Delft University of Technology, Bachelor Seminar of Computer Science and Engineering

Devising Multivariate Splits for Optimal Decision Trees

Abele Mălan
Delft University of Technology
Supervisor: Emir Demirović

Abstract

Decision trees are often desirable for classification/regression tasks thanks to their human-friendly models. Unfortunately, the construction of decision trees is a hard problem which usually implies having to rely on imperfect heuristic methods. Advancements in algorithmics and hardware processing power have rendered globally optimal trees accessible, but even state-of-the-art techniques remain limited in certain aspects. One such limitation is the use of simple, univariate predicates within the decision tree. This paper explores the practicality of further expanding the already exponentially large search space navigated by existing optimal tree algorithms in order to add support for more complex, multivariate predicates. While such an approach comes with profound negative implications on the algorithm's time complexity, it has the potential to create decision trees whose boundaries are closer to ground truth. The results of this work support such a hypothesis, but they also make it clear that further optimisation techniques are critical for multivariate optimal decision tree algorithms, especially as the variety of considered multivariate splits increases.

1 Introduction

Decision tree learning is a predictive modelling approach that draws conclusions about an item's target value from observations about its properties [1] by using a flowchart-like structure [2]. Starting with the root, each node represents a question about the data, and edges to subsequent nodes are possible answers. Such a chain of internal nodes ends with a leaf that contains a target value. An example is shown in Figure 1. The importance of decision trees in the fields of machine learning and data mining stems from their concise and intelligible models, unmatched by other black-box techniques.

Because constructing decision trees is an NP-Complete problem [3], practical implementations, like CART [4], are based on heuristics that optimise decisions at the level of individual nodes instead of the entire tree. This has the possibility of negatively affecting accuracy or compactness (i.e. the tree's size). Furthermore, a heuristic approach may not be robust enough to ensure fairness in socially sensitive contexts [5]. Improvements in computational power and algorithmic techniques have recently sparked renowned interest in the construction of optimal decision trees that address the abovementioned concerns.

Modern optimal decision tree algorithms propose a variety of different strategies. DL8.5 [6] is such an algorithm that takes a dynamic-programming (DP) approach and makes use of item-set mining techniques to radically improve performance over its predecessor DL8 [7]. The current state-of-the-art is, however, represented by MurTree [8], another DP algorithm that provides speedups worth several orders of magnitude over even DL8.5. It creates binary

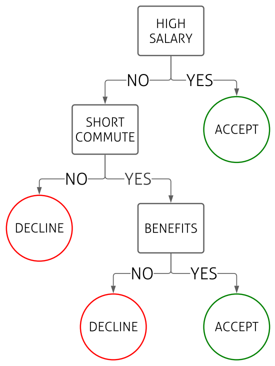


Figure 1: Sample decision tree for declining/accepting a job offer

optimal decision trees from a binary data set by employing "a series of specialised techniques that exploit properties unique to classification trees".

Nevertheless, the research regarding optimal decision trees is vastly underdeveloped compared to that of heuristic ones. It is worth asking whether algorithms like MurTree can achieve a further boost by exploiting parallel computing, or whether different ways of encoding the input can provide further advantages, but more generally, many of the questions or variations which have been applied to heuristic trees over the years are worth exploring in the setting of optimal ones as well.

This work investigates the feasibility of building multivariate optimal decision trees as a proof-of-concept extension to the MurTree algorithm and compares the results to the existing version. While most decision trees "ask" questions about one data feature at each internal node, a multivariate decision tree considers some combination of the features. Multivariate (or non-axis-aligned) decision trees are a generalised version of their orthogonal counterparts (also known as axis-aligned or univariate). Oblique trees, the most popular kind of multivariate tree, express their questions as a strictly linear feature combination. While the terms oblique and multivariate often get used interchangeably, a distinction between them exists, with the latter also enclosing non-linear feature combinations [9].

Heuristic versions of multivariate, usually oblique, trees already do exist. HHCART [10] is a notable example that has shown considerable improvements in minimising the generated tree's dimensions and improving accuracy when the decision boundaries of the data are not axis-aligned. Given the minimal amount of existing work regarding multivariate optimal trees, the hypothesis that a multivariate MurTree variant could provide tighter/simpler decision bounds depending on the nature of the data is inferred from heuristic experiments. Consequently, the project's goal is to create more accurate trees by using multivariate splits in addition to univariate ones. A multivariate extension whose trees always have a lower or, at worst, equal misclassification rate on the input data than the original version represents a satisfying answer to the research question.

Thus, the stated research question can be split into the following sub-questions:

1. What are the main principles behind existing tree algorithms?
2. How does the MurTree algorithm work at its core?
3. How could the MurTree algorithm be extended to produce multivariate trees?

4. What is the difference between the multivariate and univariate algorithms in terms of training time and classification accuracy?

The rest of the paper is structured into five main parts. The terminology section provides a more comprehensive description of decision trees and related terms frequently used throughout the text. Following that, a literature study surveys some relevant existing works on decision trees, focusing on those that cover multivariate cuts and global optimality. The preliminaries essential to formulating the contributions of the paper are then laid out. Subsequently, these contributions are presented in detail and followed up by a final experimental section that empirically evaluates the resulting algorithm to answer the proposed research question. Lastly, the conclusion includes ideas for further investigation, and a discussion on responsible research follows after it.

2 Terminology

As a primer for the literature study and the rest of the paper, this short section offers a deeper explanation for fundamental decision tree terms/concepts also showcased in the introduction.

The tree's leaves are known as classification nodes because each embodies an assignable target class. The other nodes are called decision or predicate nodes, and they guide items node-by-node towards a leaf based on the value of a statement about the item's features.

Both algorithm properties key to this work relate to predicates, but they are independent of each other. One property looks at the types of predicates allowed in the tree, while another looks at the criteria which deem a predicate the best fit for a given node.

2.1 Predicate Types

As previously mentioned, univariate predicates, as the simplest and most common type, construct a statement that looks at only one feature of the object. Depending on the feature's domain and the allowed types of conditions, the predicate could be asking whether it has a specific value, crosses a chosen threshold, or in the case of binary features, simply if it exists. Univariate predicate nodes can also be referred to as feature nodes.

Oblique predicates are simultaneously the most common subtype of multivariate predicates and a generalisation of their univariate siblings. They are generally devised by multiplying each possible feature with a coefficient, adding the results and comparing the final value to a reference value. Note that such a representation assumes feature values can be expressed numerically, but alternatives do exist. Setting the coefficient for a single feature to one while keeping all others at zero emulates a univariate predicate. The following is an example formulation of an oblique predicate over f features, where c_i is the coefficient applied to the i-th feature, x_i is the value of the i-th feature, and v is the reference value:

$$\sum_{n=1}^{f} c_i x_i > v \tag{1}$$

Multivariate predicates, as a whole, encompass even further generalisations than purely oblique ones, but they further increase the number of predicates to consider.

Decision node predicates hence split the training data into multiple non-overlapping subsets. In the following, when mentioning a tree with depth d or n nodes, the restrictions apply exclusively to predicate nodes.

2.2 Predicate Selection

Based on how predicates are assigned to nodes, two main types of decision tree algorithms are distinguished: heuristic and optimal.

Heuristic algorithms choose predicates that greedily maximise the goodness of a split at a local level based on the heuristic's value. Optimal algorithms perform an exhaustive search that directly maximises the final accuracy metric on the training data. Thus, unlike heuristic algorithms, optimal ones make seemingly suboptimal decisions locally when they allow for the most accurate model holistically.

3 Literature Study

This section serves as an outline of some noteworthy approaches, and results attained by different works concerning decision trees. It contains four subsections, obtained as a result of classifying decision trees based on their optimality, along with the type of cuts exhibited within their decision nodes. The discussion around multivariate trees focuses on the oblique variant specifically, as it is by far the most commonly explored one in literature.

3.1 Heuristic Orthogonal Trees

When it comes to heuristic orthogonal trees, two of the most popular algorithms are CART (Classification And Regression Trees) [4] and C4.5 [11]. It is worth noting that C4.5 is "a descendant of CLS and ID3", and, even though it remains a reference version within its family, an updated version called C5 does exist [12]. While the two algorithms have commonalities at a high level, some differences are worth pointing out:

- CART constructs binary decision trees (i.e. predicates are formulated as yes/no questions), while C4.5 supports splits with multiple outcomes, often referred to as multi-way;
- CART ranks predicates by the Gini diversity index, while C4.5 uses criteria based on split information;
- with regard to pruning, CART uses a cost-complexity model with parameters estimated through cross-validation, while C4.5 derives a single-pass algorithm from binomial confidence limits [12].

3.2 Heuristic Oblique Trees

Many different approaches to building heuristic oblique trees have been considered over the years. This subsection will briefly introduce a couple of the representatives that have stood the test of time, alongside a few more recent entries that aim to improve upon their forerunners' efforts.

CART-LC (Cart Linear Combinations) is an alteration of CART which was proposed alongside the classic algorithm and makes use of splits with linear combinations [4]. OC1 follows up on CART-LC by attempting to escape local minima through random restarts and perturbations [9].

HHCART is part of the modern family of oblique heuristic algorithms. As implied by the first two letters in the name, Householder transformations play an integral role in the algorithm. A Householder transformation describes, as a matrix, a linear transformation for reflecting an origin about a plane or hyperplane [13]. During the build process, training data gets reflected on a node-by-node basis before a predicate is elected. Orthogonal splits are performed on the reflected data to give an efficient way of finding oblique splits in the unreflected, original domain [10].

CO2 was created with the scope of improving upon univariate random forests by exploiting obliqueness in the individual trees within the forest. It optimises the weights of features within the split by "adopting a variant of latent variable SVM [Support Vector Machine] formulation". A continuous upper bound on classification loss is developed and optimised by stochastic gradient descent for each decision point [14].

Most recently, the Weighted Obliques Decision Trees (WODT) algorithm also relies on a continuous optimization component, accompanied by random initialisation of the weights. Splits are obtained through optimisation of "the continuous and differentiable objective function of weighted information entropy", and results proved to be among the best when compared to previous attempts in the same area of work [15].

3.3 Optimal Orthogonal Trees

Optimal decision trees, as a whole, have enjoyed a bit of a resurgence recently. A few different approaches are currently favoured, and this subsection will explore two of them, with both usually fixing at least the depth of the tree in advance. One technique takes advantage of some form of Integer Linear Programming (ILP). Such methods represent node predicates as variables and then add restrictions on these variables to enforce the structure of a decision tree. Another approach replaces this declarative structure with a more specialised one, trading some degree of flexibility in adding extra constraints for better scalability with regard to the number of features and instances present in the data set [8].

The current bleeding edge for ILP optimal trees is given by a flow-based formulation that builds binary classification trees. This encoding is exploited to solve a max-flow min-cut problem that scales better with increasing feature counts [16].

On the side of specialised algorithms, as mentioned in the introduction, DL8.5 is an important representative. A Python interface has even been created for the algorithm, under the form of the PyDL8.5 library [17]. It uses a dynamic programming formulation with a strong emphasis on caching as memoisation, as its precursor, DL8. A cache of the paths in the decision tree (referred to as item-sets) is combined with a branch-and-bound search to find the globally optimal solution [6].

MurTree, also mentioned in the introduction, creates optimal binary trees from binary features, and achieves further performance improvements over DL8.5. It provides support for limiting not only the depth of the tree, but also the amount of feature nodes. It also incorporates a novel sub-algorithm for computing trees of depth two, along with a similarity-based lower bound technique. As for caching, it provides two different approaches: one branch-based, not too dissimilar from that in DL8.5, and a more general, but costlier, one based on the data set itself [8]. More of MurTree's particularities and features will be discussed in the preliminaries, as this is the algorithm chosen to be extended with support for multivariate cuts.

3.4 Optimal Oblique Trees

Lastly, the area of optimal oblique trees seems to unfortunately be the scarcest. As previously implied, optimal decision trees are a field in which there is still plenty of researching opportunity, and optimal oblique trees appear to be a subspace for which that is even more so the case. This is certainly in part because of the doubly prohibitive cost of creating both optimal and oblique trees. Nevertheless, while it might not have direct applicability for a dynamic-programming based optimal oblique tree algorithm, which is what this work attempts, a MIP (Mixed Integer Programming) based approach has been published. In MIP some, but not all variables part of the constraints are integers. This formulation is "based on a 1-norm support vector machine model", and runtimes are improved through provided cutting plane techniques meant to tighten the relaxation of the problem [18].

4 Preliminaries

The following paragraphs detail everything required to support the findings of the research, as well as the experimental setup from subsequent sections. First comes a presentation of the framework common to all algorithms. A brief description of the specific version of CART implemented as part of this work follows. Last but not least comes a more in-depth look at the version of MurTree upon which the multivariate extension is built.

4.1 Common Framework

There are three main components shared between the different decision tree algorithms. These are the data itself, predicates, and nodes. The main points regarding the structure and implementation of all three are covered in the following paragraphs.

4.1.1 Input Data

All the algorithm implementations mentioned in this paper aim to solve a binary classification problem on a binary data set. Each row of data represents an object instance and contains a series of '0' or '1' characters divided by whitespace. The leftmost character in a row constitutes one of the two possible target values (classes) for the current instance. Subsequent characters denote the value (more specifically, the presence) of a feature. Thus the second character in a row shows the value of feature with index zero, the third character shows that of feature one and so on. Every instance has the same amount of features; there are no missing values.

A small example dataset can be generated for the univariate tree in Figure 1 to clarify the overall layout. Consider feature zero to be a high salary, feature one to be a short commute, feature two to be the benefits. Furthermore, let zero represent a declined offer and one an accepted offer:

```
0010
0011
1100
```

The misclassification is given by the number of instances incorrectly labelled by the tree. For the above dataset and the tree in Figure 1, the misclassification score is one because the second instance gets incorrectly assigned the "decline" class.

4.1.2 Predicates

In order to support the easy creation and integration of new predicate types within tree decision points, a simple interface is devised. It dictates that any predicate must give a binary outcome when presented with an object's feature vector. The logic behind any decision process is custom to the individual predicate. In this manner, defining predicates that check for an arbitrarily complex condition is trivial.

4.1.3 Nodes

The binary decision trees covered in this project contain the usual two types of nodes: predicate and classification. More specifically, the non-terminal predicate nodes hold a reference to both a left and a right child along with, of course, a predicate. When classifying, items for which the inner predicate is false are forwarded to the left child, while those for which the predicate yields true are passed over to the right one. The terminal classification nodes solely denote one of the two possible target outcomes.

4.2 CART Algorithm

As CART is the primordial heuristic orthogonal decision tree algorithm, the implementation featured in this project is primarily used as a control for other algorithms. It is an adaptation of Josh Gordon's Python code published to GitHub for purely binary data sets, which also foregoes the probabilistic nature of leaf nodes. Additionally, the possibility to limit the tree's predicate node depth has been added. When not bounded by depth, the algorithm keeps creating decision nodes as long as the heuristic value of the predicate deemed best is strictly positive.

4.3 MurTree Algorithm

The MurTree optimal orthogonal tree algorithm is the bedrock for this paper's multivariate extension. Its three abovementioned main contributions are the fast sub-algorithm for computing trees of depth two, the novel caching alternatives, and the similarity-based lower bound mechanism. Even so, many other optimisations outlined in the original paper play a role in further accelerating the tree building process. The following subsections outline which optimisations are present in this paper's implementation while providing extra context around some choices. The complete algorithm, along with all enhancements explored by the authors, is exhaustively showcased in the original paper [8].

4.3.1 Evaluation Metric

As with the original algorithm, this MurTree implementation directly minimises the number of misclassifications on the training data set. Due to the optimal nature of the algorithm, it outputs one of the possible trees with the minimal amount of misclassifications on the training data, given the depth and size constraints. Assuming the training data constitutes a representative sample, the resulting tree is fit to classify unseen objects accurately. While not of concern to this work, other possible metrics for the algorithm to optimise around exist. They are often utilised when dealing with imbalanced data sets containing more samples of one class than the other and have even been discussed in the context of MurTree [19].

4.3.2 Caching

A cache system that avoids recomputing subtrees for previously encountered subsets of the data can help greatly speed up the tree generation process. While the bookkeeping required to manage the cache does incur a significant overhead, a good caching policy often speeds up the overall computation by a considerable factor.

The MurTree paper proposes two different caching policies, one uses branches as keys for the cache entries, and the other uses the input data (sub)set for the node to be computed. A branch represents a sorted list of all the features which make up the decision nodes leading to the node that needs to be calculated. The branch-based system enjoys a lower overhead for cache management, but the resulting strategy is less general than that of its counterpart. This is because two different branches could lead to the same dataset, in which case the branch system would always recompute the subtree, while the data set one would recognise whether it has seen this subset of the entire training data before.

This implementation uses data set caching. The main reason for the choice is that once splits are no longer orthogonal, branch representation loses its main appeal, the compact representation. Furthermore, even if the branch approach were to be adapted to deal with any predicate type, because of the increased amount of possible predicates, the chances to end up with the same two branches at two different nodes within the tree would be lower. Thus, data set caching was deemed the one more likely to consistently hit the cache.

The cache itself is a multi-layered data structure. At the top layer is a list with a size equal to the number of instances in the initial training data set. Each element in this list maps a dataset, depth and amount of nodes to a cache entry. Cache entries contain two fields: the optimal subtree for the given dataset, max depth, and max amount of nodes, along with the tree's number of misclassifications on the data set. The outer list is supposed to lessen the likelihood of collisions between the hashes of different data sets.

4.3.3 Depth Two Sub-algorithm

The specialised algorithm for trees of depth two is another essential part of MurTree's performance. This project's implementation computes all optimal subtrees with a predicate node depth of at most two for each call to the subroutine. Those subtrees are depth two with three predicate nodes, depth two with two predicate nodes, and depth one. If any one of the three trees has a higher misclassification count than another with fewer predicate nodes (including the tree of depth 0, containing just a leaf node), its entry is replaced by the better one. This possibility for replacement does imply that the sub-algorithm can return a tree with fewer nodes or a lower depth than the ones given as input parameters. The only other instance in which such pruning could happen in the implemented version is inside the general recursion case, where if no subtree with the required depth and amount of predicate nodes with a lower misclassification score than the best leaf exists, the latter is returned. Finally, before the subroutine returns the requested subtree, all computed optimal subtrees are added to the cache. The incremental computation discussed in the original paper is not present in this implementation.

5 Own Contribution

Creating a multivariate optimal tree algorithm increases the search space compared to the already expensive optimal univariate alternative drastically. Even in heuristic trees, multivariate algorithms usually induce purely oblique splits since considering even higher-order splits greatly increases computational complexity, likely leads to diminishing returns and considerably hinders the simple structure touted by decision trees.

In order to manage time complexity, limitations ought to be enforced upon the kind of multivariate splits employed and the nodes allowed to consider them. The final selection of multivariate predicates takes advantage of the binary representation of features, and it also limits itself to purely bivariate formulations. Additionally, further limitations are applied to the max tree depth upon which multivariate splits are considered alongside univariate ones.

The remainder of this section outlines these constraints, formalises the extension to the algorithm, briefly analyses the increase in search space, and touches upon parallelisation of the program.

5.1 Predicate Type Constraints

The fact that the input consists of binary features can be exploited when defining which types of multivariate splits the algorithm may employ. One possible formulation, similar to that of a general oblique split, is to assign a binary weight to each feature. In such a case, the modulo two dot product of the weights and the current feature vector determines which of the root's two children is to be followed further. While this formulation is quite flexible, it requires an exponentially large amount of predicates to be considered at each decision point, making it unfeasible in most circumstances due to its poor scaling as the number of features increases.

Imposing further constraints on the kinds of multivariate splits supported by the algorithm is indispensable to narrow down the search space. To reduce the exponential amount of nodes considered for each decision point to a polynomial one, one could set a constant upper bound on the number of features that make up a predicate. Nevertheless, small changes in this upper bound still have a profound impact on the runtime. For this reason, the proposed algorithm only utilises predicates containing at most two features.

While only supporting binary predicates is a compromise hopefully avoidable in the future, the benefits of multivariate splits should become perceptible even with such simpler predicates. A binary predicate system does, however, allow Boolean operators to be used directly as predicate types. Thus, the three types of multivariate predicates supported by the proposed algorithm are manifestations of the classic AND, OR, XOR operators. The complementary operators XNOR, NAND, NOR would merely invert the results of their non-negative counterparts, and thus they are ignored.

5.2 Predicate Depth Constraints

A final set of restrictions gets placed on the depth up to which multivariate splits are employed. Regardless of any other parameters, the previously mentioned MurTree depth two sub-algorithm is responsible for generating the final up to two levels of any output tree. Hence only the usual orthogonal predicates will exist in (sub)trees with a depth lower than three. Once again, this is mainly a performance-oriented constraint, but, particularly as overall tree depth increases, the benefits of multivariate splits closer to the leaves might not be as pronounced. Furthermore, aside from the preexisting parameters from MurTree constraining the depth d and the total number of decision nodes n, a third parameter α is added. It indicates that multivariate cuts can exist in the first α levels of the tree. Thus, α should be a natural number less than or equal to $max(0, d-2)$. A value of $\alpha = 0$ gives a purely orthogonal tree (i.e. as generated by the original MurTree algorithm); $\alpha = 1$ creates a tree that only considers the previously discussed multivariate splits in the root. More generally, the topmost $2^\alpha - 1$ nodes attempt multivariate splits.

5.3 Formal Description

In what follows, let MurTree-α denote the proposed algorithm alongside the selected value for α (e.g. MurTree-2). Since the previously discussed bivariate predicates can be easily transferred over to the CART algorithm, the same notation is used for the heuristic representative. In CART's case, the only difference is that α could have a value of at most d. Allowing multivariate splits to be considered even throughout the two deepest levels is possible thanks to CART's substantially more reasonable computational requirements, as well as the lack of a specialised depth two sub-algorithm.

```
Algorithm 1: GetPredicates
  Input: Depth $d$ of current root node
  Output: Set of predicates to be considered for current split
1 $predicates \leftarrow \emptyset$
  // $f$ is the number of features
2 for $i \leftarrow 0$ to $f - 1$ do
      // $f_i$ indicates the value of the $i$-th feature
3     $predicates \leftarrow predicates \cup (f_i)$
4     if $d \leq \alpha$ then
5         for $j \leftarrow 0$ to $i - 1$ do
6             $predicates \leftarrow predicates \cup (f_i \wedge f_j)$
7             $predicates \leftarrow predicates \cup (f_i \vee f_j)$
8             $predicates \leftarrow predicates \cup (f_i \oplus f_j)$
9 return $predicates$
```

Based on the above pseudocode, a dynamic programming formulation for the proposed trees can be built on top of $T(\mathcal{D}, d, n)$ from *Eq. 1* in the original MurTree paper:

$$T(\mathcal{D}, d, n, \alpha) = \begin{cases} min\{T(\mathcal{D}(\overline{pred}), d-1, n-i-1, \alpha-1) & \\ \qquad +T(\mathcal{D}(pred), d-1, i, \alpha-1) & n > 0 \wedge d > 2 \wedge \alpha > 0 \\ \quad : f \in GetPredicates(d), i \in [0, n-1]\} & \\ T(\mathcal{D}, d, n) & \text{otherwise} \end{cases} \tag{2}$$

5.4 Search Space Increase Comparison

To emphasise how much the search space enlarges in a multivariate environment, even after all the discussed restrictions, a short comparison between the univariate and multivariate algorithms follows. Before it, to motivate the obtained results, a couple of intermediary steps are required.

With f as the number of binary features describing objects and n as the number of predicate nodes, a formula for the total amount of different univariate trees may be derived.

	$\frac{\text{MurTree-1}}{\text{MurTree-0}}$	$\frac{\text{MurTree-2}}{\text{MurTree-0}}$
$f = 10$	14.5x	3048.62x
$f = 50$	74.5x	413493.62x

Table 1: Univariate to Multivariate Search Space Increase Given f and α

Assuming that the values of node predicates are independent, each of the n nodes may have f possible values, meaning that:

$$|\text{MurTree-0 Search Space}| = f^n \text{ trees} \tag{3}$$

With this in mind, a general formula for MurTree-α can also be derived. Each of the first $2^\alpha - 1$ nodes would consider $3 * (1 + ... + (f - 1))$ multivariate predicates in addition to the f univariate ones, while no changes would be made for other nodes. Assuming all multivariate nodes are the root of subtrees with a depth greater than two, the following holds:

$$|\text{MurTree-}\alpha\text{ Search Space}| = (f + 3 * (1 + ... + (f - 1)))^{2^\alpha - 1} f^{n-(2^\alpha - 1)} \text{ trees} \tag{4}$$

Finally, a comparison can be made through Table 1 between the univariate and multivariate modes in an absolute manner. The results highlight the fundamental problem of building optimal multivariate trees. Such sizeable increases in the total number of trees are problematic when an optimal algorithm has to explore each possible tree, at least up until pruning is known to be safe. While increases in the number of features already considerably augment the size of the search problem, slight increases in the maximum multivariate depth have an even more profound impact.

5.5 Parallel Computing

Parallelisation is the final measure used by this paper to help somewhat speed up computation. Its effectiveness relies highly on the available hardware resources, and it does not necessarily aid the multivariate case more than the univariate one. Nonetheless, it can deliver sizeable speed improvements in the absolute sense.

Since the implementation for the project is in C++, OpenMP [20] was the multithreading API of choice. A thing to note is that, when running in multithreaded mode, the program returns a possibly different optimal tree than when running sequentially. Nonetheless, this work does not concern itself with the intricacies of parallelising MurTree. That topic is explored in much more detail by a colleague within the same research group. As such, there is no further discussion on implementation details.

6 Experimental Results

Throughout this section, the constructed experimental setup is presented in detail before MurTree-1 and CART-1 are compared to MurTree-0 and CART-0 over various publicly available datasets. CART is present to gauge how the improvement of the chosen multivariate splits variates between a heuristic and optimal environment.

Identifier	Dataset	#Instances	#Features
$\mathcal{D}1$	anneal	812	73
$\mathcal{D}2$	audiology	216	148
$\mathcal{D}3$	breast-wisconsin	683	120
$\mathcal{D}4$	diabetes	768	112
$\mathcal{D}5$	heart-cleveland	296	95
$\mathcal{D}6$	hepatitis	137	68
$\mathcal{D}7$	kr-vs-kp	3196	73
$\mathcal{D}8$	lymph	148	68
$\mathcal{D}9$	mushroom	8124	119
$\mathcal{D}10$	primary-tumor	336	31
$\mathcal{D}11$	segment	2310	235
$\mathcal{D}12$	soybean	630	50
$\mathcal{D}13$	tic-tac-toe	958	18
$\mathcal{D}14$	vote	435	48
$\mathcal{D}15$	yeast	1484	89

Table 2: Description of Benchmark Datasets

6.1 Experimental Setup

The choice to not include tests where $\alpha > 1$ is performance-motivated since many datasets require prohibitively high runtimes for higher α values. Of course, this also comes with the caveat of comparatively limited improvements to model accuracy.

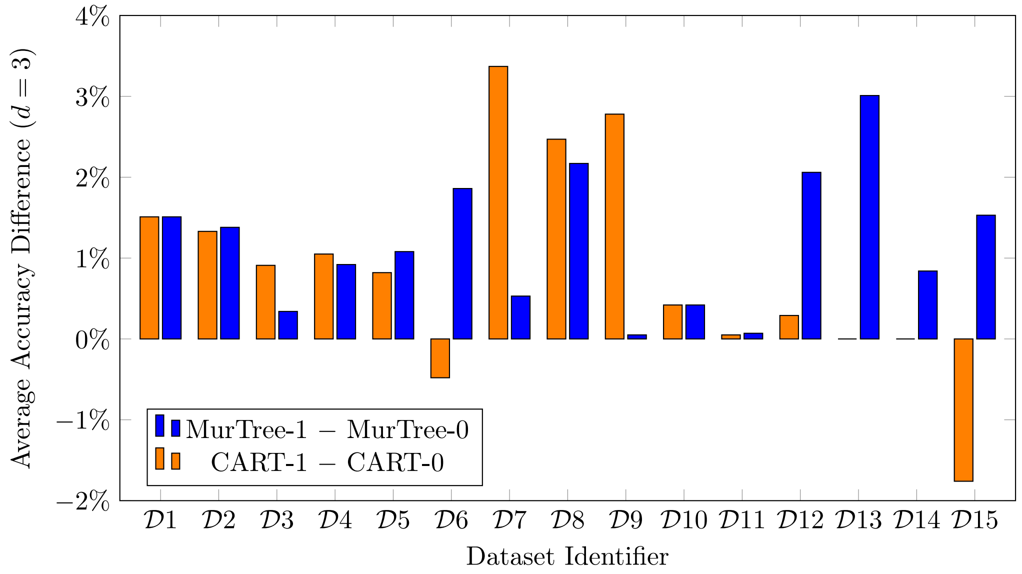
Each set of benchmarks is run for two depth constraints: $d = 3$ and $d = 4$. Furthermore, in the case of MurTree, the value of the extra n parameter is set to $2^d - 1$. Thus, MurTree aims to generate a complete binary tree, while CART only considers the depth restriction.

The utilised datasets comprise a subset of those found in the PyDL8.5 repository. They conform to subsubsection 4.1.1 and have already been used to evaluate DL8.5 and MurTree [6][8]. Table 2 shows the number of instances and features for the selected datasets, alongside a shorthand identifier to be used in subsequent plots for the sake of compactness.

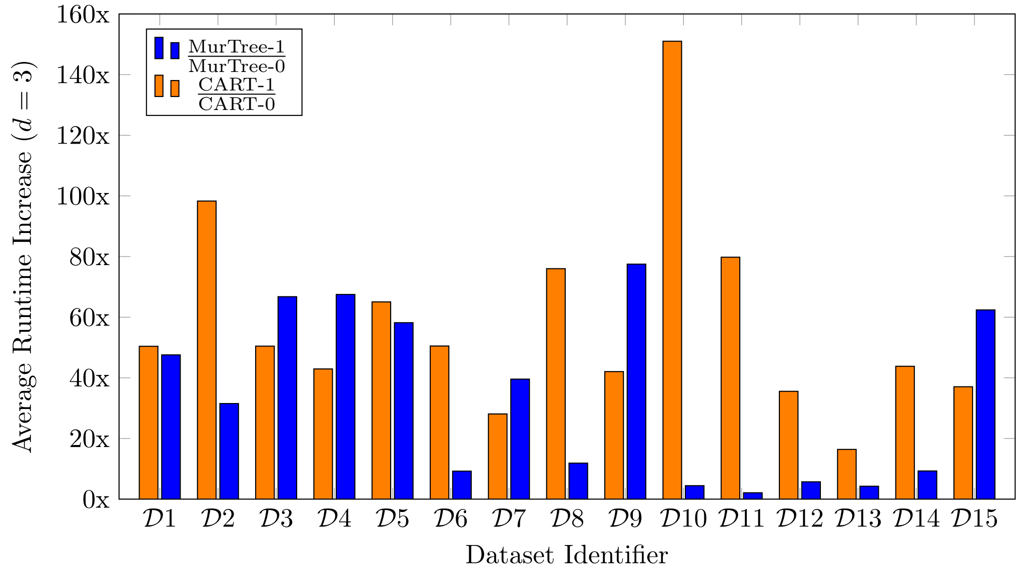
The measured metrics are average classification accuracy on the train set (i.e. the dataset used to build the tree) and average runtime. For every dataset, instances are initially shuffled pseudo-randomly, based on a seed which allows rerunning an experiment with the same parameters. It can either be generated automatically or provided by the user. For the following benchmarks, its value was 42. After the shuffle, data instances get evenly split into $k = 10$ folds. One by one, the folds serve as input for all tested algorithm instances.

The faculty's private GitLab instance hosts the complete code, and its built-in runners were used to perform the tests. A release CMake build was created in the test environment, with GCC 11.1.0 as the compiler and the number of OMP threads at the default value of 128. Note that all instances of CART were run strictly in sequential mode.

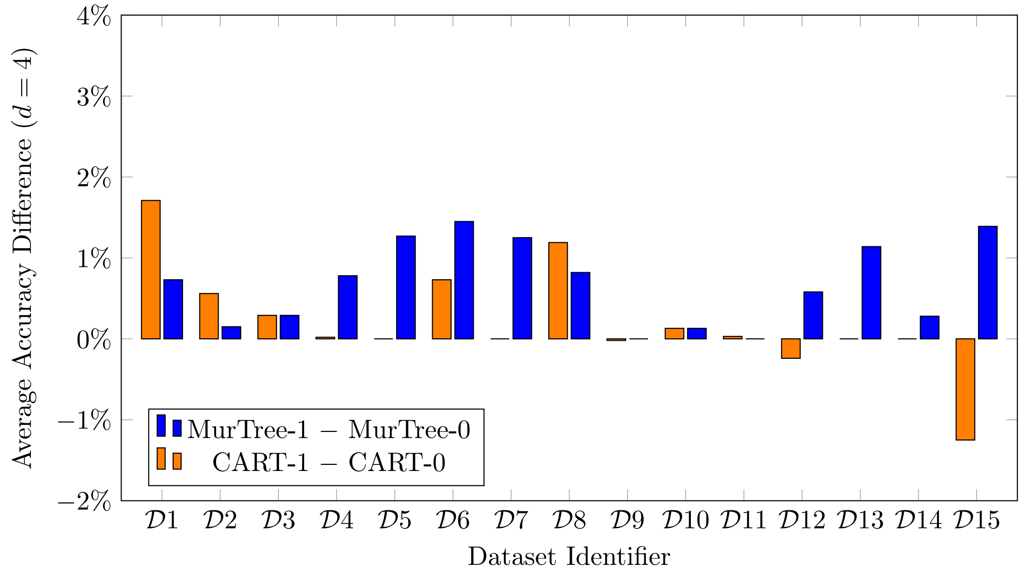
6.2 Benchmarks And Discussion



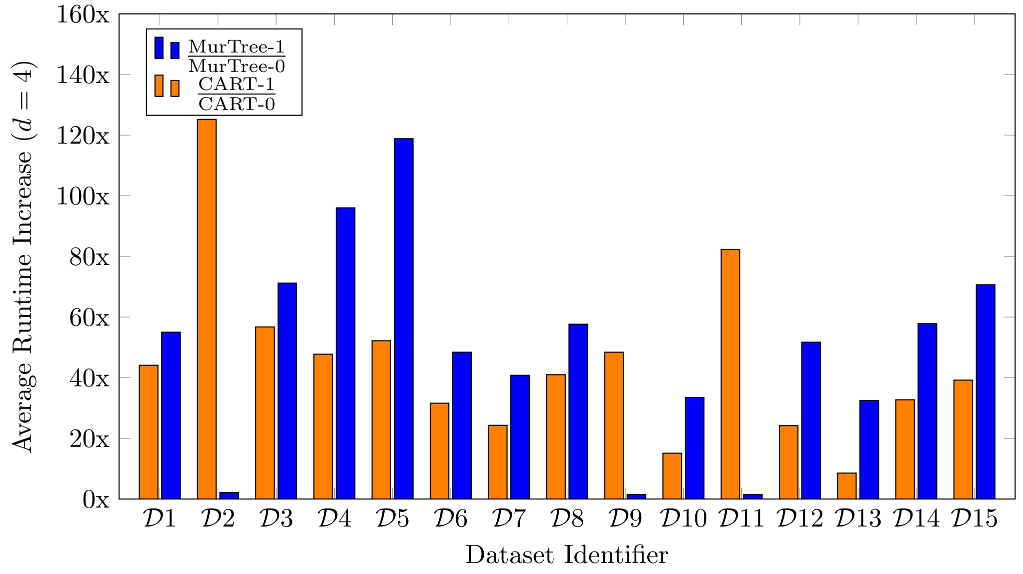
At $d = 3$, multivariate roots generally offer noticeable, albeit not huge, gains in accuracy for both algorithm types. Unsurprisingly, MurTree never suffers accuracy decreases in the multivariate setting, but CART occasionally does due to the imperfect nature of its heuristic.



When it comes to speed, for $d = 3$, both the heuristic and optimal solution tend to see decreases in performance of many orders of magnitude, as expected. The two approaches are often evenly matched, but for datasets with fewer features, MurTree curiously appears to scale much better. The phenomenon is merely a manifestation of the fact that, in these limited cases, the large amount of available threads only begins to show its effects once the search space increases somewhat.



Moving on to $d = 4$ and maintaining multivariate splits just in the root paints an accuracy picture similar to before, but with even more reserved gains. Given that the ratio of nodes in the tree considering multivariate splits is now even lower in the case of both algorithms, this is pretty intuitive. Thus, as the general depth increases further, accuracy gains would likely tend to keep decreasing so long as alpha does not also increase.



In terms of speed, the results at $d = 4$ do not dramatically differ from those prior. The overall slowdown factor remains the same between the optimal and heuristic members. There are also fewer outliers than at $d = 3$. The two remaining cases in which MurTree vastly out scales its counterpart are coincidentally the same ones in which its multivariate absolute accuracy reaches a full one hundred per cent. However, for $\alpha > 1$, MurTree's computational requirements would start growing a lot faster than those of CART, leading to much more apparent differences in relative compute time.

All in all, as the obtained results show that the multivariate version of MurTree has greater or equal training accuracy than the univariate version for each of the examined datasets, the initial research question is successfully answered. Nevertheless, the real-world limitations of the method are blatantly obvious, mainly due to the extra time required to generate the trees, even when multivariate candidates are considered only for the root node. Limiting the max multivariate depth to one also decreases the potential accuracy gains possible with more liberal use of multivariate splits.

7 Conclusions And Further Work

As evidenced by the experimental section, creating multivariate optimal trees is possible, but many compromises are required. Even with severe limitations to the nature and placement of multivariate splits, the cost increase remains one of several orders of magnitude. Techniques like multithreading, while noticeable, are futile in measuring up against the drastically expanded search space. Algorithmic improvements, likely formulated with multivariate splits in mind, are a necessity. Such improvements may be found in the area of pruning by possibly devising a procedure that aggressively discards sizeable patches of the search space as early as possible, without loss of optimality.

With that in mind, a possibly tweaked variant of the similarity-based bound showcased in the MurTree paper could be a first step towards improving efficiency for all types of predicates. Improvements to the current multithreading implementation are also possible, although they would not constitute a solution to the problem in its greater scope. Lastly, further study into the types of multivariate splits that could yield the best results within the set constraints might help make the most of the situation without radically improving the core algorithm.

8 Responsible Research

While technical results and takeaways are the primary concerns of any scientific research, it is nonetheless crucial to take a step back and think about the broader implications of the research process itself and its conclusions. Even though this work is very limited in scope compared to the forefront of computer science research, there is still space to reflect upon integrity and reproducibility.

The problem of integrity is a multifaceted one, but given the context of the research project, some possible points for concern can be addressed swiftly. First of all, because this work is part of an end-of-bachelor project, the opportunities for a conflict of interest forming are minimal, and no sponsorships or grants are involved. Secondly, no human or any other living subjects were utilised in this research, meaning that no violations are possible on that front. Thirdly, self-plagiarism is also unrealisable due to the lack of prior personal experience regarding optimal decision trees. With that in mind, the more general issue of plagiarism remains as relevant as always, considering that this work relies heavily on existing research. Still, the individual effort to properly reference and contextualise work mentioned within the project is also safeguarded by the multiple reviews from peers, course staff and the responsible teacher of the text at different stages throughout its conception.

On the contrary, bias and discrimination can become a problem when using decision trees to classify data tied to socially sensitive contexts. While all decision trees are favoured for their interpretability [1], part of the motivation for further development of optimal decision trees like MurTree is to facilitate dealing with bias in the provided data or the heuristic employed in the tree generation [8]. Optimal decision trees do not automatically guarantee this fairness, but they make programming constraints that preserve it more attainable. With that in mind, the multivariate extension to MurTree concerning this paper should at the very least not worsen MurTree's abilities to combat bias, given that it essentially just expands the space already searched by the original algorithm to allow for new types of decision conditions. One could even argue that a refined version of this multivariate MurTree could further aid in avoiding biased behaviour because it enables asking more complex questions about the data at each decision point.

Data manipulation and reproducibility were essential considerations in the creation of the experimental section. Benchmarks comparing the different algorithms were run on a variety of publicly available data sets, and the results for the metrics of interest are reported in their entirety. They clearly show both the varying accuracy advantages of the multivariate MurTree approach, as well as the generally exponential increase in computation time. Multiple measures were also taken to ensure reproducibility. The code and the data sets, in the encoding used by the algorithms, are submitted alongside this text. A seed system is used for the pseudo-random shuffling required to partition the data. When verifying experimental results, this allows anyone to reproduce the same shuffles as those used in the paper.

All in all, while this project did not pose any out of the ordinary challenges from the standpoint of reproducibility or ethical integrity, there were still important considerations to be made throughout the research process in order to maximise reproducibility and avoid compromising integrity.

References

[1] Wikipedia contributors. (2021). "Decision tree learning — Wikipedia, the free encyclopedia," [Online]. Available: `https://en.wikipedia.org/w/index.php?title=Decision_tree_learning&oldid=1027552736` (visited on Jun. 22, 2021).

[2] Wikipedia contributors. (2021). "Decision tree — Wikipedia, the free encyclopedia," [Online]. Available: `https://en.wikipedia.org/w/index.php?title=Decision_tree&oldid=1013198756` (visited on Jun. 22, 2021).

[3] L. Hyafil and R. L. Rivest, "Constructing optimal binary decision trees is NP-complete," *Information Processing Letters*, vol. 5, no. 1, pp. 15–17, 1976, ISSN: 0020-0190. DOI: `https://doi.org/10.1016/0020-0190(76)90095-8`.

[4] L. Breiman, J. Friedman, C. Stone, and R. Olshen, *Classification and Regression Trees.* Taylor & Francis, 1984, ISBN: 9780412048418.

[5] S. Aghaei, M. J. Azizi, and P. Vayanos, *Learning optimal and fair decision trees for non-discriminative decision-making*, 2019. arXiv: `1903.10598 [cs.LG]`.

[6] G. Aglin, S. Nijssen, and P. Schaus, "Learning optimal decision trees using caching branch-and-bound search," *Proceedings of the AAAI Conference on Artificial Intelligence*, vol. 34, no. 04, pp. 3146–3153, Apr. 2020. DOI: `10.1609/aaai.v34i04.5711`.

[7] S. Nijssen and É. Fromont, "Optimal constraint-based decision tree induction from itemset lattices," *Data Mining and Knowledge Discovery*, vol. 21, pp. 9–51, Jan. 2013. DOI: `10.1007/s10618-010-0174-x`.

[8] E. Demirović, A. Lukina, E. Hebrard, J. Chan, J. Bailey, C. Leckie, K. Ramamohanarao, and P. J. Stuckey, *MurTree: Optimal classification trees via dynamic programming and search*, 2020. arXiv: 2007.12652 [cs.LG].

[9] S. K. Murthy, S. Kasif, and S. Salzberg, "A system for induction of oblique decision trees," *J. Artif. Int. Res.*, vol. 2, no. 1, pp. 1–32, Aug. 1994, ISSN: 1076-9757.

[10] D. C. Wickramarachchi, B. L. Robertson, M. Reale, C. J. Price, and J. Brown, *HHCART: An oblique decision tree*, 2015. arXiv: 1504.03415 [stat.ML].

[11] J. R. Quinlan, *C4.5: Programs for Machine Learning*. San Francisco, CA, USA: Morgan Kaufmann Publishers Inc., 1993, ISBN: 1558602380.

[12] X. Wu, V. Kumar, J. Ross Quinlan, J. Ghosh, Q. Yang, H. Motoda, G. J. McLachlan, A. Ng, B. Liu, P. S. Yu, Z.-H. Zhou, M. Steinbach, D. J. Hand, and D. Steinberg, "Top 10 algorithms in data mining," *Knowl. Inf. Syst.*, vol. 14, no. 1, pp. 1–37, Dec. 2007, ISSN: 0219-1377. DOI: 10.1007/s10115-007-0114-2. [Online]. Available: https://doi.org/10.1007/s10115-007-0114-2.

[13] Wikipedia contributors. (2021). "Householder transformation — Wikipedia, the free encyclopedia," [Online]. Available: https://en.wikipedia.org/w/index.php?title=Householder_transformation&oldid=1028223110 (visited on Jun. 22, 2021).

[14] M. Norouzi, M. D. Collins, D. J. Fleet, and P. Kohli, *Co2 forest: Improved random forest by continuous optimization of oblique splits*, 2015. arXiv: 1506.06155 [cs.LG].

[15] B.-B. Yang, S.-Q. Shen, and W. Gao, "Weighted oblique decision trees," *Proceedings of the AAAI Conference on Artificial Intelligence*, vol. 33, no. 01, pp. 5621–5627, Jul. 2019. DOI: 10.1609/aaai.v33i01.33015621. [Online]. Available: https://ojs.aaai.org/index.php/AAAI/article/view/4505.

[16] S. Aghaei, A. Gomez, and P. Vayanos, *Learning optimal classification trees: Strong max-flow formulations*, 2020. arXiv: 2002.09142 [stat.ML].

[17] G. Aglin, S. Nijssen, and P. Schaus, "Pydl8.5: A library for learning optimal decision trees," in *Proceedings of the Twenty-Ninth International Joint Conference on Artificial Intelligence, IJCAI-20*, C. Bessiere, Ed., Demos, International Joint Conferences on Artificial Intelligence Organization, Jul. 2020, pp. 5222–5224. DOI: 10.24963/ijcai.2020/750. [Online]. Available: https://doi.org/10.24963/ijcai.2020/750.

[18] H. Zhu, P. Murali, D. T. Phan, L. M. Nguyen, and J. R. Kalagnanam, *A scalable mip-based method for learning optimal multivariate decision trees*, 2020. arXiv: 2011.03375 [cs.LG].

[19] E. Demirović and P. J. Stuckey, "Optimal decision trees for nonlinear metrics," *Proceedings of the AAAI Conference on Artificial Intelligence*, vol. 35, no. 5, pp. 3733–3741, May 2021. [Online]. Available: https://ojs.aaai.org/index.php/AAAI/article/view/16490.

[20] Wikipedia contributors. (2021). "Openmp — Wikipedia, the free encyclopedia," [Online]. Available: https://en.wikipedia.org/w/index.php?title=OpenMP&oldid=1027825692 (visited on Jun. 22, 2021).