Clustering data for the Optimal Classication Tree Problem

Zacharie Ales, Valentine Huré, Amélie Lambert

Clustering data for the Optimal Classification Tree Problem

Zacharie Ales[1,2], Valentine Huré[1], and Amélie Lambert[1]

[1]CEDRIC, Conservatoire National des Arts et Métiers, 75003 Paris, France
[2]UMA, ENSTA Paris, Institut Polytechnique de Paris, 91120 Palaiseau, France

Abstract

Solving the optimal classification tree problem enables to compute classifiers which are both interpretable and efficient. Most of the exact methods for this problem are based on on a Mixed Integer Linear Program (MILP) formulation. However, the efficiency of MILP solvers generally does not allow these formulations to be solved directly, once the dataset exceeds a critical size. To address this challenge, we propose in this paper an iterative exact algorithm than handles medium-sized datasets from the state-of-the-art. The basic idea is to start by solving a MILP formulation on a small subset of data points representative of the considered dataset. Then, the subset is iteratively extended until global optimality of the initial problem is reached.

A key feature is to compute relevant initial subsets of data points. For this, we introduce the concept of *data-partitions* and design several algorithms to compute them. We then define two MILP formulations to compute optimal classification trees on data-partitions. We prove that combining our iterative algorithm with our first formulation enables to obtain an optimal solution of the original problem. We also propose an alternative method based on the second formulation which is significantly faster.

We present extensive computational experiments to compare our algorithms with state-of-the-art approaches. We show that our methods constitute the best compromise between in-sample accuracy and interpretability.

1 Introduction

1.1 Related works

Classification trees are widely used machine learning models that stand out for their interpretability and solid performances. Indeed, due to their straightforward structure, predictions given by classification trees can be easily understood. More precisely, they are oriented trees where internal nodes separate data points using linear split functions, and terminal nodes assign labels to the data points that reach them. We focus in this article on binary trees which are the most commonly considered.

Building optimal binary classification trees is a well-known problem, which was proven to be NP-hard in 1976 [22]. One of the first algorithms designed was the greedy heuristic `CART` [7] that iteratively separates data points until a stopping criteria, such as a maximal depth or a minimal number of data reaching each leaf, is met. In `CART`, data points are separated according to Gini index but other algorithms such as `ID3` [30], C5.0 [31] use different impurity measures. These greedy heuristics have been improved upon by local search [9,14], or by more advanced greedy heuristics such as `GUIDE` [25,26] which is based on statistical tests. These approaches are quick and effective. However, to get good prediction performances, they tend to provide large and deep trees, thus reducing their interpretability. Heuristics using oblique hyperplane splits (see for instance [8,27,29,36]) can provide smaller trees, but at the cost of more computation time. Prediction performances of classification trees can be enhanced with ensemble methods such as `Random Forests` [6], `TreeBoost` [17], or `XGBoost` [10]. Such methods

clearly improve the performances of the predictions but, since the prediction involves several trees, interpretability is often lost.

Recently, exact solution methods for the construction of optimal classification trees that are based on Mixed-Integer Linear Programming (MILP) formulations were introduced [1, 3, 34]. Due to the significant improvements in the last decades of both the hardware computational power and the MILP solver performances [4], these MILPs can now be solved optimally for small-sized datasets. Such optimal trees tend to better highlight the underlying trends of datasets thus increasing the interpretability of the predictions. This is especially the case since these approaches generally limit the number of features involved in the separations to avoid overfitting [1, 3, 34]. They moreover reach the similar accuracy as `CART`, `ID3`, or `C5.0` heuristics but with significantly smaller trees. Furthermore, the versatility of mathematical programs eases the integration of fairness in the construction of classification trees [1,34]. Most of the MILP formulations involve binary variables that indicate which leaf reaches a data point, and thus their size increases linearly with that of the datasets. Consequently, they are often intractable for medium-sized datasets. Several approaches have been considered to tackle this issue. Some are based on the choice of a good formulation [1,34], while others focus on more specific datasets to speed-up the learning time. For instance, in [20] the authors focus on datasets with categorical features, and in [1,11,24] the dataset is restricted to binary features. More recently, non-linear models have also been proposed to compute optimal classification trees [2] or optimal randomized decision trees [5]. Finally, dynamic programming approaches were also proposed [11, 24].

Another approach to scale-up is to build heuristic algorithms based on MILP formulations. For example, in [15] a heuristic based on column generation is proposed. In [21], the resolution is speeded up by heuristically excluding parts of the search space. The authors of [12] consider SVM-separators at each node which aim to substitute binary variables by continuous ones. Finally, the method proposed in [41] selects a subset of data points for which the MILP formulation is solved. Following this latter idea, our goal in this paper is to improve the scalability of MILP formulations. For this, we reduce the size of the dataset by computing clusters of points that may follow the same path in an optimal classification tree. Then, we design an iterative algorithm that, starting from an initial clustering, solves a MILP formulation on the reduced dataset and updates the clustering for the next iterations until an optimal classification tree is found.

1.2 Our contributions

In this paper, we propose an iterative algorithm that computes optimal classification trees for medium-sized datasets. Starting from a recent efficient MILP formulation [2], our main idea is to first solve the MILP on a reduced dataset in order to fasten its resolution. Then, at each iteration, we increase the size of the dataset until optimality is proven. Note that the reduced dataset can be significantly smaller than the original one, thus enabling to handle larger datasets than with the original MILP. The efficiency of this method strongly depends on the choice of the reduced dataset. To determine a relevant one, we introduce the concept of *data-partition*. It corresponds to a partition of the initial dataset which additionally associates to each cluster a representative data and a label. We then design several algorithms to build good data-partitions which share similarities with hierarchical clustering algorithms. We adapt the formulation from [2] in two different ways to handle data-partitions. Finally, we propose an algorithm that iteratively increase the size of a data-partition by solving one of our two adapted formulations. We compare the performances of our approaches with that of state-of-the-art heuristics on medium-sized datasets of the literature.

To sum up, our contributions are:

1. The introduction of the concept of *data-partitions*, properties to describe it, and constructive algorithms.
2. Two MILP formulations that compute an optimal classification tree from a data-partition.
3. An iterative algorithm that provides classification trees using data-partitions. We prove that this algorithm provides optimal solutions when combined with one of our formulations.

4. Computational results showing that our algorithms achieve same or better performances than state-of-the-art methods while providing greater interpretability.

The paper is organized as follows. Section 2 presents a targeted review of the state of the art, recalling the MILP formulation used to compute optimal trees, as well as the main classical hierarchical clustering algorithms. In Section 3, we formally define the concept of *data-partition*. We then characterize several properties of data-partitions and we introduce three algorithms to compute them. In Section 4, we introduce two new MILP formulations that compute an optimal tree from a data-partition, and our new iterative algorithm that uses them to compute classification trees. We further prove that it reaches optimality when combined with one of our formulations. Finally, we present experimental results in Section 5 and Section 6 draws a conclusion.

2 Preliminaries

In this section, we present a targeted review of the literature. We start by presenting the MILP formulation introduced in [2] that we use in our main algorithm. Then, we recall the main principles of standard hierarchical clustering algorithms that create partitions of unlabeled data.

2.1 An optimal tree problem MILP formulation [2]

A classification tree is an oriented binary tree $T = (\mathcal{N} \cup \mathcal{L}, E)$ which associates a split function $f_t : \mathbb{R}^{|\mathcal{J}|} \to \{true, false\}$ to each of its internal node $t \in \mathcal{N}$ and a label $k \in \mathcal{K}$ to each of its leaves $\ell \in \mathcal{L}$. A data $i \in \mathcal{I}$ is a vector X_i of $|\mathcal{J}|$ features that is classified by following the path from the root to a leaf according to the split functions f_t. More precisely, a data i follows the left branch of node $t \in \mathcal{N}$ if $f_t(X_i)$ is true and the right branch otherwise. Its predicted label is the one associated with the leaf reached by data i. We consider linear split functions of the form $a^\intercal X_i < b$ with $a \in \mathbb{R}^{|\mathcal{J}|}$ and $b \in \mathbb{R}$, applied on real-valued datasets. Moreover, vectors a are restricted to be binary unit vectors leading to *axis-aligned* trees.

We now present the recent MILP formulation from [2] for computing an optimal classification tree. For a given depth δ, the tree is modeled by its sets $\mathcal{N}$ of $2^{\delta-1}$ nodes, and $\mathcal{L}$ of 2^d leaves. Each node $t \in \mathcal{N}$ separates data points according to a linear split function modeled by variables $a_{j,t}$ and b_t: data point $i \in \mathcal{I}$ goes through the left branch of t if $\sum_{j \in \mathcal{J}} a_{j,t} x_{i,j} < b_t$, otherwise it goes through its right branch. Note that here $a_{j,t}$ is a binary variable since we consider an axis-aligned model. Each leaf of the tree assigns a label to data points that reach it. The binary variable $c_{k,\ell}$ indicates if leaf ℓ assigns label k.

Since it may not be optimal to split data points at each node, we use binary variables d_t to indicate whether node $t \in \mathcal{N}$ is active or not and binary variables l_ℓ to indicate whether a leaf $\ell \in \mathcal{L}$ is reached by at least one data point. To track the path of data points in the tree, binary variables $z_{i,\ell}$ indicate whether data point i reaches leaf ℓ. Finally, the last variables of the formulation enable to count the number of misclassifications of the tree. Variable $\theta_{i,k,\ell}$ indicates whether data point i reaches leaf ℓ which is assigned label k. By summing variables θ for all leaves ℓ, all labels k and all data points with a label different from k, we obtain the number of misclassifications.

Given a maximal depth δ, and a sparsity parameter γ, formulation $(P_{\delta,\gamma})$ is the following:

$$
(P_{\delta,\gamma})\begin{cases}
\min\ f(\theta,d) = \displaystyle\sum_{\ell\in\mathcal{L}}\sum_{k\in\mathcal{K}}\sum_{i\in\mathcal{I}\setminus\mathcal{I}_k}\theta_{i,k,\ell} + \frac{1}{\gamma+1}\sum_{t\in\mathcal{N}} d_t & \\
\text{s.t.}\ \displaystyle\sum_{j\in\mathcal{J}} a_{j,t} = d_t & t\in\mathcal{N} \quad (1)\\
0 \le b_t \le d_t & t\in\mathcal{N} \quad (2)\\
d_t \le d_{a(t)} & t\in\mathcal{N}\setminus\{r\} \quad (3)\\
\displaystyle\sum_{t\in\mathcal{N}} d_t \le \gamma & \quad (4)\\
\displaystyle\sum_{k\in\mathcal{K}} c_{k,\ell} = l_\ell & \ell\in\mathcal{L} \quad (5)\\
\displaystyle\sum_{i\in\mathcal{I}} z_{i,\ell} \ge l_\ell & \ell\in\mathcal{L} \quad (6)\\
z_{i,\ell} \le l_\ell & \ell\in\mathcal{L}, i\in\mathcal{I} \quad (7)\\
l_\ell \le d_t & \ell\in\mathcal{L}, t\in A_L(\ell) \quad (8)\\
\displaystyle\sum_{\ell\in\mathcal{L}} z_{i,\ell} = 1 & i\in\mathcal{I} \quad (9)\\
\displaystyle\sum_{j\in\mathcal{J}} a_{j,t}(x_{i,j}+\mu_j-\mu^-) + \mu^- \le b_t + (1+\mu^+)(1-z_{i,\ell}) & i\in\mathcal{I}, \ell\in\mathcal{L}, t\in A_L(\ell) \quad (10)\\
\displaystyle\sum_{j\in\mathcal{J}} a_{j,t}x_{i,j} \ge b_t - (1-z_{i,\ell}) & i\in\mathcal{I}, \ell\in\mathcal{L}, t\in A_R(\ell) \quad (11)\\
\theta_{i,k,\ell} \ge c_{k,\ell} + z_{i,\ell} - 1 & i\in\mathcal{I}, k\in\mathcal{K}, \ell\in\mathcal{L} \quad (12)\\
\theta_{i,k,\ell} \ge 0 & i\in\mathcal{I}, k\in\mathcal{K}, \ell\in\mathcal{L} \quad (13)\\
a_{j,t}\in\{0,1\}, d_t\in\{0,1\} & t\in\mathcal{N}, j\in\mathcal{J} \quad (14)\\
c_{k,\ell}\in\{0,1\}, l_\ell\in\{0,1\}, z_{i,\ell}\in\{0,1\} & i\in I, k\in\mathcal{K}, \ell\in\mathcal{L} \quad (15)
\end{cases}
$$

where the first sum of the objective function is the number of misclassifications and the second is the number of nodes that applies a split, weighted by $\frac{1}{\gamma+1}$. Observe that Constraint (4) ensures that the second objective is always lower than 1. Thus, among all the trees that minimize the number of misclassifications, the problem returns an optimal solution that additionally has a minimal number of splits.

The first sets of constraints fix the structure of the tree. Constraints (1) and (2) ensure that the variables defining the split function of $t \in \mathcal{N}$ vanish when t is inactive. Constraints (3) inactivate all descendants of an inactived node. Constraints (5) ensure that one and only one label is assigned to each opened node. The second part of the model follows the path of the data in the tree. Constraints (6) and (7) link the value of variables $z_{i,\ell}$ and l_ℓ. Constraints (9) ensure that each data is assigned to one and only one leaf, and Constraints (10) - (11) that the path of each data is consistent with the split functions. Constraints (12) and (13) counts the number of misclassifications. Finally, Constraints (8) enables to strengthen the linear relaxation. Note that since the number of misclassifications is minimized, an optimal solution always assigns to a leaf ℓ the most represented label among the data reaching ℓ.

This formulation solves the optimal classification problem for a given value of the parameters (δ, γ). In practice, these parameters are tuned by solving the problem for several tuples (δ, γ) within a maximum depth δ_{MAX} and selecting the best tree according to a validation set, as introduced in [2].

2.2 Hierarchical clustering

Clustering aims to partition unlabeled data such that similar data are grouped together. A wide variety of approaches have been considered for this task depending on the data type and the notion of similarity considered. We refer the reader to [37, 38] for an in-depth surveys on the topic. In this paper, we design hierarchical clustering algorithms to cluster labelled datasets.

Hierarchical clustering considers an initial partition of the data and greedily either merge (*agglomerative* algorithms) or split (*divisive* algorithms) clusters until a stopping criteria is met. This criteria is often a given number of clusters that must be reached. In this article we focus on agglomerative approaches which, at each step, merge the pair of clusters that minimizes a given distance function based on the distance between the data points. One of the oldest agglomerative hierarchichal method is *single-linkage* [16] which associates to two clusters C_1 and C_2 a similarity equals to the minimal distance between one data point of each cluster (i.e., $\min_{i \in C_1, j \in C_2} d_{i,j}$). This method suffers from the *chaining effect* which occurs when chains of data points are merged together over long distances regardless of the homogeneity of their clusters. Indeed, since the similarity function of the single-linkage is local – as it only involves the similarity between two close data points – it does not take into account the shape of the clusters it merges often leading to poor performances. The complete-linkage method which considers the maximal distance instead of the minimal one does not have this drawback. However, it is highly sensitive to outliers. The average-linkage is a more robust approach in which the similarity function is equal to the average distance between pairs of data points from the two clusters (i.e., $\frac{1}{|C_1| \times |C_2|} \sum_{i \in C_1} \sum_{j \in C_2} d_{ij}$). Other classical distance functions have been defined such as the distance between the clusters centroids or the increase of the sum of the squared distances in the clusters [35].

More complex hierarchical algorithms have also been designed. For example, `CHAMELEON` [23] both take into account a proximity distance and an interconnectivity (i.e., homogeneity) measure in order to determine which clusters must be merged. In `CURE` [18] a cluster is represented by several selected data points and the choice of the representative points is adjusted dynamically to make it robust to noise and outliers. `BIRCH` [39] takes advantage of a tree data structure to efficiently merge clusters based on their size, the sum of their distances and their squared distances. This makes it highly effective on large datasets. In `ROCK` [19], the similarity between the clusters is based on neighbor points rather than distances. This makes it directly applicable to categorical data.

Even though, a wide variety of hierarchical clustering algorithms have been designed, their adaptation to cluster together labeled data that are likely to reach the same leaf of an optimal classification tree is not straightforward.

3 Data-partitioning of a dataset

As mentioned in the introduction, the size of formulation ($P_{\delta,\gamma}$) depends on the number of data points in the dataset, and solving it optimally becomes impracticable when the dataset has more than a few thousand points. This is mainly due to the set of variables and constraints following the path of each data points in the tree. To handle larger datasets, we propose to compute clusters of points that may follow the same path in an optimal tree. Then, in the resulting formulation, we only follow the path of one point per cluster called the *representative* of the cluster. As a result, we only have one set of variables per cluster, instead of one set per data point, which enables to significantly reduce the size of the MILP. In Section 3.1, we start by defining the concept of *data-partition* and we characterize several properties of data-partitions. In Section 3.2, we introduce 3 data-partitioning algorithms.

3.1 Definition and properties of data-partitions

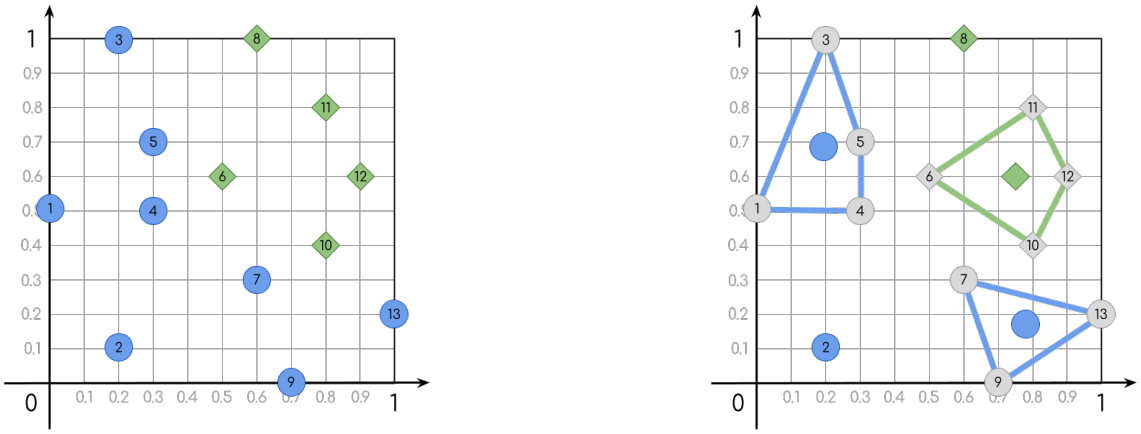
We now formally define the concept of data-partitioning.

Definition 1. *A data-partition of $\mathcal{I} = (X, Y)$ is a triplet $(\mathcal{P}, \tilde{X}, \tilde{Y})$ such that:*

- $\mathcal{P} = \{p_g\}_{g=1,...,|\mathcal{P}|}$ *is a partition of* $\mathcal{I}$*;*
- $\tilde{X} \in [0,1]^{|\mathcal{P}| \times |\mathcal{J}|}$ *is a set of representatives where* $\tilde{X}_g \in [0,1]^{|\mathcal{J}|}$ *is the representative of cluster* p_g*;*
- $\tilde{Y} \in \mathcal{K}^{|\mathcal{P}|}$ *is a set of labels where* $\tilde{y}_g$ *is the label of cluster* p_g*.*

To illustrate Definition 1, we consider the data-partition presented in Example 1.

Example 1. *Figure 1 represents a dataset* $\mathcal{I} = (X, Y)$ *of 13 points, 2 features, and 2 labels (Figure 1a) and a data-partition* $(\mathcal{P}, \tilde{X}, \tilde{Y})$ *with five clusters (Figure 1b). The symbol of each data corresponds to its class, and each cluster of the partition* $\mathcal{P} = \{\{1,3,4,5\},\{2\},\{6,10,\ 11,12\},\{7,9,13\},\{8\}\}$ *is represented by the convex hull of its data points and by its representative in color. In this example, we use barycenters as representatives. Since each cluster has data points of the same label that is thus also the label of the cluster.*



(a) Dataset of 13 data points

(b) Example of data-partition

Figure 1: Example of data-partition

When using a data-partition in a MILP formulation, we aim to replace all the points contained in one cluster by its representative in order to reduce the size of the formulation. Building a "good" data-partition is thus a trade-off between two conflicting goals: minimizing the number of clusters and ensuring that all points follow the path of their representative in the computed tree. In order to determine relevant data-partitions, we start by introducing 3 properties of data-partitions that are likely to be satisfied when the second goal is optimized. We also define a metric for each property that measures how close a data-partition is to satisfy it.

Let $G : \mathcal{I} \to \{1, ..., |\mathcal{P}|\}$ be the function that maps each data point to its cluster index, i.e. $\forall i \in \mathcal{I}, i \in p_{G(i)}$. The first property is that each cluster only contains data points of the same label.

Property 1 (Homogeneity)**.** *We say that a data-partition* $(\mathcal{P}, \tilde{X}, \tilde{Y})$ *of* $\mathcal{I} = (X, Y)$ *is* ***homogeneous*** *if and only if* $\forall i \in \mathcal{I}, y_i = \tilde{y}_{G(i)}$*.*

The metric associated with homogeneity is the proportion of data points of $\mathcal{I}$ that belong to a cluster of the same label.

Definition 2 (Measure of homogeneity)**.**

$$H(\mathcal{P}, \tilde{X}, \tilde{Y}) = \frac{|\{i \in \mathcal{I} \text{ s.t. } y_i = \tilde{y}_{G(i)}\}|}{|\mathcal{I}|}$$

Note that data-partition $(\mathcal{P}, \tilde{X}, \tilde{Y})$ is homogeneous if and only if $H(\mathcal{P}, \tilde{X}, \tilde{Y}) = 1$.

The second property characterizes the fact that the convex hulls of clusters should not intersect. Indeed, if two clusters have intersecting convex-hull they cannot be separated by a linear split function. Therefore, these clusters should either be merged since they cannot be properly separated by the tree or split into none-intersecting that may follow distinct paths in the tree.

Property 2 (Exclusion)**.** *We say that a data-partition* $(\mathcal{P}, \tilde{X}, \tilde{Y})$ *of* $\mathcal{I} = (X, Y)$ *is* ***exclusionary*** *if and only if* $\forall p_a \neq p_b \in \mathcal{P}^2, Conv(\{X_i, i \in p_a\}) \cap Conv(\{X_i, i \in p_b\}) = \emptyset$

To measure the exclusion of a data-partition, we take the proportion of clusters whose convex hull is not intersected by the convex hull of another cluster.

Definition 3 (Measure of exclusion)**.**

$$E(\mathcal{P}, \tilde{X}, \tilde{Y}) = \frac{|\{p_g \in \mathcal{P} \text{ s.t. } \forall g' \neq g, Conv(\{X_i,\ i \in p_g\}) \cap Conv(\{X_i,\ i \in p_{g'}\}) = \emptyset\}|}{|\mathcal{P}|}$$

Note that a data-partition $(\mathcal{P}, \tilde{X}, \tilde{Y})$ is exclusionary if and only if $E(\mathcal{P}, \tilde{X}, \tilde{Y}) = 1$. For the final property of data-partition, we notice that, often, when data points of different labels are close to each other, they may also be close to a split function. Therefore, clustering those data points with others may deteriorate the placement of the split function. Thus, we identify data points that are close to data points whose closest neighbor is of different label and consider that they should be alone in their clusters (i.e. in singletons). Let us denote by $n(i)$ the closest neighbor of the point $i \in \mathcal{I}$ i.e. $n(i) = \arg\min_{i' \in \mathcal{I} \setminus \{i\}} ||X_i - X_{i'}||_2$.

Property 3 (Consistency)**.** *We say that a data-partition* $(\mathcal{P}, \tilde{X}, \tilde{Y})$ *of* $\mathcal{I} = (X, Y)$ *is* ***consistent*** *if for all* $i \in \mathcal{I}$ *such that* $|p_{G(i)}| > 1$, $y_i = y_{n(i)}$.

We measure the consistency by the proportion of points belonging to clusters of size more than 1 whose closest neighbor is of a different class.

Definition 4 (Measure of consistency)**.**

$$C(\mathcal{P}, \tilde{X}, \tilde{Y}) = 1 - \frac{|\{i \in \mathcal{I} \text{ s.t. } |p_{G(i)}| > 1 \text{ and } y_{n(i)} \neq y_i\}|}{|\{i \in \mathcal{I} \text{ s.t. } y_{n(i)} \neq y_i\}|}$$

If $\{i \in \mathcal{I} \text{ s.t. } y_{n(i)} \neq y_i\}$ *is empty, we define* $C(\mathcal{P}, \tilde{X}, \tilde{Y}) = 1$.

Note that a data-partition $(\mathcal{P}, \tilde{X}, \tilde{Y})$ is consistent if and only if $C(\mathcal{P}, \tilde{X}, \tilde{Y}) = 1$.

Our aim is now to build data-partitions of minimum size that also maximize the homogeneity, exclusion and consistency metrics. In the rest of the article, we call *size reduction metric* the following ratio $R = \frac{|\mathcal{I}| - |\mathcal{P}|}{|\mathcal{I}|}$.

As an example, the data-partition represented in Figure 1b is homogeneous and exclusionary but not consistent (since data point 6 nearest neighbor is not of the same class).

3.2 Data-partitioning algorithms

The computation of a data-partition from an initial dataset requires to cluster data points. Since the data are labeled, the direct use of unsupervised clustering methods is likely to lead to poor results in terms of homogeneity, exclusion and consistency. To address this issue we design agglomerative hierarchical clustering methods since this framework enables us to partially take into account these three properties. All our algorithms start with a partition of $|\mathcal{I}|$ singletons and merge clusters iteratively.

Our first algorithm called `GreedyPartitioning (GP)` is sketched up in Algorithm 1. Let $\bar{\mathcal{I}} = \{(i,j) \in \mathcal{I}^2 : i < j\}$, we denote by D the list of pairs of points of $\bar{\mathcal{I}}$ ordered by increasing distance:

$$D = \left\{(i,j) \in \bar{\mathcal{I}} \, : \, \forall [(i,j),(k,l)] \in \bar{\mathcal{I}}^2, \, (i,j) < (k,l) \Leftrightarrow ||X_i - X_j||_2 < ||X_k - X_l||_2 \right\}$$

The first loop iterates on the pairs of points (i,j) of the input dataset $\mathcal{I}$ following the order of list D. Moreover, the clusters of i and j are merged only if i and j have the same label. We iterate until at most $\rho|\mathcal{I}|$ clusters are obtained with $\rho \in]0,1[$. Thus, the size reduction metric R of the provided data-partition is at least $1-\rho$.The second loop iterates on all the clusters of the partition in order to assign a representative and a label to each cluster.

Algorithm 1: `GreedyPartitioning`$(\mathcal{I} = (X,Y), D, \rho)$

$\mathcal{P} \leftarrow \{\{i\} \; \forall i \in \mathcal{I}\}$
for $(i,j) \in D$ **do**
  if $y_i = y_j$ **then**
    Merge in P the clusters of i and j
  if $\frac{|P|}{|\mathcal{I}|} \leq \rho$ **then**
    break
for $g \in |\mathcal{P}|$ **do**
  $\tilde{X}_g \leftarrow \left(\frac{1}{|p_g|} \sum_{i \in p_g} x_{i,j}\right)_{j \in \mathcal{J}}$ // the representative is the barycenter
  $\tilde{y}_g \leftarrow \arg\max_{i \in p_g} y_i$ // a cluster is assigned its most represented label
return $(\mathcal{P}, \tilde{X}, \tilde{Y})$

Since Algorithm 1 is similar to single-linkage clustering, it has similar properties i.e. a small time complexity but also its chaining effect [32] mentioned in Section 2.2 which can cause non-exclusionary in data-partitions.

By construction, Algorithm 1 always provides a data-partition that is homogeneous but not necessarily exclusionary or consistent.

Algorithm 1 depends on the choice of ρ that is not straightforward. Thus, we propose the `ImprovedGreedyPartitioning (IGP)` algorithm (sketchep up in Algorithm 2) that does not use a parameter as a stopping criteria. Its basic idea is to prioritize the merging of data points that do not have points of a different label close to them. For this, instead of iterating on pairs (i,j) of the ordered set D, the pairs are ordered by increasing ratio between the distance from i to j and the distance to the closest data point of different label. We thus introduce, $\forall i \in \mathcal{I}$, $d(i)$ the distance of the closest point of different label,

$$d(i) = \min_{j \in \mathcal{I} \setminus \{i\}, \text{ s.t. } y_i \neq y_j} ||X_i - X_j||_2$$

and D' the list of pair of points of $\mathcal{I}$ following the order described above:

$$D' = \left\{(i,j) \in \bar{\mathcal{I}} \, : \, \forall [(i,j),(k,l)] \in \bar{\mathcal{I}^2} \, , \, (i,j) < (k,l) \Leftrightarrow \frac{||X_i - X_j||_2}{\min(d(i),d(j))} \leq \frac{||X_k - X_l||_2}{\min(d(k),d(l))} \right\}$$

The first loop of the algorithm iterates on the pairs of points (i,j) of the input dataset $\mathcal{I}$ following the order of list D'. As in Algorithm 1, we only merge pairs of points of same label to compute a data-partition that satisfies Property 1 of homogeneity. To build a data-partition as exclusionary as possible (Property 2) Then, to prevent clusters from having intersecting convex hulls, we add a merging condition: we merge two clusters only if their medoids are closer to each other than a point of different label (i.e. if $\frac{||X_{m_{G(i)}} - X_{m_{G(j)}}||_2}{\min(d(m_{G(i)}),d(m_{G(j)})} < 1$). We use medoids to represent clusters to reduce

computations. Using the barycenter b_g would require to compute $d(b_g)$ each time a cluster is created or merged. Instead, we consider its medoid $m_g = \arg\min_{i \in g} \sum_{j \in g} ||X_i - X_j||_2$ since $d(m_g)$ has already been computed to obtain D'. Finally, we consider only the pairs of D' whose ratio is less than 1 in order to not merge data points whose closest neighbor is of a different label. Doing this ensures that the computed data-partition satisfies Property 3 of consistency. Here again, the second loop iterates on all the clusters of the partition in order to assign a representative and a label to each cluster.

Algorithm 2: `ImprovedGreedyPartitioning` $(\mathcal{I} = (X, Y), d, D')$

$\mathcal{P} \leftarrow \{\{i\}\ \forall i \in \mathcal{I}\}$
for $(i, j) \in D'$ **do**
  $m_{G(i)}, m_{G(j)} \leftarrow$ the medoids of the cluster of i and j
  if $y_i = y_j$ ***and*** $\frac{||X_{m_{G(i)}} - X_{m_{G(j)}}||_2}{\min(d(m_{G(i)}), d(m_{G(j)}))} < 1$ **then**
    Merge in $\mathcal{P}$ the clusters of i and j
  if $\frac{||X_i - X_j||_2}{\min(d(i), d(j))} \geq 1$ **then**
    break
for $g \in |\mathcal{P}|$ **do**
  $\tilde{X}_g \leftarrow \left(\frac{1}{|p_g|} \sum_{i \in p_g} x_{i,j}\right)_{j \in \mathcal{J}}$ // the representative is the barycenter
  $\tilde{y}_g \leftarrow$ Class of the data points in p_g
return $(\mathcal{P}, \tilde{X}, \tilde{Y})$

By construction, Algorithm 2 always computes homogeneous and consistent data-partitions. However, exclusion is not guaranteed.

Finally, we introduce the `ClosestRepresentativeMerge (CRM)` summed up in Algorithm 3. Our idea here is to iterate on pairs of *representatives* of clusters instead on pairs of data points. This requires to compute the representative of each cluster as soon as two clusters are merged. One can use different methods to compute them, here we use medoids.

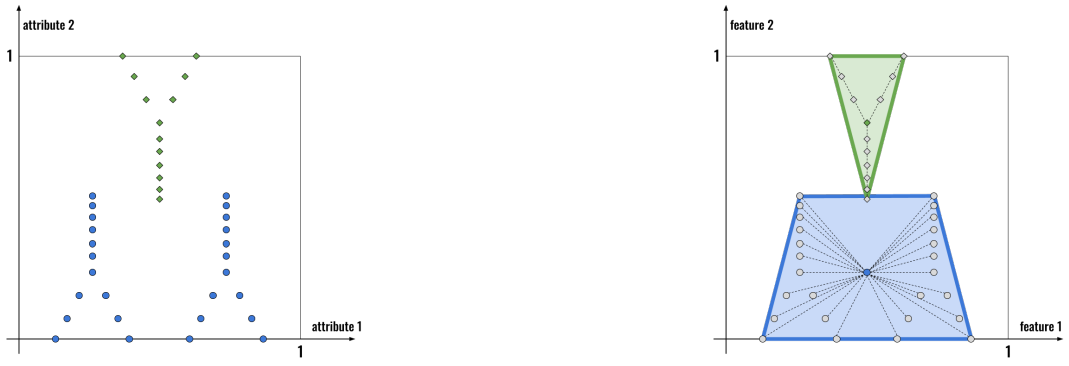
Starting from the partition made of singletons of set $\mathcal{I}$, we iterate following the increasing order of the distances D_k between the representatives of the clusters. This list is initially equal to D and is updated into D_k at each iteration k. As in Algorithms 1 and 2, we only merge pairs of clusters of same label to compute a data-partition that satisfies Property 1 of homogeneity, giving us directly the label of the cluster. Moreover, we want to build a data-partition as exclusionary as possible (Property 2). For this, we maintain a taboo set called L that contains the clusters that cannot be merged anymore: two clusters g_1 and g_2 are added to L if their representatives are the closest and if they have different labels.

Algorithm 3: `ClosestRepresentativesMerge`$(\mathcal{I} = (X, Y), D)$

$\mathcal{P} \leftarrow \{\{i\}\ \forall i \in \mathcal{I}\}$
$\tilde{X} \leftarrow X$
$\tilde{Y} \leftarrow Y$
$D_0 \leftarrow D$ // pair of clusters of $\mathcal{P}$
$L \leftarrow \emptyset$ // taboo set of clusters that cannot be merged anymore
while $|D_k| > 0$ *and* $|L| < |\mathcal{P}|$ **do**
  $(g_1, g_2) \leftarrow$ the first pair of D_k // the pair of cluster whose representatives are the closest
  $D_k \leftarrow D_k \setminus \{(g_1, g_2)\}$
  if $\tilde{y}_{g_1} \neq \tilde{y}_{g_2}$ **then**
    $L \leftarrow L \cup \{g_1, g_2\}$
  else
    if g_1 *and* $g_2 \notin L$ **then**
      $g_1 \leftarrow g_1 \cup g_2$
      $\mathcal{P} \leftarrow \mathcal{P} \backslash \{g_2\}$
      $\tilde{X}_{g_1} \leftarrow$ medoid of g_1
      $D_{k+1} \leftarrow D_k$ in which the pairs involving g_2 are removed and the order is updated according to $\tilde{X}_{g_1}$
  $k \leftarrow k + 1$
return $(\mathcal{P}, \tilde{X}, \tilde{Y})$

While Algorithm 3 aims at providing data-partitions that are homogeneous, exclusionary and consistent, only homogeneity is guaranteed by construction. Examples 2 and 3 are counter-examples for exclusion and consistency for Algorithm 3, respectively.

Example 2 (Non-exclusion)**.** *We consider a two-dimensional dataset with 2 labels depicted in Figure 2a. It contains 3 identical Y-shaped sets of data points such that the distance between successive points (on its 3 branches) decreases from the center to the extremities. It ensures that points at the bottom of the foot of each Y shapes are merged first. The data-partition provided by Algorithm 3 with barycenters as representatives is presented in Figure 2b. Clusters are plotted by their convex hulls and their representative is colored. We see that all data points of class circle and square are each merged in one cluster. The data-partition is non-exclusionary since these clusters have intersecting convex-hulls. Note that the same result would be obtained if the representatives were medoids instead of barycenters.*



(a) Dataset

(b) Data-partition provided by Algorithm 3

Figure 2: Dataset that leads to a non-exclusionary output with Algorithm 3

Example 3 (Non-consistency). *We consider the 2-dimensional dataset presented in Figure 3a. The second attribute is equal to* 0.5 *for all data points. The data-partition presented in Figure 3b is computed by Algorithm 3 with medoids as representatives. The data-partition has 2 clusters. Data points of a same cluster are linked by a dotted line. Data points of* $\mathcal{I}$ *are greyed and representatives are colored. Note that in the left cluster, both points are half-colored, it indicates that the representatives can be either of them. As* B *is closer to* C *than to* A*, the partition* $\{\{A, B\}, \{C, D, E\}\}$ *is not consistent. Note that Algorithm 3 with barycenters as representatives would also provide a non-consistent data-partition for this dataset.*

(a) Dataset

(b) Data-partition provided by Algorithm 3

Figure 3: Dataset that leads to a non-consistent output with Algorithm 3

We now present how any data-partition provided by one of these algorithms can be leveraged in a MILP to solve the classification tree problem.

4 Building optimal classification trees with data-partitions

In the previous section, we introduced the concept of *data-partition* that allows to cluster a dataset. To provide better classification trees than state-of-the-art MILP on medium and large datasets, we first adapt $(P_{\delta,\gamma})$ to take as an input a data-partition rather than a dataset. This new formulation allows us to fastly compute feasible classification trees for our original problem. Then, in order to improve the obtained solutions, we introduce the `Heuristic Iterative Algorithm (HIA)`, where at each step we refine the data-partition by splitting some of its clusters. We further show that `HIA` does not necessarily lead to an optimal classification tree for the original dataset. Then, in order to reach global optimality, we introduce a new model where we do not set *a priori* the representatives of the clusters, but we instead compute them within the optimization process. We further prove that embedding this new, but larger, formulation within our iterative framework allows us to compute an optimal solution for the original dataset. We call this approach `Exact Iterative Algorithm (EIA)`. Finally, we present a post-processing algorithm that speeds up both `HIA` and `EIA` methods.

4.1 A Heuristic Iterative Algorithm to compute classification trees

We start by adapting formulation $(P_{\delta,\gamma})$ (presented in Section 2.1) to compute a classification tree from a data-partition $(\mathcal{P}, \tilde{X}, \tilde{Y})$. The basic idea is to define variables z and θ for each cluster $g \in \{1, ..., |\mathcal{P}|\}$ instead of for each data $i \in \mathcal{I}$. Then, in the objective, we weight variables $\theta_{g,k,\ell}$ by $|p_g|$, the size of cluster g. We obtain formulation $(DP_{\delta,\gamma})$:

$$
(DP_{\delta,\gamma})\begin{cases}
\min\ \tilde{f}(\theta,d) = \displaystyle\sum_{\ell\in\mathcal{L}}\sum_{k\in\mathcal{K}}\sum_{\substack{g\in\{1,\dots,|\mathcal{P}|\}\\ s.t.\,\tilde{Y}_g\neq k}} |p_g|\theta_{g,k,\ell} + \frac{1}{\gamma+1}\sum_{t\in N} d_t & & \\
\text{s.t. } (1)-(5),\ (8),\ (14) & & \\
\quad \displaystyle\sum_{g\in\{1,\dots,|\mathcal{P}|\}} z_{g,\ell} \geq l_\ell & \ell\in\mathcal{L} & (16)\\
\quad z_{g,\ell} \leq l_\ell & \ell\in\mathcal{L}, g\in\{1,\dots,|\mathcal{P}|\} & (17)\\
\quad \displaystyle\sum_{\ell\in\mathcal{L}} z_{g,\ell} = 1 & g\in\{1,\dots,|\mathcal{P}|\} & (18)\\
\quad \displaystyle\sum_{j\in\mathcal{J}} a_{j,t}(\tilde{x}_{g,j}+\mu_j-\mu^-)+\mu^- \leq b_t + (1+\mu^+)(1-z_{g,\ell}) & g\in\{1,\dots,|\mathcal{P}|\}, \ell\in\mathcal{L}, t\in A_L(\ell) & (19)\\
\quad \displaystyle\sum_{j\in\mathcal{J}} a_{j,t}\tilde{x}_{g,j} \geq b_t - (1-z_{g,\ell}) & g\in\{1,\dots,|\mathcal{P}|\}, \ell\in\mathcal{L}, t\in A_R(\ell) & (20)\\
\quad \theta_{g,k,\ell} \geq c_{k,\ell} + z_{g,\ell} - 1 & g\in\{1,\dots,|\mathcal{P}|\}, k\in\mathcal{K}, \ell\in\mathcal{L} & (21)\\
\quad \theta_{g,k,\ell} \geq 0 & g\in\{1,\dots,|\mathcal{P}|\}, k\in\mathcal{K}, \ell\in\mathcal{L} & (22)\\
\quad c_{k,\ell}\in\{0,1\}, l_\ell\in\{0,1\}, z_{g,\ell}\in\{0,1\} & g\in\{1,\dots,|\mathcal{P}|\}, k\in\mathcal{K}, \ell\in\mathcal{L} & (23)
\end{cases}
$$

Note that similar adaptations can be straightforwardly obtained for other formulations of the optimal classification tree problem, such as those of [2, 3].

The main difference between models $(P_{\delta,\gamma})$ and $(DP_{\delta,\gamma})$ comes from the data points they consider. Obviously, when the data-partition considered in $(DP_{\delta,\gamma})$ is the original dataset $\mathcal{I}$, both problems have the same optimal value, i.e. $f(\theta,d)=\tilde{f}(\theta,d)$. However, this is not the case for most data-partitions of $\mathcal{I}$, and even for a given tree T (i.e. for given values of θ and d), we may have $f(\theta,d)\neq\tilde{f}(\theta,d)$. This is because variables θ are weighted by $|p_g|$ in $\tilde{f}$, assuming that all the data points of a cluster p_g reach the same leaf than their representative, which is not always true.

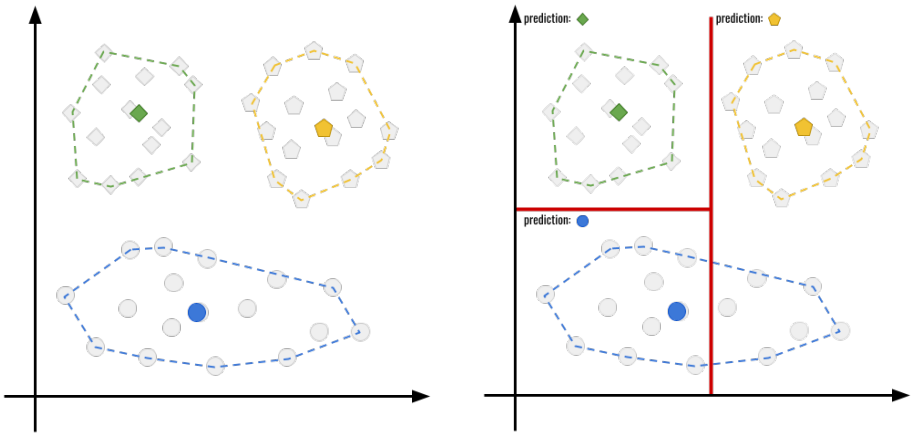
In the following we focus on the comparison of the objective value of problems $(P_{\delta,\gamma})$ and $(DP_{\delta,\gamma})$, i.e. of the values of $f(\theta,d)$ and $\tilde{f}(\theta,d)$ for a given tree. Both functions are a sum of two terms, and for a same solution (i.e. values of θ and d), only the value of the first term (the number of misclassified data) are different. To simplify the writing, we use the following notations in the rest of the paper:

- $f(\theta,d) = E(T) + \frac{1}{\gamma+1}\sum_{t\in\mathcal{N}} d_t$, where $E(T) = \displaystyle\sum_{\ell\in\mathcal{L}}\sum_{k\in\mathcal{K}}\sum_{i\in\mathcal{I}\setminus\mathcal{I}_k}\theta_{i,k,\ell}$.
- $\tilde{f}(\theta,d) = \tilde{E}(T) + \frac{1}{\gamma+1}\sum_{t\in N} d_t$, where $\tilde{E}(T) = \displaystyle\sum_{\ell\in\mathcal{L}}\sum_{k\in\mathcal{K}}\sum_{\substack{g\in\{1,\dots,|\mathcal{P}|\}\\ s.t.\,\tilde{Y}_g\neq k}} |p_g|\theta_{g,k,\ell}$.

In order to compare the values of f and $\tilde{f}$, we start by showing through Examples 4 and 5 that the values of $E(T)$ and $\tilde{E}(T)$ are incomparable.

Example 4 (An instance where $\tilde{E}(T) < E(T)$)**.** *We consider the dataset $\mathcal{I}=(X,Y)$ with 46 data points, 2 features and 3 labels. Figure 4a depicts a data-partition $(\mathcal{P},\tilde{X},\tilde{Y})$ with 3 clusters, one for each label. The representative of each cluster is the barycenter of its data points. In Figure 4b, we plot the two split functions of an optimal tree T on $(\mathcal{P},\tilde{X},\tilde{Y})$ obtained by solving $(DP_{2,2})$. The vertical line corresponds to the split function of the root node while the horizontal line corresponds to the split function of its left child. For each leaf, its predicted label is indicated in the top-left corner. Since each representative of the data-partition reaches a different leaf, $\tilde{E}(T)$ is equal to 0. However, applying T*

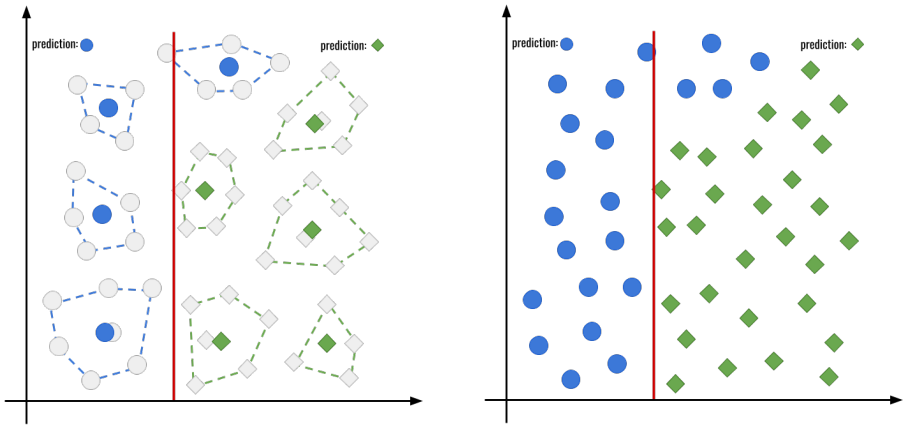
to the original dataset leads to $E(T) = 6$ since 6 samples of class circle do not follow the path of their representative. Therefore, $\tilde{E}(T) < E(T)$.



(a) A data-partition of $\mathcal{I}$ (b) An optimal tree T for $(\mathcal{P}, \tilde{X}, \tilde{Y})$

Figure 4: An example where $\tilde{E}(T) < E(T)$ ($\delta = 2$ and $\gamma = 2$)

Example 5 (An instance where $E(T) < \tilde{E}(T)$)**.** *We consider the dataset $\mathcal{I} = (X, Y)$ with 50 data points, 2 features and 2 labels and the data-partition $(\mathcal{P}, \tilde{X}, \tilde{Y})$ plotted in Figure 5a with 9 clusters. In Figure 5a, we also plot a red line which corresponds to the optimal tree T on $(\mathcal{P}, \tilde{X}, \tilde{Y})$ obtained by solving $(DP_{1,1})$. Tree T only misclassifies one representative of a cluster of size 5, thus $\tilde{E}(T) = 5$. This is greater than $E(T) = 4$ as we can see in Figure 5b.*



(a) A data-partition of $\mathcal{I}$ (b) An optimal tree T for $(\mathcal{P}, \tilde{X}, \tilde{Y})$

Figure 5: An example where $E(T) < \tilde{E}(T)$ ($\delta = 1$ and $\gamma = 1$).

In the following, we introduce the *intersection* property in order to characterize the trees T for which equality $\tilde{E}(T) = E(T)$ holds.

Property 4 (Intersection of a tree with a data-partition)**.** *We say that a tree T* ***intersects*** *a data-partition $(\mathcal{P}, \tilde{X}, \tilde{Y})$ if there exists a data point that follows a different path in T than its representative.*

Proposition 1. *Let $\mathcal{I} = (X, Y)$ be a dataset and $(\mathcal{P}, \tilde{X}, \tilde{Y})$ be a data-partition of $\mathcal{I}$. If a tree T does not intersect $(\mathcal{P}, \tilde{X}, \tilde{Y})$, then $E(T) = \tilde{E}(T)$.*

Proof. If T does not satisfy Property 4 with respect to $(\mathcal{P}, \tilde{X}, \tilde{Y})$ it means that every data point follow the same path than their representatives, and we thus have $E(T) = \sum_{\ell \in \mathcal{L}} \sum_{k \in \mathcal{K}} \sum_{i \in \mathcal{I} \setminus \mathcal{I}_k} \theta_{i,k,\ell} = \sum_{\ell \in \mathcal{L}} \sum_{k \in \mathcal{K}} \sum_{\substack{g \in \{1,\dots,|\mathcal{P}|\} \\ s.t. \tilde{Y}_g \neq k}} |p_g| \theta_{g,k,\ell} = \tilde{E}(T)$. □

From Proposition 1, we can easily characterize solutions of $(DP_{\delta,\gamma})$ for which $f(\theta, d) = \tilde{f}(\theta, d)$.

Corollary 1. *Given a dataset $\mathcal{I} = (X, Y)$ and $(\mathcal{P}, \tilde{X}, \tilde{Y})$ a data-partition of $\mathcal{I}$. For any feasible solution T of $(DP_{\delta,\gamma})$ for the data-partition $(\mathcal{P}, \tilde{X}, \tilde{Y})$ that does not satisfy the intersection property, then $f(\theta, d) = \tilde{f}(\theta, d)$.*

Corollary 1 provides a sufficient condition for which the value of the solution induced by a tree T is the same for both formulations $(P_{\delta,\gamma})$ and $(DP_{\delta,\gamma})$. However, there is no guarantee that there exists an optimal solution of $(DP_{\delta,\gamma})$ satisfying this condition. This is why we propose the generic framework `IterativeRefinement (ItR)` (sketched up in Algorithm 4) that iteratively solves a model (M) (here $(DP_{\delta,\gamma})$) for increasing sized data-partitions $(\mathcal{P}, \tilde{X}, \tilde{Y})$ of the data set (X, Y). Basically, at each iteration k, the clusters intersected by T_k, the current optimal solution of (M), are split. The algorithm stops when the current optimal solution T_k does not intersect the current data-partition $(\mathcal{P}, \tilde{X}, \tilde{Y})_k$.

Algorithm 4: `IterativeRefinement`$((X,Y),(\mathcal{P}, \tilde{X}, \tilde{Y}),(M))$

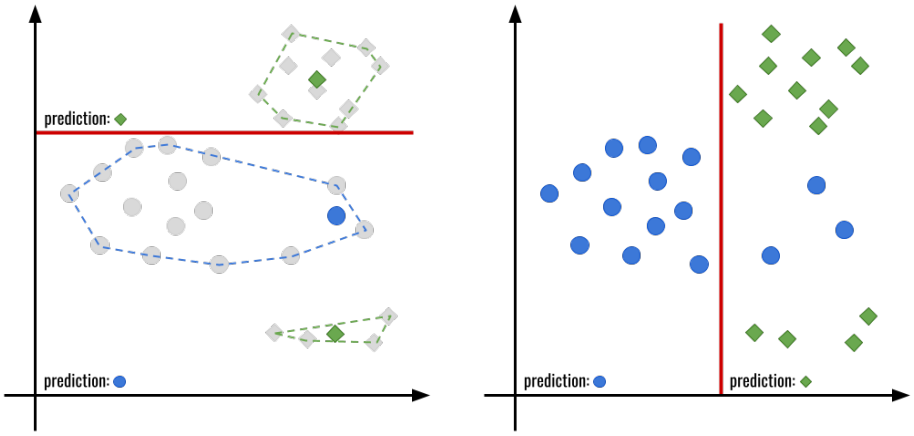
$(\mathcal{P}, \tilde{X}, \tilde{Y})_0 \leftarrow (\mathcal{P}, \tilde{X}, \tilde{Y})$
$T_0 \leftarrow$ optimal solution of (M) for $(\mathcal{P}, \tilde{X}, \tilde{Y})_0$
$k \leftarrow 0$
while T_k *intersects* $(\mathcal{P}, \tilde{X}, \tilde{Y})_k$ **do**
  Split clusters of $(\mathcal{P}, \tilde{X}, \tilde{Y})_k$ to create $(\mathcal{P}, \tilde{X}, \tilde{Y})_{k+1}$ that is not intersected by T_k
  $T_{k+1} \leftarrow$ optimal solution of (M) for $(\mathcal{P}, \tilde{X}, \tilde{Y})_{k+1}$
  $k \leftarrow k + 1$
end
Return T_k

Proposition 2. *In algorithm `IterativeRefinement` the number of loop iteration is polynomial in the size of the dataset $|\mathcal{I}|$. Moreover, it always converges to a tree T_k that does not intersect the current data-partition $(\mathcal{P}, \tilde{X}, \tilde{Y})_k$.*

Proof. In the worst case, we start with a data partition $(\mathcal{P}, \tilde{X}, \tilde{Y})_0$ composed of one cluster of size $|\mathcal{I}|$. If the initial tree T_0 does not intersect $(\mathcal{P}, \tilde{X}, \tilde{Y})_0$ we are done. Otherwise, since at each iteration we split at least one cluster, we iterate at most $|\mathcal{I}|$ times to get a data-partition having $|\mathcal{I}|$ clusters of size one. In this case we have $\tilde{X} = X$, i.e. the current data-partition is the initial dataset $\mathcal{I}$ where each data is its own representative. Thus, it cannot be intersected by T_k. □

We now focus on the case where (M) is $(DP_{\delta,\gamma})$ and call Algorithm 4 applied to model $(DP_{\delta,\gamma})$ `Heuristic Iterative Algorithm (HIA)`, and we show through Example 6, that an optimal solution of $(DP_{\delta,\gamma})$ that does not intersect its data-partition is not necessarily optimal for $(P_{\delta,\gamma})$.

Example 6. *We consider the dataset $\mathcal{I} = (X, Y)$ with 30 data points, 2 features and 2 labels represented in Figure 6. In Figure 6a, the data points which are not representatives in the data-partition $(\mathcal{P}, \tilde{X}, \tilde{Y})$ are grayed out, an the red line plotted is T^* an optimal tree for $(DP_{1,1})$. We can see that $\tilde{E}(T^*) = 4$ since only the representative of the smallest cluster is misclassified. However, T the optimal solution of $(P_{1,1})$ (see Figure 6b) leads to $E(T) = 3$.*



(a) An optimal tree T^* for $(DP_{1,1})$ (b) An optimal tree T for $(P_{1,1})$

Figure 6: Example of an optimal solution of $(DP_{1,1})$ that is not optimal for $(P_{1,1})$ ($\delta = 1$ and $\gamma = 1$)

4.2 An Exact Iterative Algorithm for computing classification trees

We now introduce a new formulation that allows us to compute an optimal tree for the initial dataset starting from any data-partition. In this formulation the choice of the representatives of the clusters is set free. More precisely, since any data point of a cluster can be its representative, we compute them within the optimization process. We further prove that embedding this new, but larger, formulation within our iterative framework (Algorithm 4) allows us to compute an optimal solution for the original dataset. We call this method `Exact Iterative Algorithm (EIA)`.

In our new formulation, we use a binary variable r_i for each data point $i \in \mathcal{I}$, where $r_i = 1$ if and only if i is the representative of its cluster. Then, to ensure that the representatives are correctly classified, the branching Constraints (24) and (25) are considered for all data points in $\mathcal{I}$. We obtain

the following the formulation:

$$
(DP'_{\alpha,\delta})\begin{cases}
\min\ \tilde{f}(\theta,d) = \displaystyle\sum_{\ell\in\mathcal{L}}\sum_{k\in\mathcal{K}}\sum_{g\in 1,\dots,|\mathcal{P}|,\tilde{Y}_k\neq k}|p_g|\theta_{g,k,\ell} + \frac{1}{\gamma+1}\sum_{t\in N} d_t & \\
\text{s.t. } (1)-(5),\ (8),\ (14),\ (16)-(18),\ (21)-(23) & \\
\quad \displaystyle\sum_{j\in\mathcal{J}} a_{j,t}(x_{i,j}+\mu_j-\mu^-)+\mu^- & \\
\qquad \le b_t + (1+\mu^+)(2-z_{g,\ell}-r_i) & g\in 1,\dots,|\mathcal{P}|, i\in p_g, \ell\in\mathcal{L}, t\in A_L(\ell) \quad (24)\\
\quad \displaystyle\sum_{j\in\mathcal{J}} a_{j,t}x_{i,j} \ge b_t - (2-z_{g,\ell}-r_i) & g\in 1,\dots,|\mathcal{P}|, i\in p_g, \ell\in\mathcal{L}, t\in A_R(\ell) \quad (25)\\
\quad \displaystyle\sum_{i\in p_g} r_i = 1 & g\in\{1,\dots,|\mathcal{P}|\} \quad (26)\\
\quad r_i\in\{0,1\} & i\in\mathcal{I} \quad (27)
\end{cases}
$$

Observe that Constraints (24) and (25) only constrain the path of a data point i if $r_i = 1$ (i.e. if it is the representative of its cluster). In Formulation $(DP'_{\delta,\gamma})$ the number of constraints is in the same order of magnitude than in Formulation $(P_{\delta,\gamma})$ but the number of variables can be significantly reduced. Indeed, the number of variables z and θ is proportional to the number of clusters rather than to $|\mathcal{I}|$.

The following proposition proves that we can characterize solutions of $(DP'_{\delta,\gamma})$ that are optimal for $(P_{\delta,\gamma})$.

Proposition 3. *Let $\mathcal{I}=(X,Y)$ be a dataset, $(\mathcal{P},\tilde{X},\tilde{Y})$ a data-partition of $\mathcal{I}$ and T an optimal solution of $(DP'_{\delta,\gamma})$ that does not intersect $(\mathcal{P},\tilde{X},\tilde{Y})$. We have $f(\theta,d)=\tilde{f}(\theta,d)$ and T is optimal for $(P_{\delta,\gamma})$.*

Proof. Note that all variables of $(P_{\delta,\gamma})$ are also in $(DP'_{\delta,\gamma})$, while $(DP'_{\delta,\gamma})$ has additional r_i variables.

1. $v(P_{\delta,\gamma}) \le v(DP'_{\delta,\gamma})$. Obviously T is feasible for $(P_{\delta,\gamma})$. Moreover, since T does not intersect $(\mathcal{P},\tilde{X},\tilde{Y})$, by Proposition 1 we have $E(T)=\tilde{E}(T)$, and thus $f(\theta,d)=\tilde{f}(\theta,d)$.
2. $v(P_{\delta,\gamma}) \ge v(DP'_{\delta,\gamma})$. Let T^* be an optimal solution to $(P_{\delta,\gamma})$. To construct a feasible solution to $(DP'_{\delta,\gamma})$ where $E(T^*)\ge\tilde{E}(T^*)$, we take the tree T^*, and we fix variables r_i as follows. For each cluster p we distinguish two cases:
   - if $\exists\bar{i}\in p$ correctly classified by T, we take $r_{\bar{i}}=1$, and $r_i=0$, $\forall i\in p$, $i\neq\bar{i}$. Thus, there is no misclassification associated to p in $(DP'_{\delta,\gamma})$ Consequently, the number of misclassifications for p in the initial dataset can only be higher.
   - else, the number of misclassifications associated to p is $|p|$ in both the initial dataset and the data-partition. We thus we can choose any $i\in p$, such that $r_i=1$.

   This choice of the representative ensures that $E(T^*)\ge\tilde{E}(T^*)$. From a feasible solution to $(P_{\delta,\gamma})$, we thus built a feasible $(DP'_{\delta,\gamma})$ where $f(\theta,d)\ge\tilde{f}(\theta,d)$.

□

Combining Propositions 2 and 3, we can deduce the following Theorem.

Theorem 1. *Let $\mathcal{I}=(X,Y)$ be a dataset and $(\mathcal{P},\tilde{X},\tilde{Y})$ be a data-partition of $\mathcal{I}$. The tree T computed by `EIA` computes an optimal solution of $(P_{\delta,\gamma})$ for $\mathcal{I}$.*

Proof. We have to prove that *i)* Algorithm 4 converges to a solution that does not satisfy the intersection property, *ii)* such a T is also an optimal solution of $(P_{\delta,\gamma})$ for $\mathcal{I}$. These two claims are true respectively by Propositions 2 and 3. □

4.3 Algorithmic features to speed-up the computation

We now present algorithmic features that enable to accelerate the computation of classification trees. Our first idea is to post-process each solution obtained by (M) in order to reduce the number of iterations of Algorithm 4. Since the variables that define the split functions are continuous, our new formulations (DP) and (DP') have a large number of equivalent optimal solutions, and thus the optimal tree T^* returned by the MILP solver may not be the most relevant for the initial dataset. Indeed, T^* may intersect clusters even if there exists an equivalent optimal tree $\bar{T}$ which does not. As stated by Property 4, a cluster p_g is not intersected by a tree T if all data points belonging to p_g follow the same path as their representative in the data partition. Starting from an initial optimal solution T^*, our aim is thus to build a new tree $\bar{T}$ that is optimal for (DP) (or (DP')), i.e. such that $\tilde{f}(\theta^*, d^*) = \tilde{f}(\bar{\theta}, \bar{d})$, and that maximizes the number of data points that follow their representative in, i.e. such that $f(\theta^*, d^*) \geq f(\bar{\theta}, \bar{d})$.

In order to ensure that our new tree $\bar{T}$ is optimal for (M), we keep in $\bar{T}$ the structure of T^*, i.e. we keep the same branching and leaf nodes, and for each branching nodes, we set the value of the linear coefficients $\bar{a} = a^*$ taken from T^*. Thus, the only differences between T^* and $\bar{T}$ is the value of $\bar{b}$, the right-hand side of the split functions. Starting from the root node, we compute for each splitting node t the new value of $\bar{b}_t$ through the resolution of an optimization problem called (OPT_t) presented below.

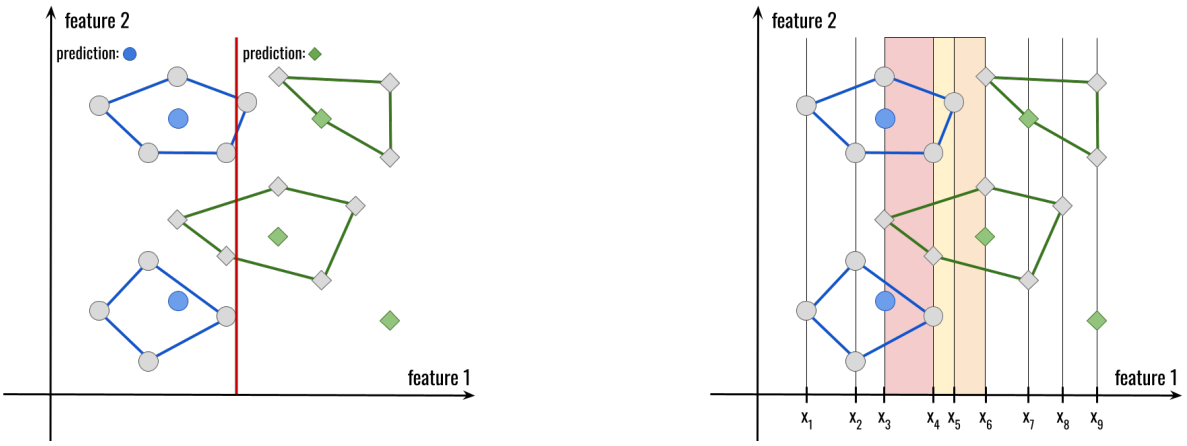
To define this problem, let $L(t) \subseteq \mathcal{P}$ ($R(t) \subseteq \mathcal{P}$ resp.) be the subset of clusters whose representatives are correctly classified and that go to the left (right resp.) at node t of T^*. To ensure that $\tilde{f}(\theta^*, d^*) = \tilde{f}(\bar{\theta}, \bar{d})$, we impose that the path of the representatives of the clusters in $L(t)$ and $R(t)$ remains the same in $\bar{T}$. Then, let $l(t) \subset \mathcal{I}$ ($r(t) \subset \mathcal{I}$, resp.) be the subset of data points that reach node t and for which going left (right resp.) make them either follow their correctly classified representative or not follow their misclassified representative. Maximizing the number of data points of $l(t)$ going to the left and of $r(t)$ going to the right may enable to reduce the number of misclassifications.

In formulation (OPT_t), we denote by $\tilde{x}_g$ the feature vector of the representative of cluster p_g, and by x_i the feature vector of data point i of the original dataset $\mathcal{I}$. Let j^* be the index of the unique non-zero component of a^*, i.e. such that $a^*_{j^*} = 1$. Similarly to Support Vector Machine algorithms [33], we try to find a separation which is as far as possible from the data points. To that end, we consider bound variables b_1 and b_2 on $\bar{b}_t$. The value of $\bar{b}_t$ is deduced by these bounds by setting it to $\max\left\{\frac{b_1+b_2}{2}, b_1+\mu\right\}$, where μ is the constant used to model the strict inequality $x_{i,j^*} < b$. We use additional binary variables z_i ($i \in l(t) \cup r(t)$), and Constraints (28) and (29), to ensure $z_i = 1$ when data point i reaches the left son of t i.e. when $x_{i,j^*} < b$. Moreover, Constraints (30) and (31) ensure that $\tilde{f}(\theta^*, d^*) = \tilde{f}(\bar{\theta}, \bar{d})$. Finally, Constraints (32) and (33) define the ranges of variables b_1 and b_2. As stated before, we aim at building $\bar{T}$ such that $f(\theta^*, d^*) \geq f(\bar{\theta}, \bar{d})$. For this, we maximize the number of data points of $l(t)$ that go left, and of $r(t)$ that go right. Finally, the secondary objective (weighted by 0.5) maximizes the margin $b_2 - b_1$. We thus obtain the following optimization problem (OPT_t):

$$
(OPT_t)\begin{cases}
\max \displaystyle\sum_{i\in l(t)} z_i + \sum_{i\in r(t)} (1-z_i) + \frac{b_2-b_1}{2} & & \\
\text{s.t.} \quad x_{i,j^*} \le b_1 + (1+\mu)(1-z_i) & i \in l(t)\cup r(t),\ j^* : a^*_{j^*}=1 & (28)\\
\quad x_{i,j^*} \ge b_2 - z_i & i \in l(t)\cup r(t),\ j^* : a^*_{j^*}=1 & (29)\\
\quad \tilde{x}_{g,j^*} \le b_1 & g \in L(t),\ j^* : a^*_{j^*}=1 & (30)\\
\quad \tilde{x}_{g,j^*} \ge b_2 & g \in R(t),\ j^* : a^*_{j^*}=1 & (31)\\
\quad 0 \le b_1 \le b_2 - \mu & & (32)\\
\quad b_1 + \mu \le b_2 \le 1 & & (33)\\
\quad z_i \in \{0,1\} & i \in l(t)\cup r(t) & (34)\\
\quad b_1 \in \mathbb{R},\ b_1 \in \mathbb{R} & & (35)
\end{cases}
$$

Note that (OPT_t) can be solve either by a MILP solver or by a tailored enumeration algorithm called `EnumInterval` that we sum up in Algorithm 5. Its basic idea is to enumerate, within the sorted set S, intervals $[b_1, b_2]$ of values of $\bar{b}_t$ for which the number of misclassifications is constant. These intervals are obtained by considering the consecutive values of data points in $l(t) \cup r(t)$ on feature j^*. For each interval, the associated number of misclassifications is computed. The value of $\bar{b}_t$ is set to the middle of an interval that produces the lowest number of errors.

We illustrate algorithm `EnumInterval` in Figure 7. We see in Figure 7a the split function (on feature $j^* = 1$) of the root node t in an optimal solution T^* of (DP') where $\delta = 2$. Note that, all representatives are correctly classified, and the subsets $l(t)$ ($(r(t)$ resp.) contains all the data points labeled with the circle (square resp.). Our goal is to translate the split function (i.e. compute $\bar{b}_t$) such that as many data points as possible follow their representatives. For this, we consider the 8 intervals of values of $\bar{b}_t$ for which the number of misclassifications is constant (see Figure 7b). Since the maximal value (on feature 1) of a representative going to the left is x_3, and the minimal value of a representative going to the right is x_6, we have $x_3 < \bar{b}_t \le x_6$. Figure 7b highlights the intervals $]x_3, x_4]$, $]x_4, x_5]$ and $]x_5, x_6]$ in set S, for which any value of $\bar{b}_t$ leads to a constant number of errors in (OPT_t). The number of errors for each interval is as follows: 4 for $]x_3, x_4]$, 3 for $]x_4, x_5]$, and 2 for $]x_5, x_6]$. Consequently, $\bar{b}_t$ is set to $\frac{x_5+x_6}{2}$.



(a) Solution on the data-partition T^*

(b) Possible intervals

Figure 7: Illustration of Algorithm `EnumInterval`

Algorithm 5: `EnumInterval`$((X,Y),(\mathcal{P},\tilde{X},\tilde{Y}),L(t),R(t),l(t),r(t),j^*,b^*)$

$B_{min} \leftarrow \max_{g\in L(t)}\{\tilde{x}_{g,j^*}\}$
$B_{max} \leftarrow \min_{g\in R(t)}\{\tilde{x}_{g,j^*}\}$
$S \leftarrow$ increasing sorted subset of x_{i,j^*} such that $B_{min} \leq x_{i,j^*} \leq B_{max}$ and $i \in l(t) \cup r(t)$
$S \leftarrow S \cup \{B_{min}, B_{max}\}$
$E \leftarrow$ number of errors induced by T^* for the sub-tree of root t
$(\bar{b}_1, \bar{b}_2) \leftarrow (b^*, b^*)$
for *all consecutive pairs* $(b_1, b_2) \in S$ **do**
  $e \leftarrow |i \in l(t) : x_{i,j^*} \geq b_2| + |i \in r(t) : x_{i,j^*} \leq b_1|$
  if $e < E$ *or* $(e = E$ *and* $b_2 - b_1 > b_2^* - b_1^*)$ **then**
    $E \leftarrow e$
    $(\bar{b}_1, \bar{b}_2) \leftarrow (b_1, b_2)$
  end
end
$\bar{b}_t \leftarrow \frac{b_1+b_2}{2}$
Return $\bar{b}_t$

Proposition 4. *From an optimal solution T^* of (DP) or (DP'), Algorithm* `EnumInterval` *computes a value $\bar{b}_t$ that induces a tree $\bar{T}$ such that $\tilde{f}(\theta^*, d^*) = \tilde{f}(\bar{\theta}, \bar{d})$, and $f(\theta^*, d^*) \geq f(\bar{\theta}, \bar{d})$.*

Proof. The proof is obvious by the initialization of E and of $(\bar{b}_1, \bar{b}_2)$ from T^*, and the condition within the loop that ensures to keep a value as good as that of the data-partition MILP ((DP) or (DP')). □

Finally, our post-processing is sketched-up in Algorithm 6.

Algorithm 6: `PostProcess`$((X,Y),(\mathcal{P},\tilde{X},\tilde{Y}),T^*)$

$\bar{T} \leftarrow T^*$
$\forall t \in \mathcal{N}, L(t) \leftarrow$ representatives correctly classified and going left in T^*
$\forall t \in \mathcal{N}, R(t) \leftarrow$ representatives correctly classified and going right in T^*
for *each branching node* $t \in \mathcal{N}$ **do**
  $l(t) \leftarrow$ data points reaching t whose representative either go to the left and is correctly classified or go to the right and is misclassified
  $r(t) \leftarrow$ data points reaching t whose representative either go to the right and is correctly classified or go to the left and is misclassified
  $\bar{b}_t \leftarrow$ `EnumInterval`$((X,Y),(\mathcal{P},\tilde{X},\tilde{Y}),L(t),R(t),l(t),r(t),j^*,b^*)$
  Replace in $\bar{T}$ the separation of node t by $a_t^* x_i < \bar{b}_t$
end
Return $\bar{T}$

From Proposition 4, we can deduce the following corollary.

Corollary 2. *Given a dataset $\mathcal{I} = (X,Y)$, an associated data-partition $(\mathcal{P},\tilde{X},\tilde{Y})$ and T^* an optimal solution of (DP) ((DP'), resp.), the tree provided by Algorithm* `PostProcess` *is also optimal for (DP) ((DP'), resp.).*

To further improve the resolution, we make the following remarks:

- Algorithm `PostProcess` satisfies Property 4 even if T^* is not an optimal solution of (DP) or (DP').
- Since Algorithm `PostProcess` preserves optimality, integrating it within `EIA` still provides an optimal solution of (P). Theorem 1 then stands when `PostProcess` is used within `EIA`.

Finally, since optimally solving (DP') can be computationally costly, we show that in our framework, we do not need to optimally solve it at every iteration to calculate a tree that is optimal for the original formulation (P). Indeed, from Proposition 3 we can easily deduce that it is sufficient to optimally solve (DP') at the last iteration of `EIA` to compute an optimal solution to (P). We thus use this property within our `ImprovedIterativeRefinement (IIR)` sketched up in Algorithm 7.

Algorithm 7: `ImprovedIterativeRefinement`$((X,Y),(\mathcal{P},\tilde{X},\tilde{Y}),(M))$

$(\mathcal{P},\tilde{X},\tilde{Y})_0 \leftarrow (\mathcal{P},\tilde{X},\tilde{Y})$
$T_0 \leftarrow$ optimal solution of (M) for $(\mathcal{P},\tilde{X},\tilde{Y})_0$
$k \leftarrow 0$
while T_k *intersects* $(\mathcal{P},\tilde{X},\tilde{Y})_k$ *and the time limit is not reached* **do**
  Split clusters of $(\mathcal{P},\tilde{X},\tilde{Y})_k$ to create $(\mathcal{P},\tilde{X},\tilde{Y})_{k+1}$ that is not intersected by T_k
  limitIter $\leftarrow 1/3$ of the remaining time
  $T_{k+1} \leftarrow$ feasible solution of (M) for $(\mathcal{P},\tilde{X},\tilde{Y})_{k+1}$ obtained within *limitIter*
  $T_{k+1} \leftarrow$ `PostProcess`$((X,Y),(\mathcal{P},\tilde{X},\tilde{Y})_k,T_{k+1})$
  if T_{k+1} *does not intersects* $(\mathcal{P},\tilde{X},\tilde{Y})_{k+1}$ *and* T_{k+1} *is not proven optimal* **then**
    $T_{k+1} \leftarrow$ Resume solving (M) for $(\mathcal{P},\tilde{X},\tilde{Y})_{k+1}$ for the remaining time
  end
  $k \leftarrow k+1$
end
Return T_k

In the rest of the paper, we denote by `Heuristic Improved Iterative Algorithm (HIIA)` the Algorithm 7 applied to model $(DP_{\delta,\gamma})$ and by `Exact Improved Iterative Algorithm (EIIA)` the Algorithm 7 applied to model $(DP'_{\delta,\gamma})$.

5 Computational results

In this section, we evaluate our algorithms on several datasets of the literature [13] whose main characteristics are reported in Table 1.

Datasets	$\mathcal{I}$	$\mathcal{J}$	$\mathcal{K}$
Monk3	122	6	2
Monk1	124	6	2
Iris	150	4	3
Monk2	169	6	2
Wine	178	13	3
Seeds	210	6	3
Haberman	306	3	2
Dermatology	358	34	6
Balance scale	625	4	3
Breast Cancer	683	9	2
Blood Transfusion	748	4	2
Car Evaluation	1728	6	4
Statlog satellite	4435	36	7
Spambase	4601	57	2
Magic04	19020	10	2

Table 1: Datasets and their characteristics.

We start by presenting a detailed comparison of our algorithmic features on a subset of datasets: *Iris*, *Monk2*, *Wine*, *Haberman*, and *Magic04*. In particular, we compare the CPU times and the accuracy of the direct solution of (P) with that of our data-partitioning algorithms: `GreedyPartitionning (GP)` (Alg. 1), `ImprovedGreedyPartitionning (IGP)` (Alg. 2), and `ClosestRepresentativeMerge (CRM)` (Alg. 3) with algorithms `HIIA` or `EIIA`. We also evaluate the impact of the use of our `PostProcess` step (Alg. 6) into the whole process. To sum up, for the two steps of our solution methods, we consider three data-partitioning schemes and the two iterative algorithms based on Algorithm 7 (Exact or heuristic). We summarize in Table 2 the methods that we have tested, but in the next sections we only report the results of the most relevant ones.

Data-partitioning	Iterative algorithm	Name
GP	HIIA	GP+HIIA
IGP		IGP+HIIA
CRM		CRM+HIIA
GP	EIIA	GP+EIIA
IGP		IGP+EIIA
CRM		CRM+EIIA

Table 2: Methods considered in our experiments with associated steps.

Finally, in the last section, we compare the best combinations of our solution methods with state-of-the-art methods `TAO` [9], `GUIDE` [25], `CART` [7], `C5.0` [31] via out-sample accuracy.

Experimental environment
Our experiments were carried out on a server with 2 CPU Intel Xeon each of them having 16 cores and 32 threads of 2.3 GHz and $8 * 16$ GB of RAM using a Linux operating system. Each process is limited to use 2 threads. For solving the MILP formulations (P), (DP) and (DP') we used the solver `gurobi` version 9.1.1 [28]. Algorithms `HIIA` and `EIIA` are implemented in `C++`. We ran algorithms `CART` and `C5.0` with `R` libraries. For methods `TAO` and `GUIDE` we directly report the results provided in [40]. For all other methods and all datasets, we consider the 5 partitions from [34]. A partition is described by a training set, a validation set and a test set representing 50%, 25% and 25% of the original datasets, respectively. We consider values of $\delta \in \{2, 3, 4\}$, and we set the time limit to 1800 seconds.

5.1 Comparison of our algorithmic features

In this section, we provide an analysis of our various algorithmic features. We start by evaluating the quality of our data-partitioning algorithms, in particular how much the computed data-partitions satisfy Properties 1–3 introduced in Section 3. Then, we assess the efficiency of our `Post-Process` (Algorithm 6) applied to algorithms `HIIA` and `EIIA`. Finally, we compare them to (P) from their in-sample accuracy point of view.

Description of the instances
We consider the following datasets:

- *Iris* and *Wine* are small datasets that can be solved optimally within 30 minutes;
- *Monk2* and *Haberman* are harder small datasets, i.e. that can not all be solved optimally within 30 minutes;
- *Magic04* is a large dataset for which the direct solution of (P) is impractical.

For each dataset, we run our tests on 5 different training sets. Moreover, we consider 8 different values of parameters $(\delta, \gamma) = \{(2,2), (2,3), (3,3), (3,5), (3,7), (4,4), (4,7), (4,10)\}$, obtaining a total of 40 instances per dataset.

5.1.1 Comparison of data-partition algorithms on Properties 1–3

We compare how the data-partitions computed by algorithms `GreedyPartitionning (GP)` (with $\rho = 0.25$), `ImprovedGreedyPartitionning (IGP)`, and `ClosestRepresentativeMerge (CRM)` satisfy the Properties 1–3. We present in Table 3 the average reduction, exclusion, and consistency metrics for the 40 considered instances. We denote by $'-'$ the cases where a data-partition was not obtained within 400s. Recall that the size reduction metric is defined as $R = \frac{|\mathcal{I}|-|\mathcal{P}|}{|\mathcal{I}|}$.

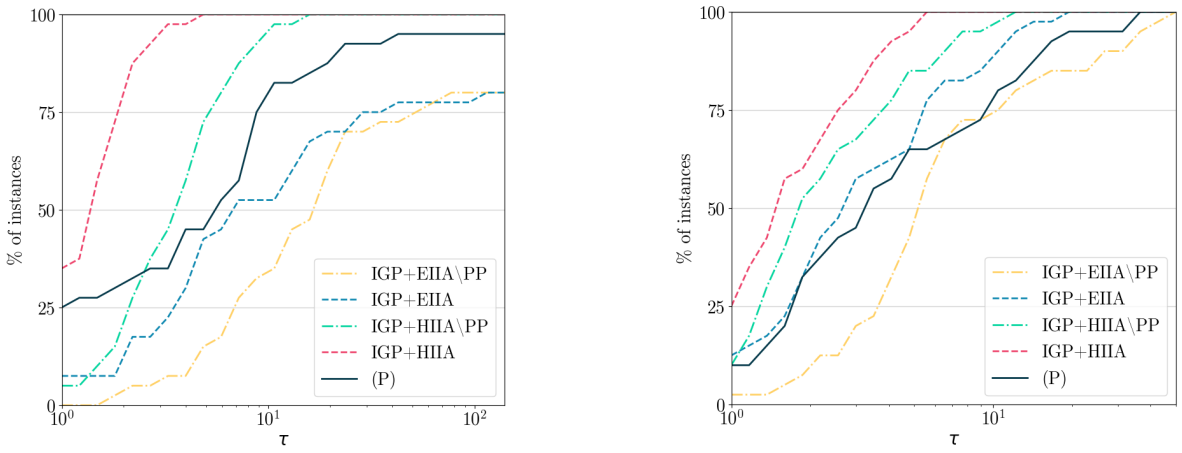
Metric	Data-partitioning algorithms	Iris	Wine	Monk2	Haberman	Magic04
Size reduction	GP	0.75	0.75	0.75	0.75	0.75
	IGP	0.77	0.26	0.73	0.39	0.54
	CRM	0.81	0.33	0.83	0.4	-
Exclusion	GP	1	1	0.87	0.77	0.97
	IGP	1	1	1	0.97	1
	CRM	1	1	1	0.97	-
Consistency	GP	0.69	0.83	0.23	0.43	0.36
	IGP	1	1	1	1	1
	CRM	1	1	0.9	1	-

Table 3: Average reduction, exclusion, and consistency metrics for algorithms `GP`, `IGP`, and `CRM`

We recall that, by construction, all algorithms provide homogeneous data-partition, the homogeneous metric is thus always equal to 1. We observe that the exclusion and consistency metrics are lower for `GP` than for `IGP` or `CRM`. This is especially the case for *Monk2*, *Haberman* and *Magic04*. Note moreover that `CRM` cannot provide data-partitions under a few minutes for datasets of more than a few thousand data points. `GP` and `IGP` have a total CPU time always smaller than a minute which is negligible in comparison to the total CPU time needed for computing the classification tree.

5.1.2 Impact of algorithm `Post-Process` on `EIIA` and `HIIA`

We now compare the CPU times for the datasets *Iris* and *Wine* of Algorithms `EIIA` and `HIIA` both used with and without algorithm `Post-Process (PP)`. We label by `\PP` the methods that do not use `Post-Process (PP)`. For these experiments, we compute the starting data-partition with algorithm `IGP` (i.e. Alg. 2). We present in Figure 8 the performance profiles of the CPU times for methods `IGP+EIIA`, `IGP+EIIA\PP`, `IGP+HIIA`, `IGP+HIIA\PP`, and (P) on the datasets *Iris* (Fig. 8a) and *Wine* (Fig. 8b). In a performance profile of CPU times, each curve corresponds to a formulation, where each point of a curve gives, for a given factor τ, the percentage of instances whose CPU time was at most τ times greater than the minimal CPU time within the considered methods. In particular, for $\tau = 1$, we have the proportion of instances on which the formulation was the best on the criterion.



(a) *Iris* (b) *Wine*

Figure 8: Performance profiles of CPU time

This profile shows a clear order in terms of CPU times between our 4 algorithms for both datasets: the fastest is `IGP+HIIA`, then `IGP+HIIA\PP`, then `IGP+EIIA` and finally `IGP+EIIA\PP`. Note that we also ran experiments with algorithms `GP` or `CRM` instead of `IGP` for computing the data-partitions and we obtained similar results. We also observe that `IGP+HIIA` and `IGP+HIIA\PP` are always faster than solving (P), with an average CPU time reduced by a factor of 2.8 and 1.7 respectively. In contrast, `IGP+EIIA\PP` is on average two time slower than (P). In the rest of the paper, we only report the results with the post-processing step since it always improves the performances.

5.1.3 Comparison of in-sample accuracy of algorithms `EIIA` and `HIIA`

In this section, we compare the performances of relevant combinations of our algorithms with (P). We present in Table 4, the average in-sample accuracy for (P) as well as for `EIIA` and `HIIA` methods combined with `GP` (with $\rho = 0.25$), `IGP` and `CRM` algorithms for computing the starting data-partition. Each line corresponds to the average in-sample accuracy over the 40 considered instances. We highlight in bold the results that are as good as that of (P). For each dataset, we also label the best performance method by a '*'.

Algorithm	Iris	Wine	Monk2	Haberman	Magic04
`GP+EIIA`	**98.8***	**98.9***	74.9	79.6	**74**
`GP+HIIA`	**98.8***	**98.9***	75	80.2	**78.7**
`IGP+EIIA`	98.6	**98.9***	75.3	**80.5**	**79.4***
`IGP+HIIA`	**98.8***	**98.9***	75.3	**80.8***	**79.4***
`CRM+EIIA`	98.6	**98.9***	75.3	**80.4**	-
`CRM+HIIA`	**98.8***	**98.9***	**75.5***	**80.7**	-
(P)	98.8*	98.9*	75.4	80.3	65.3

Table 4: Comparison of average in-sample accuracy with methods `GP+EIIA`, `GP+HIIA`, `IGP+EIIA`, `IGP+HIIA`, `CRM+EIIA`, `CRM+HIIA` and (P).

We observe that for *Iris* and *Wine*, `HIIA` has always the same in-sample accuracy as that of (P). Since almost all these instances are solved optimally by (P) within the time limit of 30 minutes, it means that `HIIA` provides almost always an optimal solution. In contrast, `EIIA` failed to reach the accuracy of (P) for the *Iris* dataset. Regarding the data-partition algorithms, we observe that for

harder datasets (i.e. *Haberman* and *Magic04*), IGP and CRM combined with EIIA or HIIA algorithms outperform GP+EIIA, GP+HIIA, and the solution of (P). In particular, results obtained for *Magic04* show that our methods scale-up better than (P). Note moreover that HIIA and EIIA are faster when combined with IGP and CRM than with GP. These results can be related to the low metric values of Table 3 obtained by GP where exclusion and consistency metrics are decreasing with respect to the reduction metric ρ. However, choosing a relevant value of ρ that leads to a good balance between the 3 criteria is highly dependent of the considered dataset.

5.2 Comparison of out-sample accuracy with state-of-the-art methods

We now compare our algorithms with state-of-the-art methods from the out-sample accuracy point of view. We first give a detailed comparison of (P) and of the possible combinations of our algorithms. Then, we compare our 2 best performing combinations with the state-of-the-art heuristics TAO [9], GUIDE [25, 26], CART [7], and C5.0 [31].

For methods HIIA, EIIA, (P), CART, and C5.0, our experiments are carried out on 5 different partitions in training, validation and test sets. Since we do not have the sources of algorithms TAO and GUIDE, we reported the results taken from [40] that were not run on the same partitions. For (P) and our algorithms, trees are obtained via an iterative learning algorithm that selects the best values for (δ, γ), where each iteration has a time limit of 30 minutes. Data-partitions are computed once per training set, before the learning process. In order to choose a relevant value of ρ for algorithm GP, we solve $(DP_{\delta=4,\gamma=15})$ (or $(DP'_{\delta=4,\gamma=15})$) with different values of ρ with a time limit of 5 minutes, and select the value of ρ that achieves the best in-sample accuracy.

5.2.1 Comparison with the direct resolution of (P)

We present in Tables 5 and 6 the out-sample accuracy results obtained with 6 combinations of our algorithms that we compare with (P) for depths $\delta \in \{3, 4\}$, respectively. Each line corresponds to the average results of a dataset, where the method with the highest out-sample accuracy is highlighted in bold. In order to compare our methods with (P), we indicate the average Relative Difference (RD), with $RD = \frac{Acc(M)-Acc(P)}{Acc(P)}$, where $Acc(P)$ is the accuracy of (P), and $Acc(M)$ the accuracy of the method M of the column. Thus, when RD is positive, it means that we are more accurate than (P).

	HIIA			EIIA			
Dataset	GP	IGP	CRM	GP	IGP	CRM	(P)
Monk3	92.26	92.90	92.90	92.26	**93.55**	92.26	**93.55**
Monk1	**90.32**	**90.32**	**90.32**	**90.32**	**90.32**	**90.32**	88.39
Iris	97.37	**97.89**	**97.89**	**97.89**	**97.89**	97.37	97.37
Monk2	53.95	56.74	54.42	55.35	53.95	57.21	**57.67**
Wine	90.65	90.65	90.20	**90.66**	90.20	90.2	90.21
Seeds	92.83	91.70	**93.21**	91.70	90.94	89.81	89.81
Haberman	72.47	74.81	**75.84**	73.77	74.03	75.32	74.03
Dermatology	87.33	**87.56**	87.11	79.78	84.44	86.67	87.11
Balance scale	70.96	71.72	**71.97**	70.70	71.46	70.57	70.7
Breast cancer	94.62	94.62	94.50	**94.97**	94.85	**94.97**	94.62
Blood transfusion	80.11	79.57	79.25	**80.53**	79.68	80.0	79.28
Car evaluation	78.52	**79.26**	77.69	78.15	78.56	77.59	**79.26**
Statlog satellite	**77.91**	**77.91**	-	77.66	77.89	-	76.86
Spambase	88.72	88.83	-	**88.93**	88.71	-	86.26
Average RD with (P)	*+0.07*	*+0.77*	*+0.19*	*-0.3*	*-0.03*	*+0.02*	

Table 5: Average out-sample accuracy for trees of depth $\delta = 3$ over 5 test sets

For depth $\delta = 3$, we observe in Table 5, that HIIA combined with any data-partitioning algorithms always has a positive RD. However, the highest RD of $+0.77$ is rather small , even if method IGP+HIIA strictly outperforms (P) on 9 datasets over 17. The two other combinations leads to similar or better performances than (P), with an RD close to zero. Regarding combinations that use EIIA, we can see that the RDs are very close to zero for any of the considered data-partitioning algorithms. Yet, unexpectedly, method GP+EIIA that has the lowest relative difference of -0.3 is the combination that most often get the highest out-sample accuracy (on more than a third of the datasets).

	HIIA			EIIA			
Dataset	GP	IGP	CRM	GP	IGP	CRM	(P)
Monk3	**93.55**	**93.55**	**93.55**	**93.55**	92.90	92.90	**93.55**
Monk1	**96.13**	93.55	**96.13**	92.90	**96.13**	95.48	92.26
Iris	97.37	**97.89**	**97.89**	**97.89**	**97.89**	97.37	97.37
Monk2	**62.33**	60.93	60.47	60.93	60.93	61.4	55.81
Wine	90.65	90.65	90.20	**90.66**	90.20	90.2	90.21
Seeds	92.83	89.81	**93.21**	91.70	90.19	89.43	92.83
Haberman	71.95	**75.58**	74.55	73.77	74.29	74.81	73.51
Dermatology	91.33	**92.0**	90.0	89.11	90.22	90.44	88.89
Balance scale	**79.75**	78.22	75.92	78.85	77.71	78.47	75.41
Breast cancer	94.62	94.50	94.50	**94.85**	94.62	**94.85**	94.62
Blood transfusion	79.79	79.47	80.64	80.32	80.75	**81.07**	78.48
Car evaluation	**86.02**	85.60	84.86	85.14	84.4	84.4	84.4
Statlog satellite	79.84	**80.50**	-	80.27	79.55	-	77.85
Spambase	89.61	90.04	-	**90.48**	89.77	-	86.26
Average RD with (P)	*+2.34*	*+2.05*	*+1.67*	*+1.88*	*+1.79*	*+1.69*	

Table 6: Average out-sample accuracy for trees of depth $\delta = 4$ over 5 test sets

For $\delta = 4$, we observe in Table 6 that the average improvement is greater than for $\delta = 3$, and always positive for all methods. Indeed, the size of the formulations become too large and (P) can no longer optimally solve the instances within the time limit. Regarding our new approaches, they all obtain better or equal results than (P). Methods GP+HIIA, IGP+HIIA, and GP+EIIA appear as the best performers on these instances, with the best accuracy on 5 datasets each. These results clearly show that data-partitioning-based methods are capable of scaling up on these datasets.

5.2.2 Comparison with state-of-the-art heuristics

We now compare our algorithms with state-of-the-art heuristics: TAO, GUIDE, CART and, C5.0. More precisely, we run CART and C5.0 with R libraries on the same partitions used for our algorithms. For CART, we compute the maximum tree and prune it according to the validation set. For C5.0, we took the computed tree as is. For GUIDE and TAO, the results presented are taken from the survey [40] and come from different partitions.

The out-sample accuracy results are presented in Figure 7. Each row corresponds to a dataset for which the best method is highlighted in bold. We recall that for methods GP+HIIA and IGP+HIIA the maximum depth is fixed at $\delta = 4$, whereas for the compared state-of-the-art approaches this parameter is left free. To compare the results obtained on the criterion of accuracy, we compute for each dataset the *Relative Gap (RG)* between the performance of the considered algorithm and that of the best one, i.e. $RG = \left|\frac{B-A}{B}\right| * 100$, where B is the highest accuracy (in bold), and A the accuracy of the considered method. We report in lines *Average RG*, the average relative gap in % over the first 8 datasets, then over the last 7, and we finally give in line *Total average RG* the average over all 15 datasets. Considering only the 8 first datasets, we observe that method TAO, that was run on different partitions, has the smallest average out-sample accuracy. When we compare the performance of the 5

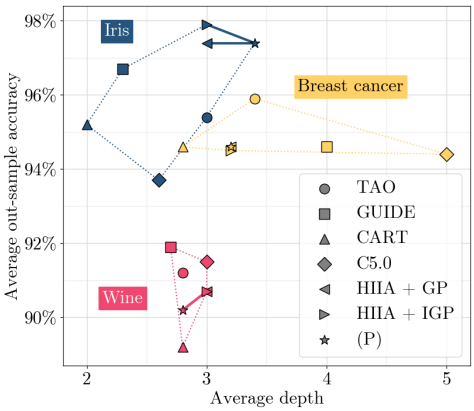
Dataset	Different partitions		Same partitions				
	TAO	GUIDE	CART	C5.0	GP+HIIA	IGP+HIIA	(P)
Iris	95.4	96.7	95.2	93.7	97.4	**97.9**	97.4
Wine	91.2	**91.9**	89.2	91.5	90.7	90.7	90.2
Dermatology	**96.1**	95.5	94.2	94.4	91.3	92	88.9
Balance scale	**82.2**	77.4	77.6	75.9	79.8	78.2	75.4
Breast cancer	**95.9**	94.6	94.6	94.4	94.6	94.5	94.6
Blood transfusion	77.9	78.8	76.1	75.7	**79.8**	79.5	78.5
Car evaluation	**96.7**	70.2	85.8	86.6	86	85.6	84.4
Spambase	92.7	**92.8**	89.2	91.5	89.6	90	86.3
Average RG	*0.7*	*4.7*	*4.3*	*4.1*	*3.2*	*3.3*	*5.1*
Monk3	-	-	90.2	90.2	**93.6**	**93.6**	93.5
Monk1	-	-	74.7	83.1	**96.1**	93.6	92.3
Monk2	-	-	55.7	60.3	**62.3**	60.9	55.8
Seeds	-	-	86.8	89.4	**92.8**	89.8	**92.8**
Haberman	-	-	66.3	68.4	72	**75.6**	73.5
Statlog satellite	-	-	80.8	**84.2**	79.8	80.5	77.9
Magic04	-	-	82.4	**84.5**	82	81.9	-
Average RG	-	-	*8.8*	*4.8*	*1.8*	*2.2*	*4.1*
Total average RG	-	-	*5.2*	*3.2*	*1.4*	*1.6*	*3.4*

Table 7: Average out-sample accuracy for TAO, GUIDE, CART, C5.0, GP+HIIA, IGP+HIIA and (P)

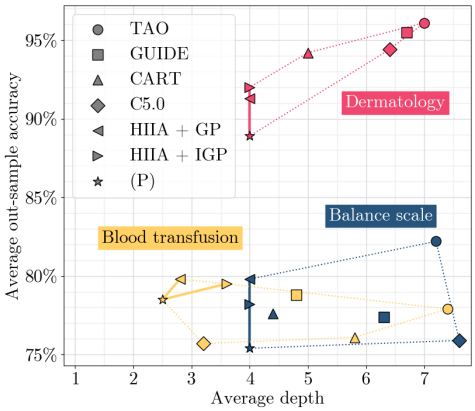
methods we tested on the same partitions, we can see that our 2 new algorithms GP+HIIA and IGP+HIIA have the best average relative gaps. This is also the case for the last 7 datasets, where GP+HIIA and IGP+HIIA outperform the compared methods. More precisely, the relative gap is reduced by a factor of 4.5, 2.6, 2.3, and 1.2 for method GP+HIIA, in comparison to methods CART, C5.0, (P), and IGP+HIIA, respectively.

As already mentioned, in addition to accuracy, an important criterion for assessing the quality of a decision tree is also its depth, since this characterizes its interpretability. To illustrate the balance between these two criteria on the trees built by the methods considered, we propose the diagrams in Figure 9 where we plot the average out-sample accuracy of trees as a function of their sizes. In these diagrams, each point corresponds to the average depth and out-sample accuracy of the trees provided by one method for one dataset. Thus, the further to the left a tree is, the more interpretable it is, the higher up it is, the better its accuracy is. To make it easier to identify the points associated to the same dataset, we draw their convex hull in the same color. Consequently, each point within a convex hull corresponds to a compromise between accuracy and complexity. To further highlight the improvement achieved by our methods HIIA+GP and HIIA+IGP compared with (P), we draw bold lines between their respective points.

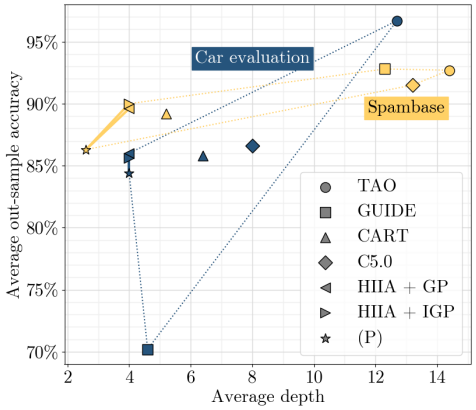
We plot in Figure 9a the results for the easiest datasets for which all methods provide small trees and have very close out-sample accuracies. Our methods provide the best performances on *Iris* while reducing the complexity of the trees compared with (P). On *Wine* all methods exhibit very similar results, but our new methods enhance the accuracy compared with (P). Finally, on *Breast cancer*, TAO has the best performances while only CART provide smaller trees than GP+HIIA and IGP+HIIA. In Figure 9b and 9c, we present results for intermediate and hard instances, respectively. We observe that the state-of-the-art heuristics almost always provide more complex trees than GP+HIIA and IGP+HIIA with depths sometimes more than three times larger. Despite this, our methods reach competitive performances. For *Blood transfusion*, we reach the best accuracy and for *Balance scale*, only TAO outperforms us but at the cost of a significant increase of the tree size (by a factor of 1.75). As expected, for *Dermatology*, *Car evaluation*, and *Spambase*, the performances of TAO highlights that



(a) Simple instances



(b) Intermediate instances



(c) Hard instances

Figure 9: Average out-sample accuracy as a function of the average depth

considering a depth larger than 4 is an advantage for achieving better performances. Note however that the larger size of the trees provided by the other state-of-the-art heuristics does not always lead to better performances. Finally we observe that our new methods always improve either the accuracy or the interpretability of the trees compared to the direct resolution of (P).

6 Conclusion

We consider the problem of computing classification trees that are both interpretable and efficient. Our approach is based on a MILP formulation embedded within an iterative algorithm that allows us to handle medium-sized datasets. More precisely, we introduce the concept of *data-partitions* to scale up the solution of the optimal classification tree problem. By grouping data points together, data-partitions enables to reduce the size of datasets. To frame the construction of data-partitions we introduce 3 properties and associated metrics that measure the level to which properties are satisfied. We define three algorithms that generate partitions likely to exhibit strong performance according to these metrics. We then leverage data-partitions to build optimal classification trees. For this, we design the iterative Algorithm 4, and 2 new formulations that take as an input a data-partition. We prove that the combination of one of our formulation with Algorithm 4 enables to reach global optimality. Computational results show that using data-partitions on medium-sized datasets outperforms the direct MILP solution, and that our methods provide the best compromise between in-sample accuracy and interpretability when compared with state-of-the-art heuristics. In future work, we will focus on extending the use of data-partitions to classification trees with oblique splits.

References

[1] Sina Aghaei, Mohammad Javad Azizi, and Phebe Vayanos. Learning Optimal and Fair Decision Trees for Non-Discriminative Decision-Making. *AAAI*, 33(01):1418–1426, July 2019.

[2] Zacharie Ales, Valentine Huré, and Amélie Lambert. New optimization models for optimal classification trees. *Computers & Operations Research*, 164:106515, April 2024.

[3] Dimitris Bertsimas and Jack Dunn. Optimal classification trees. *Machine Learning*, 106(7):1039–1082, July 2017.

[4] Dimitris Bertsimas and Angela King. Logistic regression: From art to science. *Statistical Science*, pages 367–384, 2017.

[5] Rafael Blanquero, Emilio Carrizosa, Cristina Molero-Río, and Dolores Romero Morales. Optimal randomized classification trees. *Computers & Operations Research*, 132:105281, August 2021.

[6] L. Breiman. Random forests. *Machine Learning*, 45:5–32, 2001.

[7] Leo Breiman, Jerome Friedman, Charles J. Stone, and R. A. Olshen. *Classification and Regression Trees*. Taylor & Francis, January 1984.

[8] C. E. Brodley and P. E. Utgoff. Multivariate decision trees. *Machine Learning*, 19:45–77, 1995.

[9] Miguel A. Carreira-Perpinan and Pooya Tavallali. Alternating optimization of decision trees, with application to learning sparse oblique trees. In *Advances in Neural Information Processing Systems*, volume 31. Curran Associates, Inc., 2018.

[10] T. Chen and C. Guestrin. Xgboost. *Proceedings of the 22nd ACM SIGKDD International Conference on Knowledge Discovery and Data Mining. ACM.*, 2016.

[11] Emir Demirović, Anna Lukina, Emmanuel Hebrard, Jeffrey Chan, James Bailey, Christopher Leckie, Kotagiri Ramamohanarao, and Peter J. Stuckey. MurTree: Optimal Decision Trees via Dynamic Programming and Search. *Journal of Machine Learning Research*, 23(26):1–47, 2022.

[12] Federico D'Onofrio, Giorgio Grani, Marta Monaci, and Laura Palagi. Margin Optimal Classification Trees, January 2023. arXiv:2210.10567 [cs, math].

[13] Dheeru Dua and Casey Graff. UCI machine learning repository, 2017.

[14] Jack William Dunn. *Optimal trees for prediction and prescription*. Thesis, Massachusetts Institute of Technology, 2018. Accepted: 2018-11-28T15:25:46Z.

[15] Murat Firat, Guillaume Crognier, Adriana F. Gabor, C. A. J. Hurkens, and Yingqian Zhang. Column generation based heuristic for learning classification trees. *Computers & Operations Research*, 116:104866, April 2020.

[16] K. Florek, J. Łukaszewicz, J. Perkal, Hugo Steinhaus, and S. Zubrzycki. Sur la liaison et la division des points d'un ensemble fini. *Colloq. Math.*, 2(3-4):282–285, 1951.

[17] J. H. Friedman. Greedy function approximation: A gradient boosting machine. *The Annals of Statistics*, 29(5):1189–1232, 2001.

[18] Sudipto Guha, Rajeev Rastogi, and Kyuseok Shim. Cure: An efficient clustering algorithm for large databases. *ACM Sigmod record*, 27(2):73–84, 1998.

[19] Sudipto Guha, Rajeev Rastogi, and Kyuseok Shim. Rock: A robust clustering algorithm for categorical attributes. *Information systems*, 25(5):345–366, 2000.

[20] Oktay Günlük, Jayant Kalagnanam, Minhan Li, Matt Menickelly, and Katya Scheinberg. Optimal decision trees for categorical data via integer programming. *Journal of Global Optimization*, 81(1):233–260, September 2021.

[21] Xiyang Hu, Cynthia Rudin, and Margo Seltzer. Optimal sparse decision trees. In H. Wallach, H. Larochelle, A. Beygelzimer, F. d'Alché-Buc, E. Fox, and R. Garnett, editors, *Advances in Neural Information Processing Systems*, volume 32. Curran Associates, Inc., 2019.

[22] Laurent Hyafil and Ronald L. Rivest. Constructing optimal binary decision trees is NP-complete. *Information Processing Letters*, 5(1):15–17, May 1976.

[23] George Karypis, Eui-Hong Han, and Vipin Kumar. Chameleon: Hierarchical clustering using dynamic modeling. *computer*, 32(8):68–75, 1999.

[24] Jimmy Lin, Chudi Zhong, Diane Hu, Cynthia Rudin, and Margo Seltzer. Generalized and Scalable Optimal Sparse Decision Trees. In *Proceedings of the 37th International Conference on Machine Learning*, pages 6150–6160. PMLR, November 2020. ISSN: 2640-3498.

[25] Wei-Yin Loh. Regression tress with unbiased variable selection and interaction detection. *Statistica Sinica*, 12(2):361–386, 2002.

[26] Wei-Yin Loh. Improving the precision of classification trees. *The Annals of Applied Statistics*, 3(4):1710–1737, 2009.

[27] S. K. Murthy, S. Kasif, and S. L. Salzberg. A system for induction of oblique decision trees. *Journal of Artificial Intelligence Research*, 2:1–32, 1994.

[28] GUROBI optimization. *GUROBI 9.1*, 2021.

[29] C. Orsenigo and C. Vercellis. Multivariate classification trees based on minimum features discrete support vector machines. *IMA Journal of Management Mathematics*, 14(3):221–234, 2003.

[30] J. R. Quinlan. Induction of decision trees. *Machine Learning*, 1:81–106, 1986.

[31] J. R. Quinlan. C4.5: programs for machine learning. *Morgan Kaufmann Publishers Inc., San Francisco, CA, USA.*, 1993.

[32] R. Sibson. SLINK: An optimally efficient algorithm for the single-link cluster method. *Comput. J.*, 16(1):30–34, January 1973.

[33] V. Vapnik. Pattern recognition using generalized portrait method. *Automation and Remote Control*, 1963.

[34] Sicco Verwer and Yingqian Zhang. Learning Optimal Classification Trees Using a Binary Linear Program Formulation. *Proceedings of the AAAI Conference on Artificial Intelligence*, 33(01):1625–1632, July 2019. Number: 01.

[35] Joe H Ward Jr and Marion E Hook. Application of an hierarchical grouping procedure to a problem of grouping profiles. *Educational and Psychological Measurement*, 23(1):69–81, 1963.

[36] D. C. Wickramarachchi, B. L. Robertson, M. Reale, C. J. Price, and J. Brown. Hhcart: An oblique decision tree. *Computational Statistics & Data Analysis*, 96:12–23, 2016.

[37] Dongkuan Xu and Yingjie Tian. A Comprehensive Survey of Clustering Algorithms. *Ann. Data. Sci.*, 2(2):165–193, June 2015.

[38] Rui Xu and D. Wunsch. Survey of clustering algorithms. *IEEE Transactions on Neural Networks*, 16(3):645–678, 2005.

[39] Tian Zhang, Raghu Ramakrishnan, and Miron Livny. Birch: A new data clustering algorithm and its applications. *Data mining and knowledge discovery*, 1:141–182, 1997.

[40] Arman Zharmagambetov, Suryabhan Singh Hada, Miguel Á. Carreira-Perpiñán, and Magzhan Gabidolla. An Experimental Comparison of Old and New Decision Tree Algorithms. *arXiv*, November 2019.

[41] Haoran Zhu, Pavankumar Murali, Dzung Phan, Lam Nguyen, and Jayant Kalagnanam. A scalable mip-based method for learning optimal multivariate decision trees. *Advances in neural information processing systems*, 33:1771–1781, 2020.