Optimal Decision Trees for non-linear metrics

A geometric convex hull approach

Bogdan-Andrei Bancuta[1]
Supervisors: Emir Demirović[1]**, Jacobus G. M. van der Linden**[1]
[1]EEMCS, Delft University of Technology, The Netherlands

Abstract

In the pursuit of employing interpretable and performant Machine Learning models, Decision Trees has become a staple in many industries while being able to produce near-optimal results. With computational power becoming more accessible, there has been increasing progress in constructing Optimal Decision Trees. It guarantees optimal solutions with respect to different metrics within a given size limit on training data while requiring a smaller number of nodes and becoming more viable to compute on real-world data. However, non-linear metrics, which are very effective when evaluating trees on imbalanced datasets, still represent a challenge regarding runtime performance and scalability. Previous approaches generate the Pareto Front of the set of possible solutions, an expensive operation in computing the optimal tree. To address this gap, we introduce a novel merging algorithm of two Pareto Fronts using convex hulls, offering better pruning and leading to an increase in scalability. The experiments show a significant improvement in runtime of almost 10% on bigger datasets and higher-depth trees using the F1-score metric, with the potential to be applied to other convex metrics.

1 Introduction

Machine Learning algorithms have recently been increasingly employed in critical domains such as healthcare or the judicial system [1]. Unfortunately, the lack of transparency and accountability of most of the nowadays deployed systems can lead to erroneous decisions that cannot be explained. Therefore, interpretability is important in both revealing why the model reached a certain outcome and offering explanations on how it arrived at that outcome, its "thought process" [2]. This reasoning will increase the general people's trust in these systems, which was severely damaged by the continuous presence of biases and may reveal patterns in data unbeknownst to engineers.

Decision trees is an important Machine Learning model, as it manages to break complex data into simpler parts and is easily interpretable as each decision and its effect can be observed. However, most common and popular decision trees use greedy heuristics such as CART [3], which minimize an objective function at the current level, not taking into account the structure of the tree, such as the possible number of splits left, into consideration. Therefore, the need for **Optimal Decision Trees**, which are more compact, needing fewer nodes to achieve similar performance and thus increase interpretability, can optimize custom objectives and find the global optimum on the training set while also having a relatively high out-of-sample performance is desirable [4].

The problem of constructing an Optimal Decision Tree is an NP-hard problem [5], having to consider all the possible combinations of feature splits to find the one that best optimizes the desired objective. Many approaches have tried to explore this space more efficiently, using mixed-integer programming (MIP) [6, 7], constraint programming (CP) [8], or boolean satisfiability (SAT) [9], but with severe scalability issues for trees of depth more than three or datasets with more than a thousand instances [10]. On the other hand, Dynamic Programming methods have proved to be much more effective by directly exploiting the structure of the tree, recursively splitting the problem into two identical smaller ones.

Thus, many improvements have been introduced throughout the years, such as the DL8.5 algorithm [11] that used itemset mining and branch and bound techniques to better prune and find the most promising trees faster. MurTree [4] has further improved the dynamic programming approach by including a depth-two solver that returns the optimal decision tree of depth-two iteratively by calculating the frequency counts of the positive and negative labels. As a result, it can iterate through all possible trees without the need to split the dataset over a feature or even access the dataset at all.

Another important contribution is the adaptation of decision trees to non-linear metrics. These are particularly useful when datasets are imbalanced, and the relation between the misclassifications of classes is much more complex, for example, when determining the false positives and negatives of a medical test [12]. As the independence of the left and right subtrees is no longer satisfied for this kind of metrics [13], and it represented the foundation for many of the previous works, the key was to approach this as a bi-objective optimisation problem. Demirović and Stuckey [13] have developed a new algorithm for tackling problems with two different objectives. They showed that the candidate solutions lie on the Pareto Front, a set of solutions in which no one is better than the other with respect to both objectives. This method has allowed the algorithm to run on a wide variety of metrics at the expense of runtime performance.

The current approaches compute the whole set of solutions generated by combining two Pareto Fronts and only then reducing it to the possible candidates for the best score, which is an expensive operation, quadratic with respect to the size of the sets. Therefore, the main aim of this paper is to evaluate whether other methods to store the candidate solutions can be employed to further increase the performance of the Optimal Decision Trees when using non-linear metrics. Specifically, if the convex hull of the Pareto Front can be used and how it will affect the performance of the algorithm. This method will allow for a faster computation of the candidate solutions by combining and maintaining only the convex hull of the Pareto Front; a visual representation can be seen in Figure 1. The results from our experiments show an almost 10% improvement in runtime compared to the previous state-of-the-art approaches.

The structure of the paper is as follows: Section 2 will present advancements and contributions that have already been made in improving the runtime and generalizability of Optimal Decision Trees. Notation that will be used throughout the paper, as well as background information about the problem, is introduced in section 3, and then our method of achieving a faster merging algorithm is described in section 4. Section 5 presents the experiments we performed to verify the runtime of our proposed algorithm and a discussion of the results that we have obtained from these experiments and their potential impact. Section 6 documents the steps we took to ensure our findings are accessible and reproducible. Final remarks and recommendations for future action are mentioned in section 7.

2 Related work

This section will focus on three contributions to the Dynamic Programming approach, which has proved to be the best-performing one so far [10].

Although the problem of finding Optimal Decision Trees was framed using dynamic programming more than 50 years ago [14], the emergence of new techniques has shifted the focus away throughout the years [4]. Emir Demirović et al. have formalised the notation and optimizations from the past under a conventional dynamic programming approach as well as brought forth further improvements. They have highlighted two important properties of decision trees, independence, once a feature is selected, the dataset is split into two disjoint sub datasets, thus allowing to compute the misclassification score of a node as the sum of misclassifications of its children. Overlap, refers to the order of the feature splits, more precisely that any order of the same feature splits will lead to the same result for a particular instance, thus allowing to establish an order of the splits, pruning away a lot of possible trees. Moreover, this paper also presents a specialized depth two solver, that can give the lowest misclassification score of a depth-two tree iteratively, without the need for accessing the data more than once. This is achieved by computing frequency counts of instances where the two selected features are present and then efficiently deriving from this information the tree with the lowest score.

As already briefly stated in the Introduction, non-linear metrics are important when optimizing multiple objectives but also when dealing with imbalanced datasets. Demirović and Stuckey [13] have adapted the MurTree algorithm discussed above to function on non-linear metrics by treating this as a bi-objective optimisation problem. Strictly speaking, on the F1-score metric, the solution to a certain dataset was defined as a pair of numbers (fp, fn), denoting the false positives and false negatives, respectively, achieved by a decision tree of a certain depth. Due to the non-linearity of the metric, the independence of the left and right subtrees no longer holds, and combining the best solution from each no longer guarantees the best solution for the parent node. They proposed that the optimal solution lies on the Pareto Front and, thus, each node will have one storing the candidate solutions. There is, however, one limitation to this algorithm, as it can be applied to any non-linear metric, not just the F1-score, as long as it is monotonic.

Dynamic programming approaches have proved to be more efficient than any other approach, but they lack the adaptability to different objectives or constraints [10]. van der Linden, de Weerdt, and Demirović have created a framework, STreeD, which aims to employ the efficiency and optimizations found so far in dynamic programming to be able to construct Optimal Decision Trees of any optimization task that is separable, a necessary condition for DP to work. Separability is defined as the result of a subtree not being influenced by any other decision taken outside it - each subtree is a separate problem from the other ones. The results of this framework are impressive by being able to solve five tasks such as: Cost-sensitive classification, Prescriptive policy generation, Non-linear classification metrics, Group fairness, and Classification accuracy, while sacrificing little performance compared to the dedicated algorithms for those tasks and still be orders of magnitude faster than any other approach such as mixed-integer programming.

3 Preliminaries

Section 3.1 will introduce notation that will be used throughout the rest of the paper, while Section 3.2 will formally describe the problem that we want to solve.

3.1 Notation

feature	A variable encoding information about an object
instance	A collection of features and a value denoting the label of the instance
$\mathcal{D}$	Set of instances
$\lvert\mathcal{D}\rvert$	Number of instances
$\mathcal{D}^+$	Set of instances where label is 1
$\mathcal{D}^-$	Set of instances where label is 0
$\mathcal{D}(f)$	Set of instances where the value for feature f is 1
$\mathcal{D}(\overline{f})$	Set of instances where the value for feature f is 0
d	Depth of tree
$\mathcal{F}$	Set of available features to split on

3.2 Problem definition

Described more formally, given a dataset $\mathcal{D}$ and a maximum depth d, the goal is to find a decision tree of depth d or smaller that attains the best F1-score. Thus, the solution will be of the form (fp, fn), denoting the false positives and false negatives associated with the tree with the best F1-score.

Since this metric is non-linear and monotonic, the optimal solution lies on the Pareto Front of all candidate solutions [13].

Definition 1. *(Pareto Front) Pareto Front is a set of non-dominated solutions, where no solution is strictly better than any other with regard to both false positives and false negatives.*

$$PF(S) = \{(x, y) \in S | \neg\exists(x', y') \in S, x < x' \wedge y < y'\}.$$

Definition 2. *(Merge) Given a parent node p with feature f and Pareto fronts of its children PF_{left} and PF_{right}, the Pareto Front $PF_{p,f} = merge(PF_{left}, PF_{right})$ of the parent node p is computed as ([13]):*

$$PF_{p,f} = PF\left(\{(x_1 + x_2, y_1 + y_2) : (x_1, y_1) \in PF_{left} \wedge (x_2, y_2) \in PF_{right}\}\right).$$

Now, we present the general recurrence of the dynamic programming approach, which represents the high-level idea of the algorithm, with other enhancements and optimizations not included for simplicity reasons (adapted from [13]). The algorithm takes two parameters: the dataset we wish to find the best F1-score of and the maximum depth of the tree that we should consider.

$$T(\mathcal{D}, d) = \begin{cases} PF\left(\{(|\mathcal{D}^+|, 0), (0, |\mathcal{D}^-|)\}\right) & \text{if } d = 0 \\ PF\left(\bigcup_{f \in \mathcal{F}} \text{merge}\left(T(\mathcal{D}(\bar{f}), d-1), T\left(\mathcal{D}(f), d-1\right)\right)\right) & \text{if } d > 0 \end{cases} \tag{1}$$

As previously described, in Definition 3.2, the *merge* function combines every point from the left Pareto Front with every other one from the right Pareto Front, and then the Pareto Front of the resulting set is computed, resulting in a complexity of $O(|PF_{\text{left}}| \times |PF_{\text{right}}|)$. The size of these Pareto Fronts is bounded by $O(\min(|\mathcal{D}^+|, |\mathcal{D}^-|))$, in the case where there is a tree with every possible number of false positives or negatives. Thus, we can observe how, on big datasets, this becomes a significant problem. Moreover, the merging is performed for every feature split in order to combine the best solutions from the left and right child, resulting in a complexity of $O(|PF_{\text{left}}| \times |PF_{\text{right}}| \times |\mathcal{F}|)$ for a node at a given depth. As the depth of the tree grows, not only the sizes of the Pareto Fronts will grow as a result of adding more and more candidate solutions, but also the complexity of the algorithm will grow exponentially in terms of the size of the feature space. Therefore, to achieve more scalability and to be able to make the construction of bigger Optimal Decision Trees more feasible, significantly reducing the complexity of the merging operation is desired, and also the contribution of this paper.

4 The Minkowski Sum Merge algorithm

Our approach relies on the observation that if the metric we are using is monotonic and convex, then the optimal solution from the search space, in our case all trees of depth d, is going to be included in the convex hull of this space [15]. Adapted to our problem, if we plot all the decision trees of depth d according to the false positives and false negatives, the optimal solution according to the metric will be present in the convex hull of this space, thus it is only needed to maintain the convex hull of our search space. Intuitively, only the most extreme points in the space are reasonable candidates for best score, and the convex hull allows us to do exactly that by keeping only those points that form the polygon enclosing all others. As can also be observed in Figure 1 left, the Pareto Front is still identifying non-extreme points as potential candidates.

Property 1. *(Monotonicity) If we consider classifiers of the form f(fp, fn) then monotonicity is described by the following property which describes the benefit of always improving the classification ([13]):*

$$\forall fp, fn, fp', fn' iff fp \leq fp' \wedge fn \leq fn' \implies f(fp, fn) \leq f(fp', fn')$$

This study also mentions that most practical metrics satisfy this property, including F1-score.

Property 2. *(Convexity) A function $f : \mathbb{R}^2 \to \mathbb{R}$ is convex if for $\forall$ a,b $\in \mathbb{R}^2$ and $\lambda \in$ [0, 1], the following inequality holds ([15]):*

$$f(\lambda \times a + (1 - \lambda) \times b) \leq \lambda \times f(a) + (1 - \lambda) \times f(b)$$

Definition 3. *(Convex hull) Having a set of points S in $\mathbb{R}^2$, the convex hull CH(S) is the minimal superset of S for which holds that if x, y $\in$ CH(S) and $\lambda \in$ [0, 1] ([15]):*

$$\lambda \times x + (1 - \lambda) \times y \in CH(S)$$

In our case of 2D space, the convex hull is a convex polygon enclosing all the points in S. There are many algorithms for computing the convex hull, but our solution does not rely on any specific implementation. We have decided to use the Graham Scan algorithm, which finds the convex hull in O(n $\times$ log(n)).

Comparing these two approaches, the convex hull and the Pareto Front, in terms of the number of candidate solutions considered, we observe interesting patterns, specifically that neither one of them is completely enclosed in the other, as can be seen in Figure 1 left. While the convex hull approach also takes into consideration solutions that have a high value for both objectives, it also disregards some that the Pareto Front approach is taking. Moreover, according to Michael Mampaey et al. [15], the size of the convex hull in this particular space is bounded by O($|\mathcal{D}|^{2/3}$), which is sublinear with respect to the number of instances, a better bound than the one computed above for the Pareto Front. And, since both approaches include the optimal solution, our method combines these two approaches, achieving an even smaller set of candidate solutions than either of them that still contains the optimal solution that we are looking for.

This combining step is easily done by calculating the convex hull on the already-determined Pareto Front (see Figure 1 right). Important to note that this happens for every possible feature split, and thus, the result of calculating the Pareto Front of the union (equation 1) does not necessarily return a convex polygon. Therefore, in order for the assumptions made below about the Minkowski Sum to be true, a convex hull needs to be computed at the end to guarantee the convexity of the set of solutions stored in the current node.

Minkowski Sum. As mentioned before, combining two Pareto Fronts is quite expensive, and one of the major advantages of using convex hulls is the fact that they are convex polygons which are much faster to combine using the algorithm of Minkowski Sums; an example can be seen in Figure 2.

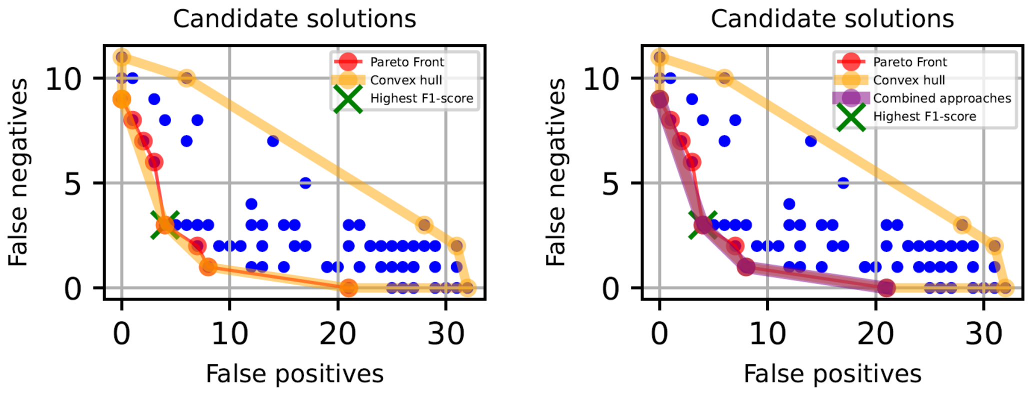

Figure 1: Pareto Front and Convex hull of depth two solutions (left), and our combined approach (right)

Definition 4. *(Minkowski sum) The Minkowski Sum of two sets A, $B \subseteq \mathbb{R}^2$ is defined as ([15]):*

$$A \oplus B = \{a + b \mid a \in A, b \in B\}$$

As can be seen from the definition above, the Minkowski Sum has virtually the same definition as the Merge function above in Definition 3.2. The property that makes this approach much better is the fact that performing the Sum on two convex polygons can be done in linear time with respect to the number of points of the two polygons [16], by just "walking" along the edges of these polygons. Therefore, with this updated method, we can bring down the complexity of the merge function from O($|PF_{\text{left}}| \times |PF_{\text{right}}|$) to O($|PF_{\text{left}}$ + $|PF_{\text{right}}|$), which should have a great impact, especially on higher-depth trees.

A pseudocode of the algorithm described above can be seen in Algorithm 1. It has a final time complexity of O($|PF_{\text{left}}| \times \log(|PF_{\text{left}}|) + |PF_{\text{right}}| \times \log(|PF_{\text{right}}|)$), the two logarithms coming from the counterclockwise ordering.

After outlining the algorithm's two primary components, we will give a brief run-through of how they function together. Looking at the children of a non-leaf node, each will have its set of candidate solutions corresponding to a convex polygon. Using the Minkowski Sum algorithm described above, we obtain another convex polygon representing the convex hull of their Cartesian product. Afterwards, we will apply the Pareto property to further prune points on the convex hull that are guaranteed not to contribute to the optimal solution. Lastly, we compute the convex hull of the re-

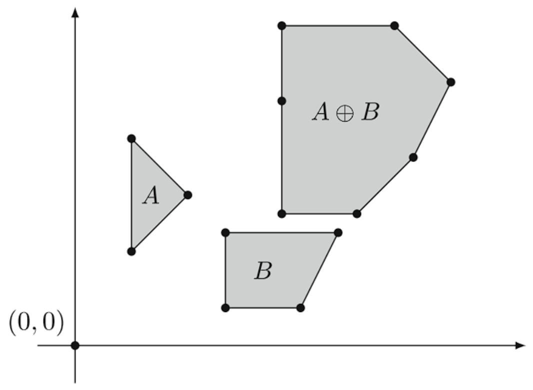

Figure 2: The Minkowski sum of two convex polygons. Each point in $A \oplus B$ is the sum of a point in A and a point in B [15].

maining set, as there are still points left that can be pruned out, and we are not guaranteed to have a convex shape yet, which is crucial. Now, we end up with the set of candidate solutions of the parent node, representing a convex polygon. Repeating the same process until we reach the root of the tree will lead to the optimal result. Since every step of this method requires some form of sorting, the final complexity is dominated by the largest set of candidate solutions, which is represented by the combination of the sets of the two children, so $|PF_{\text{left}}| + |PF_{\text{right}}|$.

Algorithm 1 Minkowski sum algorithm for combining the solutions of the left and right child

```
Input: $Sol_{left}$, $Sol_{right}$ - set of solutions corresponding to the left and right child respectively
Output: $Sol_{comb}$ - set of solutions corresponding to the combining of $Sol_{left}$ and $Sol_{right}$
Order $Sol_{left}$ and $Sol_{right}$ counter-clockwise
$Sol_{left} := Sol_{left} \cup \{Sol_{left}[0]\} \cup \{Sol_{left}[1]\}$
$Sol_{right} := Sol_{right} \cup \{Sol_{right}[0]\} \cup \{Sol_{right}[1]\}$
$Sol_{comb} := \{\}$
$i \leftarrow 0, j \leftarrow 0$
while $i < |Sol_{left}| \wedge j < |Sol_{right}|$ do
    $Sol_{comb} := Sol_{comb} \cup \{Sol_{left}[i] + Sol_{right}[j]\}$
    $cross \leftarrow$ CounterClockwise($Sol_{left}[i+1] - Sol_{left}[i], Sol_{right}[j+1] - Sol_{right}[j]$)
    if $cross \geq 0$ then
        $i \leftarrow i + 1$
    end if
    if $cross \leq 0$ then
        $j \leftarrow j + 1$
    end if
end while
while i < $|Sol_{left}|$ do
    $Sol_{comb} := Sol_{comb} \cup \{Sol_{left}[\text{i}] + Sol_{right}[\text{j}]\}$
    $i \leftarrow i + 1$
end while
while j < $|Sol_{right}|$ do
    $Sol_{comb} := Sol_{comb} \cup \{Sol_{left}[\text{i}] + Sol_{right}[\text{j}]\}$
    $j \leftarrow j + 1$
end while
return $Sol_{comb}$
```

5 Experiments and Results

Section 5.1 describes the two types of tests executed, the methods we compare against as well as the depth of trees we construct, and the technical specifications of the machine on which the experiments were performed. Section 5.2 presents the results and analysis of the merging algorithm on artificially created Pareto Fronts, while section 5.3 the integration of the merging approach into the Optimal Decision Tree algorithm.

5.1 Experiment setup

In order to show the efficiency and scalability boost of our new approach and to answer our research question of how this method affects the runtime, we perform several experiments. First, we compare the performance of the merge step, described in Definition 2, between the normal Pareto Front approach and Minkowski Sum on artificially created Pareto Fronts of different sizes. Second, we also compare the performance of the merge step integrated into the Optimal Decision Tree algorithm against two state-of-the-art algorithms in this field, the STreeD implementation with the Pareto Front [10] and the Bi-objective MurTree [13], a specialized approach for optimizing bi-objective problems. As both STreeD implementations, with and without Minkowski Sums, are orders of magnitude faster than the Bi-objective MurTree, for clarity, we have decided to omit these results from the runtime performance tables.

The experiments run on a 3.2Ghz AMD Ryzen 7 5800HS with 16GB of RAM using only one thread. We test on a total of 21 binary classification datasets used in previous works [13] with the mean of five runs being recorded. We construct trees of depth 4 (see Table 2) and 5 (see Table 3), as for depth 3, the trees are too small to observe any difference between the approaches as well as the computation being done in the range of milliseconds.

5.2 Artificial Pareto Merging results

In the isolated case, running our Minkowski Sum method against the traditional Pareto Front approach yields improvements that are several orders of magnitude faster (see Table 1). This difference is given by the fact that our Minkowski Sum Algorithm manages to combine the two Pareto Fronts in linear time after sorting instead of quadratic, which has a massive impact, especially on bigger datasets.

Table 1: Runtime(ms) for merging artificially created Pareto Fronts of different sizes. Best results are shown in bold. Average of five runs.

Size	Pareto Front	Minkowski Sum
500	8	<**1**
1000	28	<**1**
5000	898	**1**
10000	3355	**2**
20000	13711	**4**

5.3 Tree Construction Results

Looking at the runtime performance presented in Tables 2 and 3, we can conclude that the STreeD approach with Minkowski Sums is 8% faster than the STreeD using Pareto Fronts and over 400% faster than the Bi-objective MurTree (computed with geometric mean [17]). This not only indicates that on average, our approach is faster than the previous implementation of STreeD, but looking at the performance on depth 5, we start to notice a much bigger improvement than on depth 4, of over 12%. Moreover, we can observe that a higher number of instances and a bigger imbalance in the dataset results in a bigger improvement for our approach since there would be more candidate solutions generated that would be closer to each other in terms of false positives and negatives and thus harder to prune away (for example *yeast* or *german-credit* datasets). Another interesting result to note is that our method performs worse than the Pareto Front approach when the number of features is higher than the number of instances, on the *ionosphere* dataset. This may be due to the relatively low number of instances which may produce lower-sized candidate solution sets that do not favour our approach, but more experiments need to be performed to confirm the validity of this theory.

Table 2: Runtime(s) for maximizing F1-score for two classes with trees of depth 4 with timeouts beyond 1200s denoted with '-'. Best results are shown in bold. Average of five runs.

Dataset	$\|\mathcal{D}\|$	$\|\mathcal{D}^+\|$	$\|\mathcal{D}^-\|$	$\|\mathcal{F}\|$	Minkowski Sum	Pareto Front
anneal	812	625	187	93	**1**	2
audiology	216	57	159	148	**3**	3
australian-credit	653	357	296	125	**13**	13
breast-wisconsin	683	444	239	120	**5**	5
diabetes	768	500	268	112	**8**	9
german-credit	1000	700	300	112	**23**	25
heart-cleveland	296	160	136	95	**4**	4
hepatitis	137	111	26	68	<1	**<1**
hypothyroid	3247	2970	277	88	**2**	2
ionosphere	351	225	126	445	**1142**	1163
kr-vs-kp	3196	1669	1527	73	**2**	2
letter	20000	813	19187	224	**271**	277
lymph	148	81	67	68	<1	**<1**
mushroom	8124	4208	3916	119	<1	<1
pendigits	7494	780	6714	216	152	**152**
primary-tumor	336	82	254	31	**<1**	<1
segment	2310	330	1980	235	**<1**	<1
soybean	630	92	538	50	<1	**<1**
tic-tac-toe	958	626	332	27	**<1**	<1
vote	435	267	168	48	<1	**<1**
yeast	1484	463	1021	89	**5**	6

6 Responsible Research

For our research to meaningfully contribute to the scientific community and the collective of human knowledge, we must adhere to the FAIR guidelines. This entails that the data that we use needs to be **Findable**, for which we provide proper documentation in terms of links or other papers. **Accessible**, this data can be viewed by anyone interested, which is achieved through a publicly available repository[1], without requiring any authentication. By using the common CSV format, we ensure that the data is **Interoperable** and can be used for any means. Lastly, **Reusability** is a very important aspect of research, and we provide instructions and comments meant to help understand and run the code as well as comprehensive descriptions of the setup process of our experiments to ensure reproducibility, including the code that was used to perform these experiments.

[1]As some parts of the codebase contain unpublished work, the repository will be made available at a later date.

Table 3: Runtime(s) for maximizing F1-score for two classes with trees of depth 5 with timeouts beyond 1200s denoted with '-'. Best results are shown in bold. Average of five runs.

Dataset	$\lvert\mathcal{D}\rvert$	$\lvert\mathcal{D}^+\rvert$	$\lvert\mathcal{D}^-\rvert$	$\lvert\mathcal{F}\rvert$	Minkowski Sum	Pareto Front
anneal	812	625	187	93	**20**	23
audiology	216	57	159	148	<1	**<1**
australian-credit	653	357	296	125	**333**	351
breast-wisconsin	683	444	239	120	**36**	39
diabetes	768	500	268	112	**184**	209
german-credit	1000	700	300	112	**822**	929
heart-cleveland	296	160	136	95	**72**	82
hepatitis	137	111	26	68	**<1**	<1
hypothyroid	3247	2970	277	88	**33**	36
ionosphere	351	225	126	445	998	**948**
kr-vs-kp	3196	1669	1527	73	**27**	42
letter	20000	813	19187	224	-	-
lymph	148	81	67	68	<1	**<1**
mushroom	8124	4208	3916	119	<1	<1
pendigits	7494	780	6714	216	**345**	347
primary-tumor	336	82	254	31	**<1**	<1
segment	2310	330	1980	235	**<1**	<1
soybean	630	92	538	50	**7**	9
tic-tac-toe	958	626	332	27	**3**	4
vote	435	267	168	48	**10**	10
yeast	1484	463	1021	89	**92**	122

7 Conclusions and Future Work

Non-linear metrics are an important tool to assess the performance of Machine Learning models on imbalanced datasets. Optimal Decision Trees is a model that can return the optimum value for a given objective and is more interpretable than standard Decision Trees. However, since constructing one is an NP-hard problem [5] and non-linear metrics add an additional layer of complexity, all approaches encountered so far lack scalability with respect to the size of datasets and trees computed. We have presented a novel merge algorithm that increases the performance of obtaining Optimal Decision Trees by using convex hulls to directly construct a set of candidate solutions from the sets of the left and right child. When incorporated into STreeD [10], a framework for calculating Optimal Decision Trees, we have obtained an almost 10% percent improvement on the solution using Pareto Fronts, the current state-of-the-art implementation. Therefore, we have managed to reach our scalability goal of making it more feasible to compute bigger Optimal Decision Trees, which will yield better misclassification scores. As a result, critical domains such as healthcare or the judicial system can make use of this advancement to make much more accurate decisions, achieving optimal results instead of just an approximation with standard Decision Trees while also being easier to understand due to the low number of decision nodes.

Moreover, the positive results of applying Minkowski Sums to optimize the F1-score may encourage the application of this method to other partially ordered tasks or metrics where the merging step represents a bigger bottleneck or when multiple dimensions are needed, as long as the two mentioned properties of monotonicity and convexity are satisfied.

References

[1] C. Rudin, "Stop explaining black box machine learning models for high stakes decisions and use interpretable models instead," *Nature Machine Intelligence*, pp. 206–215, 2019.

[2] L. H. Gilpin, D. Bau, B. Z. Yuan, A. Bajwa, M. Specter, and L. Kagal, "Explaining explanations: An overview of interpretability of machine learning," in *2018 IEEE 5th International Conference on Data Science and Advanced Analytics (DSAA)*, pp. 80–89, 2018.

[3] L. Breiman, J. Friedman, R. Olshen, and C. Stone, *Classification and Regression Trees.* Chapman and Hall/CRC, 1984.

[4] E. Demirović, E. H. Anna Lukina, J. Chan, J. Bailey, C. Leckie, K. Ramamohanarao, and P. J. Stuckey, "Murtree: Optimal decision trees via dynamic programming and search," *Journal of Machine Learning Research*, vol. 23(26), pp. 1–47, 2022.

[5] L. Hyafil and R. L. Rivest, "Constructing optimal binary decision trees is np-complete," *Information Processing letters*, pp. 15–17, 1976.

[6] D. Bertsimas and J. Dunn, "Optimal classification trees," *Mach Learn 106*, pp. 1039–1082, 2017.

[7] S. Verwer and Y. Zhang, "Learning optimal classification trees using a binary linear program formulation," in *Proceedings of AAAI-19*, pp. 1625–1632, 2019.

[8] H. Verhaeghe, S. Nijssen, G. Pesant, C.-G. Quimper, and P. Schaus, "Learning optimal decision trees using constraint programming," *Constraints 25*, pp. 226–250, 2020.

[9] Mikolá Janota and António Morgado, "Sat-based encodings for optimal decision trees with explicit paths," in *Proceedings of the International Conference on Theory and Applications of Satisfiability Testing*, pp. 501–518, 2020.

[10] J. G. M. van der Linden, M. M. de Weerdt, and E. Demirović, "Necessary and sufficient conditions for optimal decision trees using dynamic programming," in *Advances in NeurIPS-23*, pp. 9173–9212, 2023.

[11] G. Aglin, S. Nijssen, and P. Schaus, "Learning optimal decision trees using caching branch-and-bound search," in *Proceedings of the AAAI Conference on Artificial Intelligence*, pp. 3146–3153, 2020.

[12] D. R. Grimes, E. M. Corry, T. Malagón, C. O'Riain, E. L. Franco, and D. J. Brennan, "Challenges of false positive and negative results in cervical cancer screening," *medRxiv*, 2020.

[13] E. Demirović and P. J. Stuckey, "Optimal decision trees for nonlinear metrics," in *Proceedings of AAAI-21*, pp. 3733–3741, 2021.

[14] M. R. Garey, "Optimal binary identification procedures," *SIAM Journal on Applied Mathematics*, vol. 23, no. 2, pp. 173–186, 1972.

[15] M. Mampaey, S. Nijssen, A. Feelders, R. Konijn, and A. Knobbe, “Efficient algorithms for finding optimal binary features in numeric and nominal labeled data,” *Knowledge and Information Systems*, pp. 465–492, 2015.

[16] M. de Berg, O. Cheong, M. van Kreveld, and M. Overmars, *Computational Geometry: Algorithms and Applications.* Springer, 2008.

[17] P. J. Fleming and J. J. Wallace, “How not to lie with statistics: the correct way to summarize benchmark results,” *Communications of the ACM*, vol. 29, pp. 218–221, 1986.