 if the prediction is incorrect
$f_{qu} \in \mathbb{B}$	Decision variable, 1 if feature q is in the used decision rule in node u
$d_{hr} \in \mathbb{B}$	Decision variable, 1 if the path of row r goes right in the step that reaches depth h
$c_u \in [LF, UF], \in \mathbb{Z}$	Decision variable, the threshold of the decision rule in node u

Table 2: Summary of the notation for the encoding of classification trees

The objective is to minimize the total prediction error

$$\min \sum_{r \in R} e_r, \tag{1}$$

where R is the set of all data rows and e_r is the prediction error of data row r. In (11) we show that the binary variable e_r will be set to 0 if the classification of row r is correct, and 1 if the classification is incorrect.

Every node, excluding the leaf nodes, requires exactly one of the feature variables to be used in the decision rule for that node

$$\sum_{q \in Q} f_{qu} = 1, \forall u \in U, \tag{2}$$

where Q is the set of all feature variables and U is the set of all decision nodes in the tree. f_{qu} is a binary decision variable that is 1 if node u uses feature q as decision rule and 0 otherwise.

The upcoming two constraints ensure that if row $r \in R$ goes left (right) at depth h, the value of feature $q \in Q$ of row r, denoted as a_{rq}, should be smaller (larger) than the used threshold c_u in node $u \in U$ when feature q is used in the decision rule. In order to realize this we first define

the function

$$\mathbf{dlr}(h,u,r) = \begin{cases} d_{hr} & \text{if the path to node } u \text{ goes left in the step that reaches depth } h \\ 1-d_{hr} & \text{if the path to node } u \text{ goes right in the step that reaches depth } h \end{cases}$$

where d_{hr} is a binary decision variable that is equal to 1 if row $r \in R$ goes left in the step to reach depth h (so goes left at depth $h-1$), and 0 if it goes right. To illustrate this we look back at the decision tree in Figure 1. The function $\mathbf{dlr}(2,6,r)$ returns d_{2r} since the path to node 6 goes left at depth 1. In the same way $\mathbf{dlr}(1,6,r)$ returns $1-d_{1r}$ since the path to node 6 goes right at depth 0. This formula will be used to determine whether the path of row r goes through node u: if we assume the path of row r goes through node u, then all $\mathbf{dlr}(h,u,r)$ will be 1 for $h \leq d(u)$, where $d(u)$ is the depth of node u. If the path of row r reaches node 6, we have just seen that the formula returns $1-d_{1r}$ for $h=1$ and d_{2r} for $h=2$, which are both equal to 1. For node 7, $\mathbf{dlr}(2,7,r)$ returns $1-d_{1r}$ which is equal to 0. This means that $\sum_{1\leq h\leq d(u)} \mathbf{dlr}(h,u,r) = d(u)$ if and only if the path of r goes through node u, which will be an important result that is used in the constraints.

Since the following two constraints require a big-M formulation, we first define the two different big M's we use. We explain why we use these exact values after we show the constraints they are used in.

$$M_r = \max\{\left(a_{rq} - LF\right), q \in Q\}, \tag{3}$$

$$M_r' = \max\{\left(UF - a_{rq}\right), q \in Q\} + \epsilon, \tag{4}$$

where ϵ is a small value to be set later. LF and UF are the lowest and highest feature values in the dataset, respectively. M_r essentially is the difference between the largest feature value in row $r \in R$ and the smallest feature value in the dataset. M_r' is the difference between the largest feature value in the dataset and the smallest feature value in row r plus a small value ϵ. We later show that these are tight big-M values in the constraints they appear in.

Using the 3 equations and formulas above we can force the rows to correctly take the right or left branch using the two constraints

$$M_r d_{d(u),r} + \sum_{1\leq h\leq d(u)} M_r \mathbf{dlr}(h,u,r) + \sum_{q\in Q} a_{rq} f_{qu} \leq M_r\left(d(u)+1\right) + c_u, \forall u \in U, r \in R, \tag{5}$$

$$\sum_{1\leq h\leq d(u)} M_r' \mathbf{dlr}(h,u,r) + c_u + \epsilon \leq \sum_{q\in Q} a_{rq} f_{qu} + M_r' d_{d(u),r} + M_r' d(u). \tag{6}$$

Note that the constraints are slightly different than the constraints in the paper of Verwer and Zhang (2017), since their constraint misses an M_r and M_r' on the right-hand side for (5) and (6) respectively. Another difference is the added ϵ in (6). Without an ϵ the equation should be a strict inequality to ensure a row cannot 'choose' whether it goes right or left when their feature value is equal to the threshold of the decision rule c_u. Because the feature values and thresholds

are integers we can set $\epsilon = 0.5$.

To show how these constraints work we look at a certain row $r \in R$ and node $u \in U$. The goal of (5) is to make sure that if the path of row r reaches node u and if the path goes left at node u, then the constraint should reduce to

$$\sum_{q \in Q} a_{rq} f_{qu} \leq c_u, \tag{7}$$

because then the relevant data value is smaller than the threshold in the used decision rule and thus the path goes left in the next step. As shown before, the term $\sum_{1 \leq h \leq d(u)} \mathbf{dlr}(h, u, r) = d(u)$ if and only if row r goes through node u. This means that the first sum in the constraint cancels out with $M_r d(u)$ on the right-hand side. The term $d_{d(u),r}$ is equal to 1 if row r goes left at $d(u)$ and this cancels out with the last term M_r on the right-hand side. This leaves the constraint as in (7), which means that the feature value that is used in the decision rule should be smaller than the threshold value c_u. The other terms in the constraint ensure that it only reduces to this form when row r reaches node u and goes left at depth $d(u)$. The constraint becomes redundant for the nodes row r does not go through, or if the row goes right at depth $d(u)$.

When row r does not reach node u, $\sum_{1 \leq h \leq d(u)} M_r \mathbf{dlr}(h, u, r)$ will be at most equal to $M_r (d(u) - 1)$, since at least one of the directions of the path of row r is different from the path to node u. Therefore, when only focusing on the terms with a big-M in it, the left-hand side of (5) is at least M_r smaller than the right-hand side. This is also the case when row r does reach node u but goes right at node u. This means that in the worst case

$$\sum_{q \in Q} a_{rq} f_{qu} \leq M_r + c_u, \tag{8}$$

and

$$\sum_{q \in Q} a_{rq} f_{qu} = \max\{a_{rq}, q \in Q\}.$$

Filling in this value in (8) gives

$$\max\{a_{rq}, q \in Q\} \leq M_r + c_u.$$

The smallest value of M_r for which this is true is M_r as defined in (3), because this would mean

$$LF \leq c_u, \tag{9}$$

which always holds true since $c_u \in [LF, UF]$. This immediately shows that M_r is a tight big-M value, because in the worst case scenario the constraint reduces to (9), where the left-hand side is not allowed to be any larger since a value larger than LF would lead to a non-redundant constraint. In all other cases the left-hand side will be strictly smaller than LF and thus these constraints become redundant.

Constraint (6) works exactly the same as (5), but now for when row r is going right at depth u. In this constraint the relevant data value has to be strictly larger than the threshold of the decision rule in case row r reaches node u and goes right at depth $d(u)$. The derivation of M_r'

is similar to the derivation of M_r and again leads to a tight big-M as defined in (4).

All leaf nodes should be classified as exactly one of the possible target values

$$\sum_{y \in Y} p_{l,y} = 1, \forall l \in L, \tag{10}$$

where Y is the set of target values, L the set of leave nodes in the tree and $p_{l,y}$ is a binary variable which is 1 if leaf node l is labeled as target y, and 0 otherwise.

The prediction error for row $r \in R$ should be 1 for every incorrect classification

$$\sum_{1 \leq h \leq k} \mathbf{dlr}(h, l, r) + \sum_{y \in Y, y \neq t(r)} p_{l,y} \leq e_r + k, \forall l \in L, r \in R, \tag{11}$$

where $t(r)$ is the target value of row r. If row r ends up in leaf l, $\sum_{1 \leq h \leq k} \mathbf{dlr}(h, l, r) = k$. This then forces e_r to be 1 if leaf l is classified as one of the target values that is not equal to the correct target value $t(r)$. If the classification of row r is correct, or r does not end up in leaf l, e_r will be equal to 0 since the left-hand side of the constraint would never exceed k.

In order to strengthen this formulation, Verwer and Zhang (2017) add two restrictions. The first one they add is simply the bounding of all c_u by stating it should be greater or equal than LF and at maximum UF. Since we already gave this domain to all c_u we do not add this as a new constraint in our formulation, but it is probable that the way of adding this restriction is not relevant for the solver.

The second restriction they add to the model to strengthen the formulation only applies in the case of binary classification. The constraint ensures that two leaf nodes with the same parent node have a different label

$$p_{l,y} + p_{l',y} = 1, \forall y \in Y \text{ and all pairs of leaves with the same parent node } (l, l'). \tag{12}$$

Since this restriction only strengthens the model in case of exactly two target values, we add a new restriction to the model that also strengthens the formulation in the case of more than two target values. Two nodes with the same parent node cannot be classified as the same target value

$$p_{l,y} + p_{l',y} \leq 1, \forall y \in Y \text{ and all pairs of leaves with the same parent node } (l, l'). \tag{13}$$

The full model then consists of the objective function in (1) and constraints in (2), (5), (6), (10) and (11). (12) is added to the formulation in case of two possible target values, while (13) is added in case of more than two target values.

4.1.1 Regression trees

In order to change the model to a formulation for a regression tree that minimizes the sum of absolute prediction errors, (11) has to be substituted for

$$\sum_{1 \leq h \leq k} M\mathbf{dlr}(h, l, r) + \left| p_l - t(r) \right| \leq e_r + M_g k, \forall l \in L, r \in R \tag{14}$$

where $p_l \in [LT, UT]$ is the prediction value of leaf l, with LT and UT the minimum and maximum target value, respectively. Furthermore $M = UT - LT$. This constraint forces e_r to be equal to the prediction error of row r: if row r ends up in leaf l, the big-M terms cancel out and e_r is forced to be equal to the prediction error $|p_l - t(r)|$. If row r does not end up in leaf l, no restriction is forced on e_r because of the big-M terms.

Note that in the formulation for regression trees multiple things change compared to the formulation for classification trees. The binary variable p_{ly} changes into a continuous variable p_l since the prediction does not have to be equal to one of the target variables anymore. The prediction error e_r now is the absolute prediction error of row r and thus it is a continuous variable as well. Furthermore, (10), (12) and (13) should be removed in addition to (11), because binary variable p_{ly} is replaced by the continuous variable p_l for which those constraints are not meaningful.

4.2 Reduced Error Pruning

To investigate the presence of overfitting in the classification trees, we split the data into a training set and test set. We evaluate the performance of the created decision trees using the test set and compare those results to the performance on the training set. We then use reduced error pruning (REP) to prune the created trees. We do this in order to evaluate the performance of the MIP model after pruning. Pruning methods discard certain parts of the tree when they are likely to be too specific for the training data. REP is a pruning algorithm proposed by J. R. Quinlan (1987) and is extensively used to avoid overfitting. The idea of REP is fairly simple: after a tree is created, data out of a separate pruning set determines whether certain parts of the tree should be pruned. In a bottom-up procedure it does the following: if changing an internal node into a leaf node does not result in a lower classification accuracy of the pruning set, that node becomes a leaf node and its subtrees are removed from the tree. This avoids the tree being too specific and detailed on the training set and therefore reduces overfitting.

Since the first version of REP leaves room for interpretation, we use the algorithm as proposed by Elomaa and Kaariainen (2001). The "REP" method is shown in Algorithm 1 and calls the "classify" and "prune" methods that are shown in Algorithm 2 and 3, respectively. The "REP" method has a decision tree and the pruning data as input variables, and it returns the pruned tree. First all observations in the pruning data are classified after which the tree gets pruned. In the "classify" method with the root node of a tree as input, an observation traverses to the tree. Depending on the decision rule, it is either moving to the left or right child of the current node. In every reached node the counters "total" and "positive" are updated. The "total" counter represents the number of observations that go through a node, while the "positive" counter represents the number of observations with target value 1 that go through the node, assuming a binary classification model. If there are more than two target values, extra counters should be added to update the number of passing observations for each target value. When all observations are classified, the counters are used to determine whether certain parts of the tree need to be pruned in the "prune" method. This method returns the minimum error for the pruning data for every node in the tree (if the root node is given as input to the method) and prunes the tree in a bottom-up fashion if that is favourable. If the current node is a leaf node, it returns the

error in that node. If the current node is not a leaf node, it compares the sum of the errors of its children with the error the node would have if it would be a leaf. If the latter is smaller than or equal to the sum of errors of its children, the subtrees of the current node are pruned. The current node becomes a leaf with its most favourable label. The label of the new leaf is therefore determined by the pruning data and not by the training data. After pruning the tree the "REP" method returns the pruned tree.

Algorithm 1 REP method

Input: Tree T, pruning data $pruneData$
Output: Tree T after pruning
  for all observations in $pruneData$ **do**
    CLASSIFY($T.rootNode, observation$);
  end for
  PRUNE($T.rootNode$);
  return T;

Algorithm 2 Classify method

Input: Node n, data observation obs
  increment $n.total$;
  if $obs.targetValue$ = 1 **then**
    increment $n.positive$;
  end if
  if n is not a leaf **then**
    if $obs.featureValue \leq n.threshold$ **then**
      CLASSIFY($n.leftChild, obs$);
    else
      CLASSIFY($n.rightChild, obs$);
    end if
  end if

Algorithm 3 Prune method

Input: Node n

Output: Error e of the pruning data in node n, possibly after pruning the subtrees of n

```
if n is a leaf node then
    if n.label = 1 then
        return n.total − n.positive;
    else
        return n.positive;
    end if
else
    error = PRUNE(n.leftChild) + PRUNE(n.rightChild);
    if error < min(n.positive, n.total − n.positive) then
        return error;
    else
        n becomes a leaf, remove subtrees of n from the tree;
        if n.positive > n.total − n.positive then
            n.label = 1;
            return n.total − n.positive;
        else
            n.label = 0;
            return n.positive;
        end if
    end if
end if
```

Note that the REP algorithm requires a pruning set next to the training set and test set, since the pruning set is also used to develop the final tree. The test set is used for classification of the tree created by the training set as well as the pruned tree in order to show the effect of REP. 60 % of the data is used for the training set, while the test set and pruning set both contain 20% of the data. We use stratified sampling to divide the observations to the various sets, meaning that the proportions of the classes are the same in all sets. This is especially of interest for imbalanced datasets, because it ensures there a enough observations of each class in all subsets. A last note about REP is that it does not necessarily prune redundant splits. Since our MIP formulation always creates trees that utilize every possible decision node, it may occur that a tree contains 'fake' splits that are not actually splitting the data. These redundant splits are not necessarily pruned by REP, but for the sake of interpretability one could consider pruning these parts when using the tree in practice.

5 Results

In this section we show the results of the conducted experiments. We use 8 different datasets in total that are all collected from the UCI Machine Learning Repository (Dua and Graff (2017)). All datasets consist of feature variables and one target variable, which is the variable to be

predicted. For instance, the "Diabetes" dataset is used to predict whether someone has diabetes using feature variables such as age and glucose level. To give a wide overview of the performance of the methods, we use datasets with various sizes. For each dataset, the type of tree, number of observations, number of feature variables and number of classes are shown in Table 3. Certain datasets originally have categorical feature variables that cannot be used directly. To solve this, we create dummy variables for all these categories. In the "Bank" data for instance, there is a variable called "Month" which states in which month the customer was in contact with the bank. It does not make sense to map these months to integers since they do not represent an hierarchical order. Therefore we make 12 different dummy variables for the months of the year. This is done for other categorical variables as well.

To compare the MIP formulation with a heuristic, we use CART as a benchmark model. Since CART gives reasonable solutions very quickly, we also use the CART solutions as a warm start for the MIP problem (MIP_WS). We used the CPLEX solver in Java for the MIP problems and the "rpart" package in R for CART. The running times of all instances are limited to 30 minutes. Because the MIP problem gets large very quickly for a larger depth k, we run all models for all depths up to depth $k = 5$. For the classification trees, we compare the methods by their accuracy, while for regression trees we compare the total absolute errors.

Data	**Type**	**Observations**	**Features**	**Classes**
Iris	Classification	150	4	3
Haberman	Classification	306	4	2
Breast	Classification	699	11	2
Blood	Classification	748	5	2
Diabetes	Classification	768	8	2
Authentication	Classification	1371	5	2
Bank	Classification	4521	45	2
Housing	Regression	501	13	-
Red Wine	Regression	1699	11	-

Table 3: Descriptive statistics of the data

We first show the results of the models on the training data in Section 5.1 after which we show the results on the test data and pruned trees in Section 5.2. We conclude this section by giving a visualization of a trained tree before and after pruning in Section 5.3.

5.1 Performance on training data

First we examine the performance of the training set on the MIP formulation. The results for classification trees are shown in Table 4. For the smallest dataset "Iris", the MIP model finds all optimal solutions and therefore outperforms CART. However, the "Bank" data shows that MIP is not always competitive to CART as it clearly exposes the weakness of the MIP formulation: because of the 4521 observations and 45 feature variables, the problem becomes too large to solve to optimality in 30 minutes. This leads to a classification accuracy of only 11.5 % for depth $k = 4$ and $k = 5$, while the method already found trees with an accuracy of 89% for lower depths. The

magnitude of the problem seems to be an issue for MIP_WS as well, as it is performing very similar to CART despite running for 30 minutes. In general, the results MIP and MIP_WS are comparable for depth $k = 1$ and $k = 2$ and they manage to find optimal trees more often than CART. For higher depths MIP_WS slightly outperforms both MIP and CART in most cases.

	$k = 1$	$k = 2$	$k = 3$	$k = 4$	$k = 5$
Iris					
CART	0.667*	0.960*	0.960	0.960	0.960
MIP	0.667*	0.960*	0.993*	1*	1*
MIP_WS	0.667*	0.960*	0.993*	1*	1*
Haberman					
CART	0.742	0.771	0.778	0.794	0.804
MIP	0.758*	0.781*	0.791	0.817	0.859
MIP_WS	0.758*	0.781*	0.797	0.830	0.867
Breast					
CART	0.924	0.950	0.954	0.963	0.963
MIP	0.927*	0.961*	0.973	0.980	0.977
MIP_WS	0.927*	0.961*	0.969	0.977	0.979
Blood					
CART	0.762	0.762	0.795	0.798	0.813
MIP	0.769*	0.781*	0.798	0.799	0.806
MIP_WS	0.769*	0.781*	0.798	0.810	0.825
Diabetes					
CART	0.736	0.772	0.772	0.780	0.819
MIP	0.75*	0.777	0.780	0.762	0.711
MIP_WS	0.75*	0.776	0.792	0.803	0.820
Authentication					
CART	0.853*	0.916	0.938	0.953	0.953
MIP	0.853*	0.927*	0.939	0.970	0.961
MIP_WS	0.853*	0.927*	0.968	0.972	0.962
Bank					
CART	0.893*	0.893	0.893	0.893	0.893
MIP	0.893*	0.893	0.886	0.115	0.115
MIP_WS	0.893*	0.894	0.894	0.894	0.893

Table 4: Accuracy of classification trees with maximum depth k, * denotes an optimal solution

The results for regression trees are shown in Table 5. For both datasets the MIP model finds an optimal solution for k = 1, and performs better than CART until k = 4. After this the problem likely becomes too large to solve as an MIP problem in limited time. For the "Red Wine" data, MIP_WS consistently performs better than both CART and MIP. For the " Housing" data, MIP_WS consistently performs better than CART, while it only gets outperformed by MIP for depth k = 4.

	$k = 1$	$k = 2$	$k = 3$	$k = 4$	$k = 5$
Housing					
CART	2527.9	1797.3	1602.9	1450.4	1450.4
MIP	2499.1*	1743.7	1600.8	1423.4	1622.7
MIP_WS	2499.1*	1743.7	1524.0	1428.4	1430.6
Red Wine					
CART	911.5	898.5	826.0	819.4	803.3
MIP	780*	748	775	809	933
MIP_WS	780*	732	755	761	798

Table 5: Total absolute error of regression trees with maximum depth k, *denotes optimal solution

Because we used exactly the same datasets and similar methods as Verwer and Zhang (2017), we can compare our results to theirs. We see that most of our results of MIP and MIP_WS agree with their outcomes. The accuracies for the "Iris" data are identical, while most results of the other datasets are comparable to their results. The biggest difference is the performance of MIP and MIP_WS for depth $k = 4, 5$, since the accuracy of Verwer and Zhang (2017) in these cases is substantially higher for the "Diabetes" and "Red Wine" dataset, while the absolute error is lower for the "Housing" dataset. Another difference is the performance of the CART model, because their implementation of CART gives a substantially higher accuracy and lower absolute error in especially the "Bank", "Red Wine" and 'Housing" dataset. The new datasets we tested the methods on do not show any new patterns or remarkable outcomes compared to what Verwer and Zhang (2017) have shown before.

5.2 Performance on test data and pruned trees

We continue to investigate the performance of the MIP, MIP_WS and CART model for classification trees. The datasets are now split into three parts: a training set, test set, and pruning set. Table 6 shows the accuracy of each model for depth $k \in \{1, .., 5\}$ for three different cases: "Train" denotes the classification accuracy of the training data, "Test" the accuracy of the test data and "REP" the accuracy after the REP algorithm is applied on the trained tree.

We first examine the difference between the accuracy of the models on the training data compared to the test data. For all models and depths, the accuracy of the tree on the test data typically is lower than the accuracy on the training data. In most cases the difference becomes larger for larger depths. This is the case since at lower depths the tree prioritizes finding the most important splits that often are very relevant for the test data as well. At higher depths however, the tree likely creates some splits to correctly classify just a few specific observations. These splits are not as effective for the test data as for the training data. Another interesting result is that the difference between the training and test accuracy typically is lower for the CART model compared to the MIP and MIP_WS model. The reason for this is that MIP and MIP_WS create trees that utilize every possible split, while CART often creates trees that are much smaller and only have a few decision rules and labels, even at the higher depths. Therefore, these CART trees still capture most of the important splits but do not have as many splits that

are very specific to the training data. This leads to a smaller difference between the training and test accuracy which results to a similar performance of CART on the test data compared to MIP_WS, while the latter outperformed CART on the training data.

The "Haberman" dataset is the clearest example of all those results. Firstly, we see that there is a substantial difference between the training and test accuracy for all models and depths. The magnitude of this difference typically is larger than for the other datasets. This can be explained by the size of the "Haberman" dataset, as it just has 306 observations. This leads to a difference between training and test accuracy that becomes as large as 23.4 percentage points for the MIP tree with depth $k = 5$. Secondly, for MIP and MIP_WS the tree with depth $k = 5$ improves substantially for the training data compared to depth $k = 4$, but the accuracy of the test data stays the same for MIP and only improves slightly for MIP_WS. Lastly, the difference between the test and training accuracies of CART are smaller than these differences for the other models, because even for depth $k = 5$ CART creates a tree with only 3 splits and 4 labels. Although the accuracy on the training data for depth $k = 3, 4, 5$ is larger for MIP and MIP_WS, the accuracy on the test data is larger for CART. This shows the power of small trees relative to new data.

We now investigate the impact of the REP algorithm on the trees. The first thing that stands out is that most trees of depth $k = 2$ and $k = 3$ are pruned, while all trees for depth $k = 4$ and $k = 5$ are pruned (we denote a pruned tree by an asterisk in Table 6). This shows that the original trees often contain parts that are very specific to the training data. These parts can harm the accuracy of new data classified by the tree. Only the trees with depth $k = 1$ are less likely to be pruned, because pruning a tree of depth $k = 1$ always leads to a tree that only contains a single leaf node and no splits. This only is the case if this single leaf node leads to a higher accuracy for the pruning set than the split designed by the model. However, this phenomenon still occurs for all trees with depth $k = 1$ for two out of our six datasets.

When we compare the accuracies of the pruned trees to the accuracies of the trained trees on the test data, there does not seem to be a clear pattern which tree is more accurate for what depths. Note that although the proportions of the target values are the same in the training, test and pruning set, they are only divided once. Therefore we should be reluctant to draw conclusions about the differences in accuracies because of the randomness of the sets. We do see some cases where the accuracy after pruning remains the same while the tree does get pruned (for instance for the "Diabetes" dataset for MIP_WS, $k = 2, 3, 4$). If a tree is used in practice the pruned tree will almost always be preferred over the original tree, since a smaller tree typically is easier to interpret. Another thing to note is that the pruned trees of MIP and MIP_WS can still contain redundant splits. For the sake of interpretability, removing these redundant splits is recommended when using the trees in practice.

	MIP			WS			CART		
	Train	**Test**	**REP**	**Train**	**Test**	**REP**	**Train**	**Test**	**REP**
Haberman									
k = 1	0.765	0.754	0.738*	0.765	0.721	0.738*	0.765	0.721	0.738*
k = 2	0.798	0.705	0.705*	0.798	0.705	0.705*	0.792	0.705	0.705*
k = 3	0.836	0.656	0.705*	0.836	0.639	0.705*	0.798	0.738	0.738*
k = 4	0.850	0.721	0.721*	0.885	0.672	0.689*	0.798	0.738	0.738*
k = 5	0.902	0.721	0.738*	0.923	0.689	0.705*	0.798	0.738	0.738*
Breast									
k = 1	0.921	0.936	0.936	0.921	0.934	0.934	0.921	0.936	0.936
k = 2	0.960	0.943	0.943*	0.960	0.943	0.943*	0.945	0.943	0.943*
k = 3	0.978	0.957	0.964*	0.978	0.950	0.957*	0.950	0.950	0.943*
k = 4	0.976	0.950	0.943*	0.981	0.964	0.950*	0.950	0.950	0.943*
k = 5	0.981	0.950	0.921*	0.988	0.943	0.943*	0.950	0.950	0.943*
Blood									
k = 1	0.766	0.753	0.760*	0.766	0.766	0.760*	0.763	0.760	0.760*
k = 2	0.777	0.780	0.760*	0.777	0.780	0.760*	0.763	0.760	0.760*
k = 3	0.797	0.773	0.773	0.797	0.767	0.767	0.792	0.767	0.767
k = 4	0.797	0.740	0.753*	0.808	0.773	0.773*	0.792	0.767	0.767*
k = 5	0.790	0.747	0.760*	0.841	0.747	0.745*	0.780	0.793	0.767*
Diabetes									
k = 1	0.746	0.780	0.780	0.746	0.714	0.714	0.746	0.714	0.714
k = 2	0.778	0.766	0.766*	0.778	0.766	0.766*	0.770	0.766	0.766*
k = 3	0.791	0.760	0.734*	0.802	0.766	0.766*	0.776	0.773	0.766*
k = 4	0.802	0.701	0.688*	0.835	0.766	0.766*	0.796	0.773	0.766*
k = 5	0.791	0.669	0.669*	0.854	0.786	0.766*	0.809	0.799	0.766*
Authentication									
k = 1	0.858	0.847	0.847	0.858	0.847	0.847	0.858	0.847	0.847
k = 2	0.928	0.901	0.901	0.928	0.901	0.901	0.916	0.920	0.920
k = 3	0.980	0.949	0.949	0.970	0.953	0.953*	0.936	0.927	0.927*
k = 4	0.995	0.974	0.949*	0.993	0.956	0.949*	0.967	0.945	0.934*
k = 5	0.996	0.971	0.938*	0.996	0.964	0.938*	0.967	0.945	0.934*
Bank									
k = 1	0.897	0.884	0.884	0.897	0.884	0.884	0.897	0.884	0.884
k = 2	0.898	0.883	0.883	0.899	0.884	0.884*	0.899	0.884	0.884*
k = 3	0.886	0.887	0.885*	0.900	0.884	0.884*	0.899	0.884	0.884*
k = 4	0.115	0.115	0.885*	0.899	0.885	0.884*	0.899	0.884	0.884*
k = 5	0.117	0.117	0.885*	0.899	0.884	0.884*	0.899	0.884	0.884*

Table 6: Accuracy of the MIP, MIP_WS and CART model on the training data, test data and after pruning. In the "REP" columns, * denotes a pruned tree.

5.3 Visualization of an MIP_WS tree and the impact of REP

We now illustrate the complexity of the trees created by MIP and MIP_WS by looking at an example. In Figure 2 we show the tree created by MIP_WS for depth $k = 4$ for the "Diabetes" data. The numbers represent the stream of observations of the test data that go through the tree. Note that the specifications and size of the test data are equivalent to those of the pruning data, meaning this could have been the pruning data as well. The tree did not have any redundant splits for the training data, but we can see the tree does have five redundant splits for the test data at depth $k = 3$. This shows that the trained tree likely created splits that are very specific to the training data in order to classify a few more observations correctly.

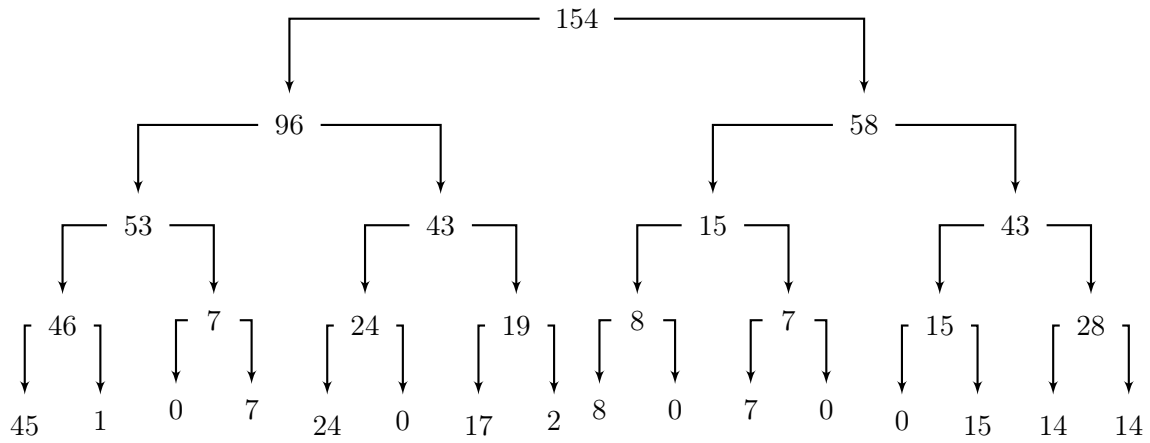

Figure 2: Decision tree trained by MIP_WS for depth $k = 4$. The numbers denote the number of observations of the test data that go through each node.

To illustrate the impact of the REP algorithm, we show the same tree after pruning in Figure 3. The first thing that stands out is the reduction of the size of the tree from 15 decision nodes and 16 leaves, to 4 decision nodes and 5 leaves. The left side of the tree already stops at depth $k = 1$. This means that for the pruning set, turning the left node at $k = 1$ into a leaf generates an error equal to or lower than the error of the original tree. Interestingly, the accuracy of the test data on the original tree is exactly the same as on the pruned tree. This emphasizes that the complexity of the MIP_WS tree does not necessarily improve the classification accuracy of out-of-sample data.

The CART model for the same instance created a tree with 7 decision nodes and 6 leaves. The fat CART generates sparser trees explains why the difference in accuracy of training data and test data generally is lower for CART than for MIP and MIP_WS. As the tree of CART is not too large to begin with, the REP algorithm has a modest impact. It reduces the tree to 6 decision nodes and 5 leaves for this instance.

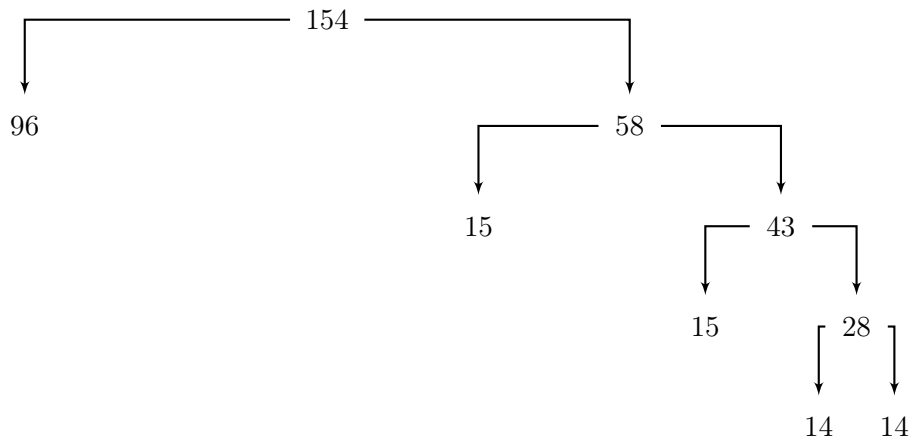

Figure 3: The tree of Figure 2 after Reduced Error Pruning. The numbers denote the number of observations of the test data that go through each node.

6 Conclusion

In this research, we explored the performance of an MIP formulation that tries to create optimal decision trees with depth 1 to 5. We had a particular interest in the performance on data that is not used to train the model, and the impact of the classification accuracy after pruning the trees. We implemented the MIP formulation as proposed by Verwer and Zhang (2017) and compared this MIP model with benchmark model CART, and the MIP model that has the CART solution as warm start (MIP_WS). We first evaluated the accuracy of the trees on the complete dataset for the same datasets used by Verwer and Zhang (2017), as well as for new datasets. We found that MIP_WS typically generates trees with the highest accuracy on the training data, but also see that the MIP formulation performs poorly on a large dataset for depths $k = 4$ and $k = 5$. These results agree with the findings of Verwer and Zhang (2017).

Secondly, we evaluated the performance of the three models on out-of-sample data. We found that the accuracy on the training data is consistently higher than the accuracy on the test data. In particular MIP and MIP_WS are sensitive to overfitting, because they create trees that utilize every possible decision and leaf node. Contrary to that, CART can develop trees with only a few decision nodes and leaf nodes, which makes them less sensitive to overfitting. Because of this, CART does not get outperformed by MIP_WS for the accuracy on out-of-sample data.

Lastly we used the reduced error pruning (REP) algorithm with the goal to improve the accuracy on the test data and prune parts of the trees that are too specific to the training data. The algorithm uses pruning data and prunes subtrees if this would lead to a higher or equal classification accuracy of the pruning set. We found that most of the trees are pruned, but they do not generally increase or decrease the accuracy of the test data. However, for use in practice these pruned trees are preferred over the original trees, since sparser trees typically are easier to interpret.

The experiments we conducted show that the MIP_WS performs best on the training data for sufficiently small datasets, but it does not outperform CART anymore when classifying new data on the trees. After the trees are pruned by the REP algorithm, the size of most trees

decreases but the performance stays relatively similar to before. This means the pruned trees of MIP_WS are not necessarily preferred over the pruned CART trees regarding the classification accuracy of new data. Because CART has a faster running time and creates smaller trees, it actually is more pragmatic and thus generally preferred over MIP_WS for use in practice.

An important note is that the datasets are only divided into the training set, test set and pruning set once. This adds a certain randomness to the results. In future research, cross-validation can be used to lower this randomness. In addition, other pruning methods can be examined to evaluate whether the MIP formulation can outperform CART when classifying new data. Lastly, the performance of this MIP formulation can be compared to other MIP formulations for decision trees in order to get a better insight in the characteristics and pros and cons of attempting to create optimal decision trees.

References

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

Bonner, G. (2001). Decision making for health care professionals: Use of decision trees within the community mental health setting. *Journal of Advanced Nursing*, *35*(3), 349–356.

Breiman, L., Friedman, J., Olshen, R., & Stone, C. (1984). Classification and regression trees–crc press. *Boca Raton, Florida*.

Carrizosa, E., Molero-Rio, C., & Romero Morales, D. (2021). Mathematical optimization in classification and regression trees. *Top*, *29*(1), 5–33.

Dua, D., & Graff, C. (2017). UCI machine learning repository. `http://archive.ics.uci.edu/ml`

Elomaa, T., & Kaariainen, M. (2001). An analysis of reduced error pruning. *Journal of Artificial Intelligence Research*, *15*, 163–187.

Günlük, O., Kalagnanam, J., Li, M., Menickelly, M., & Scheinberg, K. (2021). Optimal decision trees for categorical data via integer programming. *Journal of Global Optimization*, *81*, 233–260.

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

Podgorelec, V., Kokol, P., Stiglic, B., & Rozman, I. (2002). Decision trees: An overview and their use in medicine. *Journal of medical systems*, *26*, 445–463.

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

Quinlan, J. R. (1987). Simplifying decision trees. *International journal of man-machine studies*, *27*(3), 221–234.

Quinlan, J. (1993). C4.5: Programs for machine learning. *Machine learning*, *16*, 235–240.

Shouman, M., Turner, T., & Stocker, R. (2011). Using decision tree for diagnosing heart disease patients. *AusDM*, *11*, 23–30.

Verwer, S., & Zhang, Y. (2017). Learning decision trees with flexible constraints and objectives using integer optimization. *Integration of AI and OR Techniques in Constraint Programming: 14th International Conference, CPAIOR 2017, Padua, Italy, June 5-8, 2017, Proceedings 14*, 94–103.

Verwer, S., & Zhang, Y. (2019). Learning optimal classification trees using a binary linear program formulation. *Proceedings of the AAAI conference on artificial intelligence*, *33*(01), 1625–1632.

Zhu, H., Murali, P., Phan, D., Nguyen, L., & Kalagnanam, J. (2020). A scalable mip-based method for learning optimal multivariate decision trees. *Advances in neural information processing systems*, *33*, 1771–1781.

A Programming code

We now give a brief summary of our code. In java, we made a *DecisionTrees* class to solve the Cplex model for classification trees and a *RegressionTree* class to do the same for the regression trees. The input is an excel file that consists of the integer data (after Step 1 of the data transformation). The last transformations are done in java itself. These two classes use the *Node* class in one of its functions. The *DoubleRegressionTree* class is the same as the *RegressionTree* class but allows for non integer target values.

The *dataTransformo* class has again the integers of the data as input and writes an excel file with the fully transformed data. It now can be used in R for the runs of the CART model.

In R we have the Cart_R script for CART on the regression trees with the transformed data as input data. We use the CART_regression script for regression trees.

The *dataPartition* class in java divides the data in the training, test en pruning set. It writes an excel file of the training set so this can be used in the CART algorithm in R. The classification of the test data on the trained cart tree actually is done in java at the same time the cart tree is pruned.

The *Tree* class in java represents a tree, while the *NodeOfTree* class represents a node in the tree. The *pruningDecisionTrees* class and *PrunedCart* class are used to prune the trees created by the MIP formulation and CART, respectively. When using CART as a warm start or when the CART trees are pruned, the CART trees are manually implemented in the java code.