Erasmus University Rotterdam
Erasmus School of Economics
Bachelor Thesis Econometrie en operations research

Bachelor Thesis

Frank van der Duijn Schouten (482507)

Supervisor:	Badenbroek, RM
Second assessor:	Akyuz, MH
Date final version:	28-6-2023

Abstract

In this paper we analyze if incorporating multiple features in a single node of a decision tree has an effect on the performance (in terms of in sample classification accuracy) of decision trees under time constraints. We use trees as found by CART as warm starts for the solver and analyse if either the univariate or multivariate decision tree performs better. We find that for small datasets having multiple features in a node has a slight positive effect on the classification accuracy. For medium sized and larger datasets we find that using multiple features in a node gives even or worse results. We conclude that making decisions on multiple features in a node does not increase the speed at which the solver improves trees found by a heuristic.

1 Introduction

Decision trees are rapidly increasing in popularity due to the rise of machine learning. Due to the fact that learning optimal decision trees is NP-complete [5], heuristics such as CART [2] and ID3 [7] are very popular in finding suboptimal trees in good time. A tree is called suboptimal if there exists a tree which has a higher classification accuracy for the same data and tree depth. To instead find optimal trees [8] and [1] formulate mixed integer optimization programs. The MIO program in [8] finds good results in reasonable time for small to medium sized data sets but, when the number of features and data points gets too big the program in and of itself produces poor results in a restricted time frame. The authors of [1] show that their formulation almost always outperforms heuristic methods on self-generated trees and outperforms heuristic methods in most cases on real life data. The authors do mention that while for most cases the solving time was limited to 30 minutes, for some of the larger data sets this time was extended to 2 hours since the solver could not produce good results otherwise.

While [8] find that their program performs poorly for larger data sets, that is trees produced in a restricted time frame of 30 minutes are significantly worse than trees produced by CART, the authors do show that the program can improve trees found by CART.

Both heuristics and the MIO formulation in [8] are restricted by the fact that they use single variate decision making (ie, at every node only a single feature value is assessed). However, since cross-information between certain feature values could contain very valuable information for classification, this restriction may be very much undesirable. The chance that certain features hold more information when combined grows with the number of features. This is counterbalanced by the fact that the number of combinations also grows with the number of features. This leads to the possibility that the desire to incorporate these combinations when number of features grows, is balanced by the increase in computation time this would involve.

Since MIO formulations appear to perform poorly on larger data sets, as shown above, and thus, for larger data sets, seem to have their best use in trying to improve trees as found by a heuristic, the focus of the MIO formulation is easily shifted in this direction. When looking to improve a tree as found by CART it is reasonable to direct attention to using information which is not accounted for yet. [3] and [1] find that allowing multivariate splits generally results in higher accuracy trees.

This is not entirely unexpected since the combination of multiple distinct features could be a very clear indication of a certain class, and this is not used in the CART algorithm. An obvious idea to improve trees as found by this algorithm is to incorporate multivariate decision making in the MIO formulation. Because we would use the tree as found by CART as our starting point, and this requires a lot less computation from a solver, the balance between increase in computation time and chance of increase in information, as mentioned above, could very well shift to the point where incorporating combinations of feature values could lead to a faster improvement of the trees as found by a heuristic.

The improvement of trees using multivariate decision making comes at a cost of interpretability, since splitting on a multitude of values is hard to interpret. To balance interpretability and optimality of a tree it seems reasonable to restrict the number of features used per node. In other words we would like to be able to limit the number of dimensions of every decision made in the tree. While [1] shows that their MIO formulation can be rewritten for the case of multivariate decision making their formulation does not allow for the restriction of the number of dimensions of the decision, such that to our knowledge we are the first to investigate this case.

In this paper we investigate if incorporating multi variate decision making in the mixed integer formulation of a decision tree, while limiting the dimensions of the decisions, results in improving trees, as found by a heuristic, faster than single variate decision making.

Note that since [6] notes that C4.5 and CART gave close to identical results, in this analysis it seems unnecessary to use both and therefore we will only use CART.

In section 2 we will construct an MIO program where the number of feature values is a free variable which can be filled in later. In section 3 we experiment with the formulation on datasets from the UCI machine learning repository and analyze these results. Finally in section 4 we summarize and conclude our findings.

2 Formulating an MIO program

To start modelling the decision we tree we begin by defining the tree itself. We use the model as described in [8], with some slight adjustments. While [8] denotes their process of modelling the decision tree, for clarification the formulation will be explained in detail below as well.

2.1 Decision trees

In this research we analyze decision trees, if the reader is not familiar with this concept we will explain the concept here briefly. If the reader is familiar this section can be skipped.

A decision tree consists of a number of nodes, every node has (in this case) either 0 or 2 children. When a node has children we consider this a branch or internal node. When a node does not have children we consider this a leaf or external node. Every internal node of a decision tree contains a threshold and information about which variable it uses. Each observation in our data consists of multiple variables and a class. For each observation in each node of the decision tree we compare the value of the variable specified in the node with the threshold of the node. If this value is less than or equal to the threshold we send our observation to the left. If the value is higher than the threshold we send our observation to the right. We continue through this process until an observation reaches an external node. Every external node of a decision tree contains a single class. Finally we compare the class of the observation to the class of the external node where it ends up. If the classes are the same the classification is correct, if the classes are not the same the classification is incorrect. The goal of a decision tree is to maximize the percentage of observations classified correctly.

2.2 Data transformation

The MIO program in [8] maps all feature values to integers. They do this because they argue that this makes the solver faster, which makes it very reasonable. In this formulation however we will use a multivariate split on basis of a linear constraint of 2 different feature values. Because of this mapping the feature values to integers would not result in a decision tree as desired, since a linear constraint does not work properly when the mapping of feature values is non-linear. Therefore in this research we linearly map our feature values to [0,1]. This allows us to use a linear combination of feature values and as a bonus does not require us to use a big-M formulation.

The bank database contains certain categorical feature values. In this analysis every category is assigned a numerical value between 0 and 1 and treated as such. We realise that [8] creates dummy variables for each category and that this might lead to some slightly different results.

2.3 Mixed integer formulation

Now that our data is defined on a set range we can start modeling the tree. For every node k in the tree we need a decision variable $c_k \in [0,1]$ which is the threshold on basis of which we decide to go left or right.

Second we need to keep track of which feature value is used in the decision of each node. We do this by introducing binary variable

$$f_{i,k} \in \mathbb{B} \quad \forall i, \forall k$$

which is 1 if and only if feature i is used in the decision of node k. We need to model that only one feature is used in every node, by setting

$$\sum_{i=1}^{m} f_{i,k} = 1 \quad \forall k$$

we achieve this. Next we model if data row r takes a left or right at node k by means of binary variable $d_{h,r}$. In this formulation $d_{h,r}$ takes value 1 if the feature value $v(r,i)$ is less than the threshold value c_k (from here also written as going left), and value 0 if it is more than the threshold value (from here also written as going right). Let m be the number of variables per observation then

$$c_k - \sum_{i=1}^{m} v(r,i) * f_{i,k} + 1 \geq d_{h,r} \quad \forall k, \forall r \tag{1}$$

and

$$c_k - \sum_{i=1}^{m} v(r,i) * f_{i,k} \leq d_{h,r} \quad \forall k, \forall r \tag{2}$$

model our decision-making per node. Note that formulations above hold specifically since $v(r,i)$ is always between 0 and 1.

We only need these restrictions to hold if the mapping of row r actually goes through node k. To do this we define a new variable: $path(h,k,r)$. This variable is equal to $d_{h,r}$ if the path to node k goes left at depth h and is equal to $1 - d_{h,r}$ if the path to node k goes right at depth h. What this does is essentially give value 1 to the variable $path(h,k,r)$ if the mapping of datarow r goes in the direction of node k at depth h and 0 otherwise. Adding this to the restrictions makes sure that the $d_{h,r}$ variable is free when the datarow does not pass through node k. Adding this to equations 1 and 2 and rewriting to omit subtractions gives:

$$depth(k) + c_k + 1 \geq d_{depth(k),r} + \sum_{h=0}^{depth(k)-1} path(h,k,r) + \sum_{i=1}^{m} v(r,i) * f_{i,k} \quad \forall k, \forall r \tag{3}$$

$$\sum_{h=0}^{depth(k)-1} path(h,k,r) + c_k \leq d_{depth(k),r} + \sum_{i=1}^{m} v(r,i) * f_{i,k} + depth(k) \quad \forall k, \forall r \tag{4}$$

It must be noted here that when the feature value is exactly equal to the threshhold of a node, that is: $\sum_{i=1}^{m} v(r,i) * f_{i,k} = c_k$, then the $d_{depth(k),r}$ variable can take either value 0 or 1. The solver could use this to select a certain threshold to be equal to feature values such that different datarows with the same feature value could be sent in different directions from the same node. To omit this problem we add a small constant δ which forces the $d_{depth(k),r}$ variable to be 1 if the values are equal. Adding this to constraint 4 gives :

$$\sum_{h=0}^{depth(k)-1} path(h,k,r) + c_k + \delta \leq d_{depth(k),r} + \sum_{i=1}^{m} v(r,i) * f_{i,k} + depth(k) \quad \forall k, \forall r \tag{5}$$

We need to select our variable δ such that it can not influence other decisions, in other words we need it to be smaller than the smallest difference between feature values of different observations. In implementation we found the lowest difference in feature values and set our δ parameter to be 95% of this.

Now that we defined our nodes and datarow mappings we focus on the classification errors. We define the classification error as a binary variable and our objective function as minimizing the sum of the classification errors. To begin we define for every data row a binary variable

$$e_r \in \mathbb{B}$$

which is 0 if the row was classified correctly and 1 otherwise. We aim to minimize the sum of the classification errors by

$$\min \sum_{r=1}^{n} e_r$$

To establish the classification errors we first define predictors for every individual leaf. We define

$$p_{l,t} \in \mathbb{B},$$

which is 1 if the prediction of leaf l is class t and 0 otherwise. We set

$$\sum_{t=1}^{g} p_{l,t} = 1 \quad \forall l$$

to model that every leaf has exactly one prediction:

Let $t(r)$ denote the actual class of row r. Now we need that if the classification is wrong we add 1 to our error total.

$$\sum_{t \neq t(r)} p_{l,t} \leq e_r \quad \forall r, \forall l \tag{6}$$

We do of course need that the error total is only updated if row r actually ends in leaf l. We can use the path variable again. With y denoting the depth of the tree we add this to equation 6 giving:

$$\sum_{h=1}^{y-1} path(h,l,r) - y + 1 + \sum_{t \neq t(r)} p_{l,t} \leq e_r \quad \forall r, \forall l$$

2.4 Multivariate decisions

To allow for multivariate decisions we use a technique as shown in [1]. We relax the restriction that $f_{i,k}$ is binary and instead let it range between -1 and 1. This allows us to have a value for $f_{i,k}$ that is nonzero for multiple values of i in the same node. To track the number of features used in each node we introduce a new binary variable $s_{i,k}$ which is 1 if and only if feature i is used in node k. We make sure this holds by the following restrictions.

$$s_{i,k} \leq f_{i,k} \quad \forall i, \forall k$$

$$s_{i,k} \geq -f_{i,k} \quad \forall i, \forall k$$

To limit the number of features used in each node to Q, we impose the following restriction.

$$\sum_{i=1}^{m} s_{i,k} \leq Q \quad \forall k$$

Finally combining all of the above we get the following MIP:

$$
\begin{aligned}
& min \sum_{r=1}^{n} e_r && && (7)\\
\textbf{subject to: } & \sum_{i=1}^{m} f_{i,k} = 1 && \forall k && (8)\\
& depth(k) + c_k + 1 \geq d_{depth(k),r} + \sum_{h=0}^{depth(k)-1} path(h,k,r) + \sum_{i=1}^{m} v(r,i) * f_{i,k} && \forall k, \forall r && (9)\\
& \sum_{h=0}^{depth(k)-1} path(h,k,r) + c_k + \delta \leq d_{depth(k),r} + \sum_{i=1}^{m} v(r,i) * f_{i,k} + depth(k) && \forall k, \forall r && (10)\\
& \sum_{t=1}^{g} p_{l,t} = 1 && \forall l && (11)\\
& \sum_{h=0}^{y-1} path(h,l,r) + \sum_{t \neq t(r)} p_{l,t} \leq e_r + y && \forall r, \forall l && (12)\\
& s_{i,k} \geq f_{i,k} && \forall i, \forall k && (13)\\
& - s_{i,k} \leq f_{i,k} && \forall i, \forall k && (14)\\
& \sum_{i=1}^{m} s_{i,k} \leq Q && \forall k && (15)\\
& f_{i,k} \in [-1,1] && && (16)\\
& c_k \in [0,1] && && (17)\\
& d_{h,r} \in \mathbb{B} && && (18)\\
& p_{l,t} \in \mathbb{B} && && (19)\\
& e_r \in \mathbb{B} && && (20)\\
& s_{i,k} \in \mathbb{B} && && (21)
\end{aligned}
$$

We use the model as defined here in the multivariate decisions section for modelling both the case where we analyse only one feature per node and the case where we analyse 2 features per node. This is because we analyze the effect of having a different number of feature values in a node and using a different formulation could skew the results. This means that for the case where we use just one feature value in a node we simply use the model stated above and set Q to 1.

In implementation we set the preciseness of some aspects of the solver to non-standard values. We needed to do this since some differences in observations were really small and we had some problems especially with the $d_{h,r}$ variable. Since this greatly influenced some classification accuracy's this problem had to be addressed. While changing these settings might not seem like a big problem, this might impact the speed of the solver.

3 Computational experiments

We tested our program on several datasets from the UCI machine learning repository [4]. Specifically we tested on the Iris, Red wine and Bank datasets. The Iris dataset consists of 150 observations of 4 features, mapping to 3 different classes. The Red wine dataset is a portion of the wine dataset, consisting of 1599 observations of 11 features and an integer score between 1 and 10 of how good the wine is. Finally the Bank dataset consists of 4521 observations of 16 features values and a binary target variable. Since we are trying to find the optimal trees in this paper we use all data to construct the best tree, i.e. there is no performance evaluation out of sample.

The implementation of the CART algorithm was done by the scikit-learn package in Python (sklearn). The trees found by CART were manually transferred to the Java environment where they were used as warm starts for the solver.

For ease of notation from now on we will refer to the program with a maximum of Q features per node as Solver_Q.

The time limit for each solver was set to 30 minutes and all experiments were performed with a Ryzen 7 3800 CPU and 32 GB RAM. We used CPLEX 22.1.1 as our solver.

3.1 Results

Table 1 shows the results from the experiments.

Table 1: Classification accuracy

		d=1	d=2	d=3	d=4	d=5
Iris	Cart	0.667*	0.96*	0.973	0.993	1*
	Solver_1	0.667*	0.96*	0.993*	1*	1*
	Solver_2	0.667**	0.98**	1**	1**	1**
Red wine	Cart	0.561*	0.561	0.576	0.630	0.665
	Solver_1	0.561*	0.583	0.607	0.637	0.665
	Solver_2	0.574	0.564	0.580	0.633	0.665
Bank	Cart	0.885	0.901	0.904	0.912	0.918
	Solver_1	0.893*	0.901	0.905	0.912	0.919
	Solver_2	0.898	0.901	0.905	0.912	0.919

* Indicates that this is the optimal value when restricted to 1 feature per node.
** Indicates that this is the optimal value when restricted to 2 features per node.

3.2 Comparison to previous results

We first compare our found results to the results as found in [8]. As we use the same datasets we expect to get at least very comparable and in some cases identical results.

For the Iris dataset we see that both the CART algorithm and Solver_1 produce identical results as in [8]. This is of course expected since the find optimal results and a slight change in formulation should not have changed the optimal solution.

For the Red Wine dataset we run into the slight problem that [8] uses a regression tree to indicate a score for each datarow while we use a classification. However when we scale the numbers found we see that results are very comparable in increase in accuracy when increasing depth. This, combined with the fact that the solver only find an optimal solution for depth 1, shows that output for this dataset is not much different from previous results. The only notable difference is that our formulation appears to find slight improvements for depth 2,3 and 4 while [8] does not find any or very small ones. This is most likely due to the fact that we use a classification tree instead of a regression tree.

Comparing results for the Bank dataset show that CART gives slightly better results in [8] for depth 5. This is almost certainly due to the fact that they use dummy variables for categorical variables while we do not. This appears to be the only notable difference in results. CART and Solver$_1$ give near identical results for every depth and only for depth 1 do both solvers state that it found the optimal solution. ([8] states that for depth 2 CART is optimal but they do not state this for their solver which has the same classification accuracy. We suspect this is a typo of some sorts, or this is the optimal solution however the solver can not say this with certainty within the time frame). Both their and our solver find the optimal solution for depth 1 and no improvements at depth 2. Both their and our solver find a very small improvement for depth 3. Finally for depth 4 and 5 we find no improvement while [8] again finds some very small improvements. This difference is most probably an effect of a different treatment of categorical variables and data transformation.

It seems clear that the results we obtained are very similar to those reported in [8]. While some previously found results are slightly better, because the size of the difference is so small we can say with confidence that above mention of non-ideal settings for the solver and a slightly less optimal data-transformation has had a minimal effect on the results in our research.

3.3 Comparing univariate and multivariate trees

Table 2: Running times for Iris dataset (ms)

	d=1	d=2	d=3	d=4	d=5
Solver$_1$	300	1129	4468	6850	10364
Solver$_2$	356	1300	4827	8496	13554

In the Iris dataset we observe that for each depth optimal solutions are reached. We of course expect Solver$_2$ to always outperform Solver$_1$ because restrictions for Solver$_2$ are less strict. This does make it hard to analyze the solvers performance under time constraint. Therefore we ran all experiments for the Iris database again and collected the time needed for the solver. Results for this can be found in table 2. We see that Solver$_2$ always requires a bit more time which is somewhat expected since the program is bigger. Increase in computation time does not appear to big for almost every depth. We see the biggest relative increase for depth 5 which points to the conclusion that Solver$_2$ is worse for bigger programs.

For the Red Wine dataset we first observe that none of the solvers find any improvements for depth 5. Secondly we see that Solver$_1$ outperforms Solver$_2$ at depth 2, 3 and 4. This indicates that for medium sized programs where the solver is able to find an improvement but no optimal solution Solver$_1$ performs better. For depth = 1 we observe that this program is small enough again such that for depth 1 Solver$_1$ finds an optimal solution but Solver$_2$ performs slightly better. This indicates that for the smaller programs the relaxation that Solver$_2$ brings with it outweighs the extra computation time.

The Bank dataset appears to be simply too big for the program as none of the solvers find any improvement greater than 0.001 for depth 2 and higher. This shows that the Bank dataset is too big to analyze for the program in 30 minutes with our hardware setup. Again for depth 1 Solver$_2$ performs slightly better than Solver$_1$ which finds the optimal solution.

From the results above it seems rather clear that incorporating multiple features in a node does not improve trees found by CART faster then single feature decision-making. For all of the datasets and depths if Solver$_2$ does not find an optimal solution it always performs worse than Solver$_1$.

From the classification accuracy's of the Iris database and the Red Wine and Bank database on depth 1 we can conclude that incorporating multiple features in a single node can lead to higher classification accuracies when the time constraint is not of essence. This is of course expected since we are relaxing restrictions.

We can see from the running times of the Iris database that Solver$_2$ starts becoming a bit slower than Solver$_1$ when the size of the program increases. From the results of the Red Wine and Bank databases we can conclude that when the databases become larger, the relaxations that Solver$_2$ enables do not weigh up against the increase in program size this brings with it in terms of performing under time constraints.

It is also noteworthy that the formulation of Solver_1 works the same but is computationally slightly worse than the formulation and data-transformation as in [8]. The authors themselves explain why their formulation is better so we refer to their paper for an explanation. Since this formulation is computationally worse and still outperforming Solver_2 it seems very clear that the use of the MIP as described in [8] almost always has the preference over Solver_2.

4 Conclusion

We analyzed what relaxing the number of features in a node of a decision tree does to the improvement of the CART heuristic under time-constraints. Results show that when using an MIP to establish a decision tree, allowing the tree to split on multiple, in our case 2, features at the same node, does not improve trees, as found by a heuristic, faster than restricting the number of features per node to 1. Since allowing for multiple features in a node is still a relaxation, for depths where the solver is able to find optimal solutions in the time frame allowing for multivariate splits can still give a good improvement of the classification.

Since it is not entirely clear to what extend Solver_2 is slower than Solver_1 and we have no Solver_Q for $q \geq 3$, for future research it seems interesting to look at classification accuracy's and running times for different solvers (that is different values of Q) and different sized databases, if we let the solver run to optimality. This way some benchmarks could be set in literature for which version and variation of the version of the general program that is described in [8], [1] and this paper, is best used for different sized datasets.

References

[1] Dimitris Bertsimas and Jack Dunn. "Optimal classification trees". In: *Machine Learning* 106 (July 2017). DOI: 10.1007/s10994-017-5633-9.

[2] L. Breiman et al. *Classification and Regression Trees.* 1984.

[3] Carla Brodley and Paul Utgoff. "Multivariate Decision Trees". In: *Machine Learning* 19 (Apr. 1995), pp. 45–77. DOI: 10.1023/A:1022607123649.

[4] Dheeru Dua and Casey Graff. *UCI Machine Learning Repository.* 2017. URL: http://archive.ics.uci.edu/ml.

[5] Laurent Hyafil and Ronald L. Rivest. "Constructing optimal binary decision trees is NP-complete". In: *Information Processing Letters* 5.1 (1976), pp. 15–17. ISSN: 0020-0190. DOI: https://doi.org/10.1016/0020-0190(76)90095-8.

[6] Sreerama Murthy and Steven Salzberg. "Decision Tree Induction: How Effective is the Greedy Heuristic?" In: Proceedings of the First International Conference on Knowledge Discovery and Data Mining (1995), pp. 222–227.

[7] J. R. Quinlan. "Induction of decision trees". In: *Machine Learning* 1.1 (Mar. 1986), pp. 81–106. ISSN: 1573-0565. DOI: `10.1007/BF00116251`.

[8] Sicco Verwer and Yingqian Zhang. "Learning Decision Trees with Flexible Constraints and Objectives Using Integer Optimization". In: (2017). Ed. by Domenico Salvagnin and Michele Lombardi, pp. 94–103.