Applied Intelligence (2021) 51:7552–7572
https://doi.org/10.1007/s10489-021-02281-x

A branch & bound algorithm to determine optimal bivariate splits for oblique decision tree induction

Ferdinand Bollwein[1] · Stephan Westphal[1]

Accepted: 17 February 2021 / Published online: 12 March 2021

Abstract
Univariate decision tree induction methods for multiclass classification problems such as CART, C4.5 and ID3 continue to be very popular in the context of machine learning due to their major benefit of being easy to interpret. However, as these trees only consider a single attribute per node, they often get quite large which lowers their explanatory value. Oblique decision tree building algorithms, which divide the feature space by multidimensional hyperplanes, often produce much smaller trees but the individual splits are hard to interpret. Moreover, the effort of finding optimal oblique splits is very high such that heuristics have to be applied to determine local optimal solutions. In this work, we introduce an effective branch and bound procedure to determine global optimal bivariate oblique splits for concave impurity measures. Decision trees based on these bivariate oblique splits remain fairly interpretable due to the restriction to two attributes per split. The resulting trees are significantly smaller and more accurate than their univariate counterparts due to their ability of adapting better to the underlying data and capturing interactions of attribute pairs. Moreover, our evaluation shows that our algorithm even outperforms algorithms based on heuristically obtained multivariate oblique splits despite the fact that we are focusing on two attributes only.

Keywords Branch and bound · Decision trees · Multiclass classification · Bivariate oblique splits

1 Introduction

In recent years, an increasing necessity for interpretable machine learning models has become apparent. Univariate decision trees are a popular tool in this context for solving classification tasks. Their major advantage is that they are easy to understand and interpret, giving insight into the underlying data's structure. Drawbacks of the common univariate decision tree induction methods are that the resulting trees often become very large, even after pruning, and the prediction accuracy is often lower than that of other classification methods.

To tackle these drawbacks, oblique decision tree induction algorithms which employ hyperplanes splits have been developed to produce smaller and more accurate trees. Given m attributes $X_1, \dots, X_m$, these splits correspond to rules of the form

$$a_1X_1 + \dots + a_mX_m \leq b$$

for $a \in \mathbb{R}^m$ and $b \in \mathbb{R}$. Interpreting these splits, however, is far more difficult and finding global optimal hyperplanes has been proven to be NP-complete for some splitting criteria [1]. Thus, heuristics are necessary for finding local optimal solutions.

In this work, we propose a branch and bound algorithm to determine global optimal bivariate oblique splits for concave impurity measures. These splits correspond to oblique splits for which only two coefficients a_{j_1} and a_{j_2} for $j_1, j_2 \in \{1, \dots, m\}$ are non-zero. Thus, they have the form

$$a_{j_1}X_{j_1} + a_{j_2}X_{j_2} \leq b$$

or equivalently,

$$a'X_{j_1} + b' \leq X_{j_2}$$

for suitable parameters a' and b'. Compared to univariate decision tree induction algorithms these splits result in smaller more accurate trees due to the inclusion of an additional attribute. On the other hand, by allowing only two

✉ Ferdinand Bollwein
ferdinand.bollwein@tu-clausthal.de

Stephan Westphal
stephan.westphal@tu-clausthal.de

1 Institute of Mathematics, Clausthal University of Technology, Erzstraße 1, 38678 Clausthal-Zellerfeld, Germany

attributes per split, the trees remain fairly interpretable and the risk of including irrelevant features in the splits during tree induction is reduced compared to general oblique splits.

To show the strength of bivariate splits, we want to look at simple data consisting of two attributes X_1 and X_2 which is labeled as positive if the ratio $\frac{X_1}{X_2}$ is greater than or equal to a certain value $a \in \mathbb{R}$ and negative, otherwise. Clearly, the rule $\frac{X_1}{X_2} \geq a$ is equivalent to $X_1 \geq aX_2$ and can thus be expressed as a bivariate split. Univariate decision trees, on the other hand, require a stepwise approximation of this rule. This observation is illustrated in Fig. 1. Such dependencies are likely to appear in real-life applications. As an example from the medical domain there is the so-called waist-to-height ratio which is a measure for abdominal obesity and, according to a study from 2010 (see [2]), a superior measure to the well-known body mass index for the risk of heart attacks. This measure is defined as $WHtR = \frac{U}{H}$ where U denotes a person's waist circumference and H its height. A $WHtR$ greater than 0.5 indicates an increased risk of obesity-related cardiovascular diseases. As we have seen, this rule can only be approximated by univariate splits while a single bivariate oblique split suffices to express this rule.

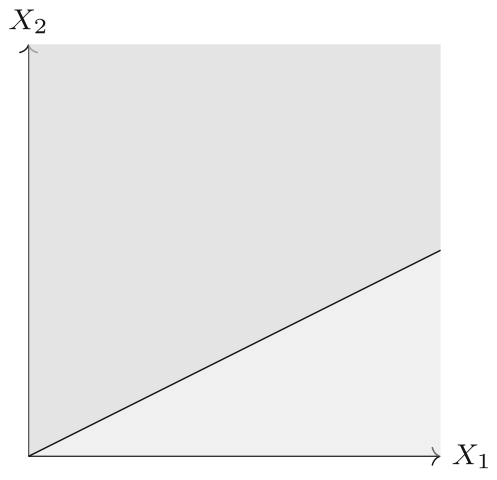
This simple example shows various major benefits of bivariate splits: First, it is possible to express rules based on ratios of two attributes. Furthermore, if the rules are learned algorithmically from data, a human understandable interpretation is possible. The possibility of creating a two-dimensional visual representation of the splits also helps to get a deeper understanding of the data at hand.

Our algorithm is compatible with the very general class of concave impurity measures which includes the most commonly used splitting criteria such as entropy, gini impurity and classification error. This is a major advantage as it is not restricted to one specific criterion and we obviate the need to introduce a new criterion specifically designed for our purposes.

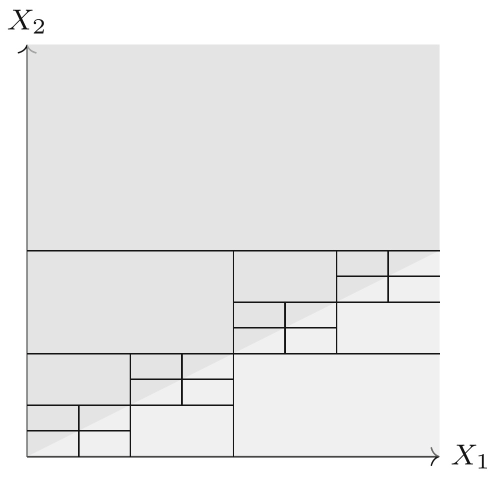
In our evaluation we show that the trees based on global optimal bivariate oblique splits outperform univariate decision trees as well as general oblique decision trees in terms of prediction accuracy and tree size. The biggest disadvantage of our tree building algorithm is the increased runtime compared to univariate decision tree induction algorithms due to the higher complexity of finding bivariate splits. Nevertheless, due to the efficient implementation of the branch and bound procedure, acceptable results are achieved such that our method is applicable in real-life scenarios. Overall, our proposed bivariate decision tree induction algorithm is a reasonable alternative when univariate decision trees fail to give accurate results, yet interpretation of the data at hand is desired.

2 Related work and contribution

Artificial intelligence is more and more affecting our lives in a variety of fields such as health care, finance, criminal justice and defense and researchers have become increasingly aware of the importance of interpretability of machine learning models. Extensive overviews discussing concepts, challenges, trends and existing approaches regarding this topic are, for example, presented in [3, 4]. While there is a significant amount of research that tries to derive interpretable surrogate models [5], including decision trees [6], from black-box models such as neural networks, support vector machines or random forests, some authors plead for relying on inherently interpretable and transparent prediction models instead [7].



(a) Bivariate split



(b) Approximation of the bivariate split by univariate splits

Fig. 1 Comparison of a bivariate split and an approximation by univariate splits for the rule $\frac{X_1}{X_2} \leq 0.5$

Univariate decision tree algorithms such as CART [8] and C4.5 [9] which recursively divide the feature space in a top-down manner by introducing splits minimizing some splitting criterion induce such inherently interpretable models. The general aim of finding the smallest binary decision tree has been proven to be NP-complete [10], yet the greedy strategy made it possible to induce high-quality decision trees fast in practice. Nevertheless, the resulting trees are often not as accurate as other classification models. Moreover, the data can often not be split in a compact way by splits involving only a single attribute which makes interpretation and deriving understandable rules more complicated.

Oblique decision trees are often smaller and more accurate in practice, yet finding optimal oblique splits is NP-complete for some splitting criteria [1] including classification error. Therefore, efficient heuristics are necessary to induce the decision trees in a reasonable amount of time with the greedy recursive partitioning scheme. The first heuristic was introduced by [8]. They use a deterministic hill climbing approach to find local optimal oblique splits. Heath et al. [11] develop a simulated annealing heuristic which perturbs one hyperplane parameter at a time to escape local optima. Murthy et al. [12] combine these two approaches by introducing randomization techniques in the hill climbing approach. Other approaches for finding near optimal oblique splits include meta-heuristics such as simulated annealing, genetic algorithms and evolutionary strategies [13] or heuristics based on Householder transformation [14], linear discriminant analysis [15] and logistic regression [16]. Although these oblique decision tree induction approaches generally produce better classifiers, their main problem is that the oblique splits, involving all of the variables, are hard to understand and interpret. Thus, one loses one of the most advantageous properties of tree-based classification models.

Recently, also integer linear programming has been employed to determine global optimal univariate and oblique decision trees of a prior specified maximum size [17, 18]. Blanquero et al. [19] develop a continuous optimization formulation instead to determine optimal randomized oblique decision trees. Additionally, penalty terms are introduced in the objective function to limit the number of overall involved attributes and the number of attributes per split to improve interpretability. The major drawback of these approaches is that solving the proposed optimization problems is much more time consuming than using the top-down strategy and optimal solutions can only be obtained in a reasonable amount of time for shallow trees and for small to medium-sized datasets.

A common strategy to obtain interpretable machine learning models is to restrict the complexity of the model to be learned beforehand. Lubinsky et al. [20] follows this approach and investigates different kinds of bivariate splits as an interpretable alternative to univariate splits for decision tree induction. He focuses on two-class classification tasks and shows, by using a brute-force algorithm for determining the splits, that decision trees based on bivariate splits, including bivariate oblique splits, are in fact able to produce much smaller and more accurate decision trees compared to their univariate counterparts while remaining interpretable. He states that the brute-force algorithm is very time consuming and often intractable for large datasets. He further introduces additive splitting criteria which enable the development of efficient divide-and-conquer algorithms. Although these additive splitting criteria yield good results in terms of accuracy, they are not as effective as traditional impurity measures such as gini impurity in terms of tree size.

Bioch et al. [21] also argue that oblique decision trees based on many variables are hard to interpret and therefore propose using bivariate oblique splits. They suggest using the hill climbing approach of [8] for finding good splits as well as the simpler approach of considering only 45° lines in addition to standard univariate splits. The evaluation of this approach shows that bivariate oblique decision trees are superior to univariate decision trees and competitive with multivariate decision trees in terms of tree size and accuracy.

In this work, we follow Lubinsky and Bioch et al.'s argumentation that bivariate oblique decision trees can be a true interpretable alternative to univariate as well as multivariate decision trees. However, we aim at finding the best possible split in each node to ensure that we capture the underlying structure of the data as well as possible to produce highly accurate and compact decision trees. This task is computationally intractable for general oblique splits. In contrast to Lubinsky, we also allow for an arbitrary amount of classes and we rely on traditional concave impurity measures including classification error, entropy and gini impurity. We develop an effective branch and bound procedure that exploits the concavity of these impurity measures to compute good lower bounds which is the key for our branch and bound algorithm. Furthermore, we evaluate our method to show that it is capable of producing interpretable, compact and highly accurate decision trees.

3 Preliminaries

3.1 Bivariate oblique decision tree induction

In this work, we study bivariate oblique decision trees for multiclass classification problems. Given m real-valued attributes (or features) $X_1, \ldots, X_m$ and a set of class labels $L = \{1, \ldots, k\}$, a bivariate oblique decision tree is a binary rooted tree that classifies each point $x \in \mathbb{R}^m$ to a certain label $y \in L$. For that, each internal node is associated with a rule of the form $aX_{j_1} + b \leq X_{j_2}$ for some indices $j_1, j_2 \in \{1, \dots, m\}$ and values $a, b \in \mathbb{R}$ and each leaf node is associated with a certain class label. Throughout the remainder of this work, we abbreviate the set of all possible rules as $\Theta := \{aX_{j_1} + b \leq X_{j_2} : j_1, j_2 = 1, \dots, m;\ a, b \in \mathbb{R}\}$. Depending on the context, we will also sometimes refer to the rules as split or line, because the expression basically describes a line in the two dimensional space spanned by the attributes X_{j_1} and X_{j_2} which splits the feature space into two subspaces. In order to classify a point $x \in \mathbb{R}^m$, it pursues a path from the root to a leaf node according to the specified rules. If the rule is satisfied, x is sent to the left sibling and otherwise, x is sent to the right. Finally, the point is assigned the class label of the leaf node at the end of this path. An example of a simple bivariate oblique decision tree of depth two is illustrated in Fig. 2.

To automatically learn such decision trees, we make use of a training set $\mathcal{X} := \{(x^i, y^i) : i = 1, \dots, n;\ x^i \in \mathbb{R}^m;\ y^i \in L\}$ that consists of a sequence of n data points $x^i \in \mathbb{R}^m$ with a corresponding label $y^i \in L$. We iteratively divide the training data into subsets by introducing bivariate oblique splits that minimize a certain impurity measure. This process is carried out until all subsets are pure or no further splitting is possible, i.e., all points in the subsets have the same attribute values. The resulting leaf node is then assigned the most frequent label among the subset of training samples. This algorithm is summarized in Algorithm 1.

Algorithm 1 Bivariate oblique decision tree induction.

```
   Data: Set of training samples $\mathcal{X}_0$
   Result: Binary Decision Tree
1  Create root node $T_0$ associated with $\mathcal{X}_0$;
2  while $\exists$ non-pure leaf node $T_i$ and a split is possible do
3      Determine the best bivariate oblique split
       $aX_{j_1} + b \leq X_{j_2}$ on $\mathcal{X}_i$ according to a prior defined
       impurity measure;
4      Partition $\mathcal{X}_i$ into two new subsets $\mathcal{X}_{i_1}, \mathcal{X}_{i_2}$
       according to the split;
5      Split $T_i$ into two new nodes $T_{i_1}, T_{i_2}$ and associate
       each $T_{i_j}$ with a new leaf node with the
       corresponding subset $\mathcal{X}_{i_j}$ for $j = 1, 2$;
6  end
7  forall the leaf nodes $T_i$ do
8      Assign the most frequent class label in $\mathcal{X}_i$ to $T_i$;
9  end
```

The general idea of iteratively splitting the training set in a greedy fashion according to some impurity measure

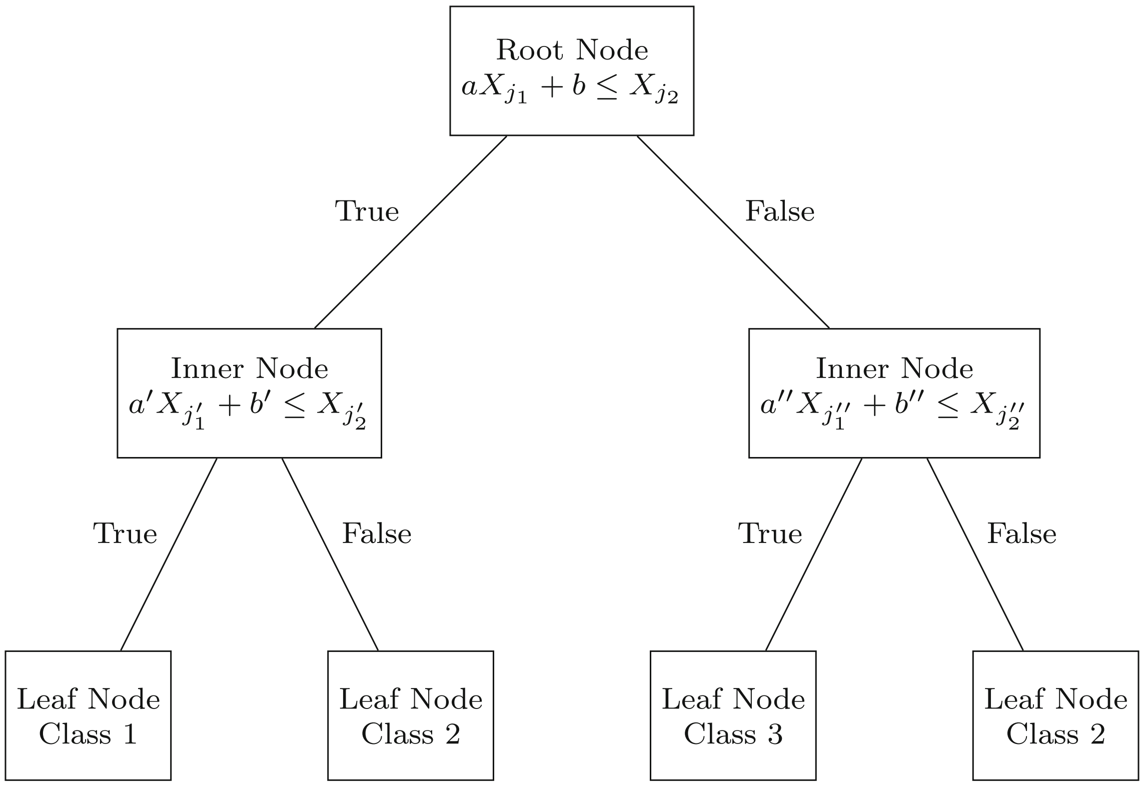


Fig. 2 Simple example of a bivariate oblique decision tree of depth two for a 3-class classification problem

in a top-down manner forms the base of many univariate and multivariate decision tree algorithms. Our major contribution lies in the technique to effectively determine optimal bivariate oblique splits which we explain in detail in Section 4. Before that, we take a closer look at the properties of the underlying impurity measures.

3.2 Concave impurity measures

Similar to common univariate decision tree methods such as CART and C4.5 our decision tree induction algorithms iteratively determines splits that minimize a certain impurity measure. In this work, we focus on so-called *concave impurity measures* as introduced in [22]. These impurity measures are concave functions $i : \mathcal{P}_k \rightarrow \mathbb{R}_{\geq 0}$ defined on the probability space $\mathcal{P}_k := \{(p_1, \ldots, p_k) \in [0,1]^k : \sum_{j=1}^k p_j = 1\}$ that satisfy $i(\lambda p + (1-\lambda)q) \geq \lambda i(p) + (1-\lambda)i(q)$ for $\lambda \in [0,1]$ and $p, q \in \mathcal{P}_k$. Moreover, for every $p \in \mathcal{P}_k$ with $p_j = 1$ for some $j \in \{1, \ldots, k\}$ it holds that $i(p) = 0$. The most popular among those concave impurity measures are:

- Classification Error: $e(p) = 1 - \max_{j=1,\ldots,k} p_j$
- Entropy: $h(p) = -\sum_{j=1}^k p_j \log p_j$
- Gini Impurity: $g(p) = 1 - \sum_{j=1}^k p_j^2$

In our tree building algorithm we iteratively split a set of training samples into two subsets. To determine the quality of the split we compute the weighted sum of impurities of these subsets. This leads to the definition of *frequency-weighted impurity measures*: Let i denote a concave impurity measure, the corresponding frequency-weighted impurity measure is the function $I : \mathbb{R}^k \rightarrow \mathbb{R}_{\geq 0}$ defined by $I(n) := \|n\|_1 \cdot i\left(\frac{n_1}{\|n\|_1}, \ldots, \frac{n_k}{\|n\|_1}\right)$. These frequency-weighted impurity measures satisfy the following properties (see [22]):

- Homogeneity: $I(\lambda n) = \lambda I(n)$
- Strict Concavity: $I(m+n) \geq I(m) + I(n)$ with equality if and only if m and n are proportional, i.e., $\left(\frac{n_1}{\|n\|_1}, \ldots, \frac{n_k}{\|n\|_1}\right) = \left(\frac{m_1}{\|m\|_1}, \ldots, \frac{m_k}{\|m\|_1}\right)$

Finally, this definition lets us define the impurity of a split $\theta := aX_{j_1} + b \leq X_{j_2}$ on a set of training samples $\mathcal{X}_0$. The split divides the training samples into two new subset $\mathcal{X}_1$ and $\mathcal{X}_2$ such that $ax^i_{j_1} + b \leq x^i_{j_2}$ for all $(x^i, y^i) \in \mathcal{X}_1$ and $ax^i_{j_1} + b > x^i_{j_2}$ for all $(x^i, y^i) \in \mathcal{X}_2$. Let $n_1 = (n_{11}, \ldots n_{1k})$, $n_2 = (n_{21}, \ldots n_{2k})$ denote the vector of absolute frequencies of class labels of the two subsets, respectively. Then, the impurity of the split is calculated as:

$$\hat{I}(\theta, \mathcal{X}_0) := I(n_1) + I(n_2)$$

4 A branch and bound algorithm for optimal bivariate oblique splits

4.1 General outline

Branch and bound algorithms are successfully used in the field of operations research to solve difficult optimization problems. In contrast to heuristic approaches, they determine global optimal solutions for problems for which a total enumeration of all feasible solutions is very time consuming or even impossible. The basic idea behind a branch and bound algorithm is to recursively divide the set of solutions of the optimization problem into two or more subsets and to compute upper and lower bounds for the value of the objective function of these subsets. This procedure can be seen as the branching of a rooted tree for which each node represents a certain subset of the search space. In case of a minimization problem, the upper bounds are typically arbitrary feasible solutions of the respective subsets. If the lower bound of any node exceeds the current best objective value we know that no solution of this node can improve the objective value and therefore, no further branching of it is necessary. This procedure is carried out until no more unexamined nodes are left. Although branch and bound procedures usually have exponential worst case-time complexity, they are widely used in practice when the bounding operations allow to reduce the domain of feasible solutions effectively.

Given a set of attributes $X_1, \ldots, X_m$, a set of labels $L = \{1, \ldots, k\}$, a training set $\mathcal{X}$ consisting of n labeled observations and an impurity measure i, the minimization problem we are trying to solve is finding features $j_1^*, j_2^* = 1, \ldots, m$ and values $a^*, b^*, \in \mathbb{R}$ such that the bivariate oblique split $\theta^* := a^* X_{j_1^*} + b^* \leq X_{j_2^*}$ minimizes $\hat{I}(\theta^*, \mathcal{X})$.

As [12] observe, the number of distinct bivariate oblique splits for n data points and two attributes is at most $O(\binom{n}{2})$. The general idea behind this result is that any two data points define a line that has to be considered as a split. This line can be slightly rotated upwards and downwards. As every distinct combination of attributes $j_1, j_2 = 1, \ldots, m$ with $j_1 \neq j_2$ has to be examined, the overall number of splits that have to be considered is in the order of $O(m^2 n^2)$. Deciding which points satisfy a rule for evaluating the impurity of a split requires an additional factor of $O(n)$. This results in an overall complexity of $O(m^2 n^3)$ to find the optimal bivariate oblique split if a brute-force algorithm is applied. Although the effort is polynomial, especially for large datasets the explicit enumeration and evaluation of all possible splits is very time consuming. It should also be kept in mind that for the top-down approach of Algorithm 1 the optimization problem has to be solved not only once, but for every non-leaf node of the resulting decision tree. Thus, an efficient algorithm to determine optimal bivariate oblique splits is crucial for the algorithm to run in a reasonable amount of time.

Up next, we describe our proposed branch and bound algorithm for determining optimal bivariate oblique splits for two attributes. This procedure is carried out for all combinations $j_1, j_2 = 1, \ldots, m$ with $j_1 \neq j_2$. We emphasize the fact that these executions are not completely separate as we can reuse the impurity of the best split found for different attributes as an initial bound for the branch and bound procedure.

Suppose, we want to find the best split $aX_{j_1} + b \leq X_{j_2}$ for the two attributes j_1 and j_2 on $\mathcal{X}$. The parameters that have to be determined are $a, b \in \mathbb{R}$. Therefore, the search space corresponds to the Euclidean plane $\mathbb{R}^2$. Note, that for any split $aX_{j_1} + b \leq X_{j_2}$ with $|a| > 1$ there is a split $a'X_{j_2} + b' \leq X_{j_1}$ with $|a'| \leq 1$ with the same impurity. For $a > 1$ the equivalent rules in terms of impurity are $a' = \frac{1}{a}$ and $b' = -\frac{b}{a} + \epsilon$ for $\epsilon > 0$ sufficiently small and for $a < -1$, the parameters are $a' = \frac{1}{a}$ and $b' = -\frac{b}{a}$. Hence, by considering any combination of attributes $j_1 \neq j_2$ instead of just $j_2 > j_1$, we can restrict the search space to $[-1, 1] \times \mathbb{R}$. As we only have to consider slopes in the range $[-1, 1]$, we can further restrict the range of the axis intercept b. Let $\underline{B} := \min\{x^i_{j_2} \pm x^i_{j_1} : i = 1, \ldots, n\}$ and $\overline{B} := \max\{x^i_{j_2} \pm x^i_{j_1} + \epsilon : i = 1, \ldots, n\}$, then any rule $aX_{j_1} + b \leq X_{j_2}$ with $a \in [-1, 1]$ and $b > \overline{B}$ will be satisfied and any rule with $a \in [-1, 1]$ and $b < \underline{B}$ will not be satisfied by all the points in the training data. Hence, it suffices to consider $[-1, 1] \times [\underline{B}, \overline{B}]$ as the overall search space for the parameters a and b. This is also the root problem of our branch and bound procedure.

Subsequently, we branch on the variables a and b to divide the problem into smaller subproblems. Hence, all subproblems that are explored can be characterized by a region $C := [\underline{a}, \overline{a}] \times [\underline{b}, \overline{b}]$ with $\underline{a}, \overline{a} \in [-1, 1]$ and $\underline{b}, \overline{b} \in [\underline{B}, \overline{B}]$. For any such region C, a feasible candidate solution can be obtained by choosing any $a \in [\underline{a}, \overline{a}]$ and $b \in [\underline{b}, \overline{b}]$ and computing $\hat{I}(aX_{j_1} + b \leq X_{j_2}, \mathcal{X})$. If this solution is better than the current best solution, we store it as the new so-called *incumbent solution*. In Section 4.6, we explain how a lower bound for any rule within the bounds of C can be established. If this lower bound is greater than or equal to the current incumbent solution, the subproblem is pruned.

One peculiarity of our algorithm is that we also introduce so-called *node exploration criteria* for the subproblems C. We say that such an exploration criteria is met, if we can efficiently enumerate all relevant rules within the subproblem's bounds. If such a criteria is met, we explore the node and prune it. On the one hand, these exploration criteria help reducing the number of nodes to explore, on the other hand, they are necessary to ensure that the branch and bound procedure terminates. We explore these exploration criteria in more detail in Section 4.3.

If a node is not pruned, we branch on it. This means that we choose $(a, b) \in (\underline{a}, \overline{a}) \times (\underline{b}, \overline{b})$ and create four new subproblems $C_1 := [\underline{a}, a] \times [\underline{b}, b]$, $C_2 := [\underline{a}, a] \times [b, \overline{b}]$, $C_3 := [a, \overline{a}] \times [\underline{b}, b]$ and $C_4 := [a, \overline{a}] \times [b, \overline{b}]$.

Clearly, any univariate split on an attribute X_{j_2} can be interpreted as a bivariate oblique split $0X_{j_1} + b \leq X_{j_2}$ for an arbitrary attribute X_{j_1}. As an optimal univariate split can be found at relatively low costs of $O(mn \log n)$ by sorting each attribute and evaluating each possible value of an attribute from left to right, it makes sense to start off the branch and bound procedure with such a split as an initial feasible solution.

Another technique we have found to be very effective is normalization. For each feature X_j, we compute the median $Med(X_j)$ and the interquartile range $IQR(X_j)$ of the training samples and replace x^i_j by $\frac{x^i_j - Med(X_j)}{IQR(X_j)}$. An optimal rule $aX'_{j_1} + b \leq X'_{j_2}$ for the transformed features $X'_1, \ldots, X'_m$, can afterwards be transformed into an optimal split $a^* X_{j_1} + b^* \leq X_{j_2}$ for the original features by choosing the values $a^* := a\frac{IQR(X_{j_2})}{IQR(X_{j_1})}$ and $b^* := -a\frac{Med(X_{j_1})IQR(X_{j_2})}{IQR(X_{j_1})} + b \cdot IQR(X_{j_2}) + Med(X_{j_2})$.

All in all our branch and bound procedure is summarized in Algorithm 2.

4.2 Preliminary definitions

Before we describe the branching strategy, the node exploration criteria and the computation of the lower and upper bounds in detail, we introduce some necessary definitions for the subproblems C of the branch and bound procedure. First, we define the set of valid rules of the subproblem as $\Theta(C) := \{aX_{j_1} + b \leq X_{j_2} : (a, b) \in C\}$.

Every subproblem divides the feature space into the three disjoint regions $R_U(C) := \{x \in \mathbb{R}^m : ax_{j_1} + b \leq x_{j_2}\ \forall (a, b) \in C\}$, $R_L := \{x \in \mathbb{R}^m : ax_{j_1} + b > x_{j_2}\ \forall (a, b) \in C\}$ and $R_I(C) := X \setminus (R_U(C) \cup R_L(C))$. Accordingly, we define $\mathcal{X}_U(C) := \{(x^i, y^i) \in \mathcal{X} : x^i \in R_U(C)\}$, $\mathcal{X}_L(C) := \{(x^i, y^i) \in \mathcal{X} : x^i \in R_L(C)\}$ and $\mathcal{X}_I(C) := \{(x^i, y^i) \in \mathcal{X} : x^i \in R_I(C)\}$ as the set of training samples in the respective regions. Note, that for all training samples in $\mathcal{X}_L(C)$ any rule in $\Theta(C)$ is satisfied and for all $\mathcal{X}_U(C)$ no rule in $\Theta(C)$ can be satisfied. Hence, we also refer to these regions as *lower* and *upper invariant*, respectively. Uncertainty regarding the location of a point relative to an optimal rule within C only remains for points in $R_I(C)$. This observation is illustrated in Fig. 3.

Algorithm 2 Branch and bound procedure to determine optimal bivariate oblique splits.

```
Input: Attributes $X_1, \dots, X_m$; training set $\mathcal{X} = \{(x^1, y^1), \dots, (x^n, y^n)\}$; impurity measure $\hat{I}$
Output: Optimal bivariate oblique split $\theta^*$
1  $\mathcal{X} := Normalize(\mathcal{X})$;
2  $\theta^* :=$ Best univariate split $0X_{j_1^*} + b^* \leq X_{j_2^*}$;
3  $ub := \hat{I}(\theta^*, \mathcal{X})$;
4  foreach $j_1, j_2 = 1, \dots, m,\ j_1 \neq j_2$ do
5      $\underline{B} := \min\{x^i_{j_2} \pm x^i_{j_1} : i = 1, \dots, n\}$;
6      $\overline{B} := \max\{x^i_{j_2} \pm x^i_{j_1} + \epsilon : i = 1, \dots, n\}$;
7      Initialize $Q$ with a root node
       $C_0 := [-1, 1] \times [\underline{B}, \overline{B}]$;
8      while $Q \neq \emptyset$ do
9          Retrieve node $C := [\underline{a}, \overline{a}] \times [\underline{b}, \overline{b}] = Q.pop()$;
10         Compute lower bound $lb := lowerbound(C)$;
11         if $lb < ub$ then
12             if exploration criteria is met then
13                 Explore and prune node $C$;
14             else
15                 Choose $a \in [\underline{a}, \overline{a}]$, $b \in [\underline{b}, \overline{b}]$ and set
                   $\theta := aX_{j_1} + b \leq X_{j_2}$;
16                 if $\hat{I}(\theta, \mathcal{X}) < ub$ then
17                     $ub := \hat{I}(\theta, \mathcal{X})$;
18                     $\theta^* := \theta$;
19                 end
20                 Choose $(a, b) \in (\underline{a}, \overline{a}) \times (\underline{b}, \overline{b})$ as
                   branching parameters and divide $C$ into
                   four new subproblems:
                   $C_1 := [\underline{a}, a] \times [\underline{b}, b]$
                   $C_2 := [\underline{a}, a] \times [b, \overline{b}]$
                   $C_3 := [a, \overline{a}] \times [\underline{b}, b]$
                   $C_4 := [a, \overline{a}] \times [b, \overline{b}]$
                   Append $C_1, C_2, C_3, C_4$ to $Q$;
21             end
22         else
23             Prune node $C$;
24         end
25     end
26 end
27 return $Denormalize(\theta^*)$
```

4.3 Node exploration criteria

As already mentioned before, we introduce node exploration criteria in our branch and bound procedure to identify situations for which we can efficiently enumerate all relevant rules within a subproblem C.

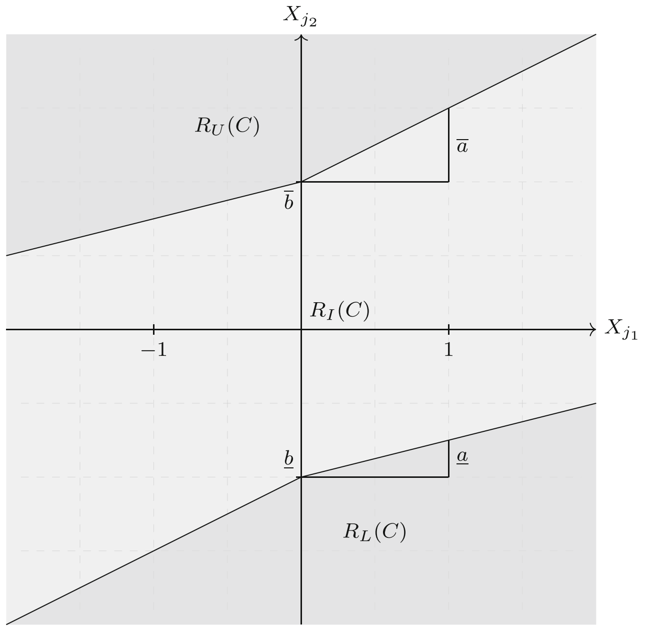


Fig. 3 Illustration of a subproblem $C = [\underline{a}, \overline{a}] \times [\underline{b}, \overline{b}]$ of the branch and bound procedure

Node Exploration Criterion 1 This criterion is met if $|\mathcal{X}_I(C)| = 0$. In this case any rule in $R(C)$ will be satisfied by the points in $\mathcal{X}_L(C)$ and violated by the points in $\mathcal{X}_U(C)$. Hence, any split within the subproblem's bounds will yield the same impurity. This means that we can simply evaluate an arbitrary feasible split of the subproblem and no further exploration is necessary.

Node Exploration Criterion 2 The second criterion is met if the points in $|\mathcal{X}_I(C)|$ are not distinguishable, i.e., it holds that $x^i_{j_1} = x^{i'}_{j_1} =: v$ and $x^i_{j_2} = x^{i'}_{j_2}$ for all $(x^i, y^i), (x^{i'}, y^{i'}) \in \mathcal{X}_I(C)$. If $v \geq 0$, we know that $\overline{a}X_{j_1} + \overline{b} \leq X_{j_2}$ or $\underline{a}X_{j_1} + \underline{b} \leq X_{j_2}$ is an optimal rule within the bounds of C. Otherwise, either $\underline{a}X_{j_1} + \overline{b} \leq X_{j_2}$ or $\overline{a}X_{j_1} + \underline{b} \leq X_{j_2}$ is optimal. Thus, it is possible to test all relevant candidate solutions of the subproblem.

Node Exploration Criterion 3 The third criterion is based on an interesting behavior of concave impurity measures found by [23]. He shows that if a set of labeled data points could be arbitrarily partitioned into two subsets, there would exist an optimal partition which does not split any class. We will generalize this result by showing that if we want to distribute a set of points among two possibly non-empty subsets such that the weighted impurity of the two resulting sets is minimal, there exists an optimal distribution that does not split any class. Before proving this result, we introduce the notation $u \circ v = (u_1v_1, \dots, u_kv_k)^T$ to denote the component-wise product of two vectors $u, v \in \mathbb{R}^k$.

Theorem 1 *Let $\mathcal{X}_L(C)$, $\mathcal{X}_U(C)$ and $\mathcal{X}_I(C)$ denote sets of labeled data points with vectors of absolute frequencies of class labels $n_L, n_U, n_I \in \mathbb{N}^k$. For any frequency-weighted impurity measure I there exist $\alpha_L, \alpha_U \in \{0,1\}^k$ with $\alpha_L + \alpha_U = \mathbf{1}$ such that*

$$I(n_L+\alpha_L \circ n_I)+I(n_U+\alpha_U \circ n_I) \leq I(n_L+\beta_L \circ n_I)+I(n_U+\beta_U \circ n_I)$$

for all $\beta_L, \beta_U \in [0,1]^k$ with $\beta_L + \beta_U = \mathbf{1}$.

Proof It is a well-known result that a concave function defined on a convex polytope attains a minimum at a vertex of the domain. Such a convex polytope is defined by $\mathcal{P} := [0,1]^k$. Moreover, we know that the composition of a concave function with an affine transformation remains concave and that the sum of concave functions is again concave. Therefore, $\mathcal{I} : \mathcal{P} \mapsto \mathbb{R}_{\geq 0}$ defined by $\mathcal{I}(\beta) := I(n_L + \beta \circ n_I) + I(n_U + (\mathbf{1} - \beta) \circ n_I)$ is a concave function. Hence, $\mathcal{I}$ is minimized by some $\alpha \in \mathcal{P} \cap \{0,1\}^k$. Clearly, this observation is equivalent to the statement of the theorem. □

Suppose all of the points in $\mathcal{X}_I(C)$ have the same class label. Theorem 1 tells us that if we could arbitrarily distribute those points among $\mathcal{X}_L(C)$ and $\mathcal{X}_U(C)$, it would be optimal to put all of the points into one of the sets. Therefore, if we find two rules $a_L X_{j_1} + b_L \leq X_{j_2}$ and $a_U X_{j_1} + b_U \leq X_{j_2}$ with $(a_L, b_L), (a_U, a_U) \in C$ such that $a_L x^i_{j_1} + b_L \leq x^i_{j_2}$ and $a_U x^i_{j_1} + b_U > x^i_{j_2}$ for all $(x^i, y^i) \in \mathcal{X}_I(C)$ we know that one of them has minimal impurity among all other rules within the bounds of C. Therefore, we can test those lines, choose the one with minimal impurity and test whether it becomes the new incumbent solution. No further analysis of the subproblem is needed.

Finding such rules or deciding that they don't exist can be carried out with the following method: If there exists $(a_L, b_L) \in C$ such that $a_L x^i_{j_1} + b_L \leq x^i_{j_2}$ for all $(x^i, y^i) \in \mathcal{X}_I(C)$, then it also holds that $a_L x^i_{j_1} + \underline{b} \leq x^i_{j_2}$. Thus, we have to solve the following system of inequalities:

$$\begin{aligned} a_L x^i_{j_1} + \underline{b} &\leq x^i_{j_2}, \text{ for all } (x^i, y^i) \in \mathcal{X}_I(C),\ x^i_{j_1} \neq 0 \\ a_L &\geq \underline{a} \\ a_L &\leq \overline{a}. \end{aligned}$$

This is equivalent to

$$a_L \leq \min \left\{ \{\overline{a}\} \cup \left\{ \frac{x^i_{j_2} - \underline{b}}{x^i_{j_1}} : (x^i, y^i) \in \mathcal{X}_I(C),\ x^i_{j_1} > 0 \right\} \right\}$$

and

$$a_L \geq \max \left\{ \{\underline{a}\} \cup \left\{ \frac{x^i_{j_2} - \underline{b}}{x^i_{j_1}} : (x^i, y^i) \in \mathcal{X}_I(C),\ x^i_{j_1} < 0 \right\} \right\}.$$

The existence of such a value a_L is easily verified. Similar arguments lead to the conclusion that there exists $a_U x^i_{j_1} + b_U > x^i_{j_1}$ if and only if there exists a_U such that

$$a_U > \max \left\{ \{\underline{a}\} \cup \left\{ \frac{x^i_{j_2} - \overline{b}}{x^i_{j_1}} : (x^i, y^i) \in \mathcal{X}_I(C),\ x^i_{j_1} > 0 \right\} \right\}$$

and

$$a_U < \min \left\{ \{\overline{a}\} \cup \left\{ \frac{x^i_{j_2} - \overline{b}}{x^i_{j_1}} : (x^i, y^i) \in \mathcal{X}_I(C),\ x^i_{j_1} < 0 \right\} \right\}.$$

Node Exploration Criterion 4 The last node exploration criteria is met, if there exist $a^*, b^* \in \mathbb{R}$ such that $a^* x^i_{j_1} + b^* = x^i_{j_2}$ for all $(x^i, y^i) \in \mathcal{X}_I(C)$. In other words, the points in $\mathcal{X}_I(C)$ are colinear. This criterion is necessary to ensure termination of the branch and bound procedure. Suppose, for example $\underline{a} \leq a^* \leq \overline{a}$ and $\underline{b} \leq b^* \leq \overline{b}$ and we are branching on (a, b) which results in the subproblems $C_1 := [\underline{a}, a] \times [\underline{b}, b]$, $C_2 := [\underline{a}, a] \times [b, \overline{b}]$, $C_3 := [a, \overline{a}] \times [\underline{b}, b]$ and $C_4 := [a, \overline{a}] \times [b, \overline{b}]$. For one of the subproblems C_l with $l \in \{1, \ldots, 4\}$ it holds that $(a^*, b^*) \in C_l$ and therefore $\mathcal{X}_I(C) = \mathcal{X}_I(C_l)$. Hence, there will always remain a subproblem for which the uncertainty set does not change. Clearly, this means that the first two node exploration criteria are not sufficient to ensure termination of the branch and bound procedure. Luckily, in the situation where all points are colinear, all relevant rules can be efficiently evaluated. This result is summarized in the following theorem and the proof yields an efficient algorithm to determine these rules.

Theorem 2 *Let $C = [\underline{a}, \overline{a}] \times [\underline{b}, \overline{b}] \subseteq \mathbb{R}^2$ with $\underline{a} < \overline{a}$ and $\underline{b} < \overline{b}$ and $\mathcal{X}_I(C)$ such that there exist $a^*, b^* \in \mathbb{R}$ with $a^* x^i_{j_1} + b^* = x^i_{j_2}$ for all $(x^i, y^i) \in \mathcal{X}_I(C)$. We can find $(a, b) \in C$ such that $\theta := a X_{j_1} + b \leq X_{j_2}$ has minimal impurity among all other splits $\theta^{prime} := a' X_{j_1} + b' \leq X_{j_2}$ with $(a', b') \in C$ in $O(|\mathcal{X}_I(C)|)$ time if the samples $(x^i, y^i) \in \mathcal{X}_I(C)$ are sorted by attribute X_{j_1}, i.e., it holds that $x^i_{j_1} \leq x^{i'}_{j_1}$ if $i < i'$. Otherwise, the time complexity is $O(|\mathcal{X}_I(C)| \log |\mathcal{X}_I(C)|)$.*

Proof Let $n_C = |\mathcal{X}_I(C)|$ and w.l.o.g. $\mathcal{X}_I(C) = \{(x^1, y^1), \ldots, (x^{n_C}, y^{n_C})\}$. Moreover, we assume that $\mathcal{X}_I(C)$ is sorted by attribute X_{j_1}. Otherwise, we employ any efficient sorting algorithm to sort the samples in $O(|\mathcal{X}_I(C)| \log |\mathcal{X}_I(C)|)$ time.

For all $(a, b) \in C$, we call $(\mathcal{X}^+_{(a,b)}, \mathcal{X}^-_{(a,b)})$ with $\mathcal{X}^+_{(a,b)} := \{(x^i, y^i) \in \mathcal{X}_I(C) : a x^i_{j_1} + b \leq x^i_{j_2}\}$ and $\mathcal{X}^-_{(a,b)} := \{(x^i, y^i) \in \mathcal{X}_I(C) : a x^i_{j_1} + b > x^i_{j_2}\}$ a linear partition of $\mathcal{X}_I(C)$. Further, let $\mathcal{P} := \{(\mathcal{X}^+_{(a,b)}, \mathcal{X}^-_{(a,b)}) : (a, b) \in C\}$ denote the set of all possible linear partitions of $\mathcal{X}_I(C)$. Clearly, lines that yield the same linear partition will have the same impurity. Therefore, to find the parameters

$(a, b) \in C$ which lead to a rule of minimal impurity we have to investigate each possible linear partition and find according parameters for the line.

Let us first assume that there is no limitation on the parameters of the line, i.e., $C = \mathbb{R}^2$. In this case $\mathcal{P}$ contains the partitions $(\mathcal{X}(C), \emptyset)$, $(\emptyset, \mathcal{X}(C))$. These partitions can, for example, be realized by choosing the parameters $a := a^*$ and $b := b^* + \epsilon$ or $b := b^* - \epsilon$, respectively. Further, $\mathcal{P}$ contains the partitions $(\{(x^{i'}, y^{i'}) : i' \leq i\}, \{(x^{i'}, y^{i'}) : i' > i\})$ and $(\{(x^{i'}, y^{i'}) : i' > i\}, \{(x^{i'}, y^{i'}) : i' \leq i\})$ for $i = 1, \dots, n_C - 1$. The lines to create these partitions have to pass a point on the line segment $L_i := \{(x, a^*x+b^*) : x^i_{j_1} \leq x < x^{i+1}_{j_1}\}$ or $L'_i := \{(x, a^*x + b^*) : x^i_{j_1} < x \leq x^{i+1}_{j_1}\}$ and need to have a greater or smaller slope than a^*, respectively.

Clearly, if we limit the range of the subproblem to $C := [\underline{a}, \overline{a}] \times [\underline{b}, \overline{b}]$ the number of possible linear partitions reduces. Therefore, we have to check which ones of these $2n_C$ partitions are still realizable.

We start off by introducing a method to find a line that goes above all points and creates the partition $(\emptyset, \mathcal{X}(C))$ or tells us that such a line does not exist. Suppose, there exists a line $(a, b) \in C$ that satisfies $ax^i_{j_1} + b > x^i_{j_2}$ for all $(x^i, y^i) \in C$. Clearly, also $ax^i_{j_1} + \overline{b} > x^i_{j_2}$ holds. In other words, we can restrict the search to lines with an axis intercept of $\overline{b}$. Due to the sorting, if $x^1_{j_1} \geq 0$ it holds that $x^{n_C}_{j_1} \geq 0$ and we choose $a := \overline{a}$. Else, if $x^{n_C}_{j_1} \leq 0$, we know that also $x^1_{j_1} \leq 0$ holds and we choose $a := \underline{a}$. In all other situations, a necessary and sufficient set of conditions for the parameter a is:

$$\begin{aligned} ax^1_{j_1} + \overline{b} &> x^1_{j_2} \\ ax^{n_C}_{j_1} + \overline{b} &> x^{n_C}_{j_2} \\ a &\geq \underline{a} \\ a &\leq \overline{a} \end{aligned}$$

Since $x^1_{j_1} < 0$ and $x^{n_C}_{j_1} > 0$ we deduce the inequalities $a < \frac{x^1_{j_2} - \overline{b}}{x^1_{j_1}}$ and $a > \frac{x^{n_C}_{j_2} - \overline{b}}{x^{n_C}_{j_1}}$. Therefore, if $a_1 := \max\left\{\frac{x^{n_C}_{j_2} - \overline{b}}{x^{n_C}_{j_1}}, \underline{a}\right\} \geq \min\left\{\frac{x^1_{j_2} - \overline{b}}{x^1_{j_1}}, \overline{a}\right\} =: a_2$, no line above all of the points is possible. Otherwise, we choose $a := \frac{a_1 + a_2}{2}$.

Finding a line beneath all points to create the partition $(\mathcal{X}(C), \emptyset)$ is analogous to the case discussed above. If $x^1_{j_1} \geq 0$, we choose $a := \underline{a}$ and $b := \underline{b}$, if $x^{n_C}_{j_1} \leq 0$, we choose $a := \overline{a}$ and $b := \underline{b}$. Otherwise, we check whether $a_1 := \max\left\{\frac{x^1_{j_2} - \underline{b}}{x^1_{j_1}}, \underline{a}\right\} > \min\left\{\frac{x^{n_C}_{j_2} - \underline{b}}{x^{n_C}_{j_1}}, \overline{a}\right\} =: a_2$. If this evaluates to true, no line beneath all points exists. Otherwise, $a := \frac{a_1 + a_2}{2}$ and $b := \underline{b}$ yields the desired partition.

Next, we want to find lines for the partitions $(\{(x^{i'}, y^{i'}) : i' \leq i\}, \{(x^{i'}, y^{i'}) : i' > i\})$ for $i = 1, \dots, n_C - 1$ or decide that such a line does not exist. This is equivalent to solving the following linear program for sufficiently small $\epsilon > 0$:

$$\begin{aligned} (LP) \quad & \max\ \delta_1 \\ & \text{s.t.}\ (\underline{a} + \delta_a)x^i_{j_1} + (\underline{b} + \delta_b) + \delta_1 = x^i_{j_2} \\ & \qquad (\underline{a} + \delta_a)x^{i+1}_{j_1} + (\underline{b} + \delta_b) - \delta_2 - \epsilon = x^{i+1}_{j_2} \\ & \qquad \delta_a + \delta_3 = \overline{a} - \underline{a} \\ & \qquad \delta_b + \delta_4 = \overline{b} - \underline{b} \\ & \qquad \delta_a, \delta_b, \delta_1, \delta_2, \delta_3, \delta_4 \geq 0 \end{aligned}$$

There exists a line for the partition if and only if (LP) has an optimal solution with an objective value greater than zero. Such a line is then defined by $a := \underline{a} + \delta^*_a$ and $b := \underline{b} + \delta^*_b$ for any feasible solution $\delta^*_a, \delta^*_b, \delta^*_1, \delta^*_2, \delta^*_3, \delta^*_4 \geq 0$ with $\delta^*_1 > 0$. If (LP) has an optimal solution with $\delta_1 > 0$ then it also has an optimal basic solution with $\delta_1 > 0$. As $\underline{a} < \overline{a}$, we further know that in an optimal basic solution either δ_a or δ_3 must be part of the basis. Similarly, we know that δ_b or δ_4 must be part of the basis. Hence, there remain eight bases to be checked: $\{\delta_1, \delta_a, \delta_b, \delta_2\}$, $\{\delta_1, \delta_a, \delta_b, \delta_3\}$, $\{\delta_1, \delta_a, \delta_b, \delta_4\}$, $\{\delta_1, \delta_3, \delta_b, \delta_2\}$, $\{\delta_1, \delta_3, \delta_b, \delta_4\}$, $\{\delta_1, \delta_a, \delta_4, \delta_2\}$, $\{\delta_1, \delta_a, \delta_4, \delta_3\}$, $\{\delta_1, \delta_3, \delta_4, \delta_2\}$.

Similarly, finding a line $(a, b) \in C$ such $(\mathcal{X}^+_{(a,b)}, \mathcal{X}^-_{(a,b)}) = (\{(x^{i'}, y^{i'}) : i' > i\}, \{(x^{i'}, y^{i'}) : i' \leq i\})$ for $i = 1, \dots, n_C - 1$ is equivalent to solving the linear program

$$\begin{aligned} (LP) \quad & \max\ \delta_1 \\ & \text{s.t.}\ (\underline{a} + \delta_a)x^{i+1}_{j_1} + (\underline{b} + \delta_b) + \delta_1 = x^{i+1}_{j_2} \\ & \qquad (\underline{a} + \delta_a)x^i_{j_1} + (\underline{b} + \delta_b) - \delta_2 - \epsilon = x^i_{j_2} \\ & \qquad \delta_a + \delta_3 = \overline{a} - \underline{a} \\ & \qquad \delta_b + \delta_4 = \overline{b} - \underline{b} \\ & \qquad \delta_a, \delta_b, \delta_1, \delta_2, \delta_3, \delta_4 \geq 0 \end{aligned}$$

for small $\epsilon > 0$ and the same set of basic solutions has to be evaluated as in the previous case.

Altogether, we introduced a constant time method for all of the $2n_C$ partitions P which either yields parameters (a, b) such that $(\mathcal{X}^+_{(a,b)}, \mathcal{X}^-_{(a,b)}) = P$ or decides that the partition is not possible. □

The necessity of sorting the data points in each subproblem can be avoided easily. Before starting the tree building algorithm one creates m arrays of pointers to the actual training data. These arrays are then sorted according to the values of the respective attribute. From that point on one can work with these arrays of pointers instead of the actual values of the training data.

4.4 Branching strategy

In this section, we present the strategy for choosing the parameter a or b used for branching a subproblem $C := [\underline{a}, \overline{a}] \times [\underline{b}, \overline{b}]$. We require the computation of the parameters to be simple, i.e., the time complexity should be $O(|\mathcal{X}_I(C)|)$, because we found that more elaborate branching strategies could not compensate the drawback of being able to process less subproblems in the same amount of time.

Branching strategy 1 The simplest, most obvious choice for the parameters is $a := \frac{\underline{a}+\overline{a}}{2}$ and $b := \frac{\underline{b}+\overline{b}}{2}$. A shortcoming of this strategy is that it does not take into account the position of the points in $R_I(C)$. Hence, we develop a more elaborate strategy that is still fairly easy to compute.

Branching strategy 2 In the previous section, we introduced node exploration criteria for situations in which there are no points in $\mathcal{X}_I(C)$ or the points in this set are not distinguishable. Hence, it would be desirable if the number of uncertain datapoints of the resulting subproblems would be as small as possible. In the best case, we would evenly distribute the points in $\mathcal{X}_I(C)$ among the four subproblems. Obviously, this is not always possible. It is also very likely that certain points will have to be in the region of uncertainty of two or more subproblems, i.e., for $(x^i, y^i) \in \mathcal{X}_I(C)$ and two subproblems C' and C'' of C it holds that $(x^i, y^i) \in \mathcal{X}_I(C')$ and $(x^i, y^i) \in \mathcal{X}_I(C'')$. Still, it does make sense to approximate the desired aim by choosing some sort of mean parameters for branching which we construct as follows: For each $(x^i, y^i) \in \mathcal{X}_I(C)$, we find $a^i \in [\underline{a}, \overline{b}]$ and $b^i \in [\underline{b}, \overline{b}]$ such that $a^i x^i_{j_1} + b^i = x^i_{j_2}$. As values for b^i, we choose the mean intercept of all possible lines that pass through the point $(x^i_{j_1}, x^i_{j_2})$ that have a slope within the interval $[\underline{a}, \overline{a}]$:

$$b^i := \begin{cases} \frac{\max\{\underline{b}, x^i_{j_2} - \overline{a}x^i_{j_1}\} + \min\{\overline{b}, x^i_{j_2} - \underline{a}x^i_{j_1}\}}{2}, \text{ if } x^i_{j_1} > 0 \\ \frac{\max\{\underline{b}, x^i_{j_2} - \underline{a}x^i_{j_1}\} + \min\{\overline{b}, x^i_{j_2} - \overline{a}x^i_{j_1}\}}{2}, \text{ if } x^i_{j_1} < 0 \\ x^i_{j_2}, \text{ if } x^i_{j_1} = 0 \end{cases}$$

It follows that $a^i = \frac{x^i_{j_2} - b^i}{x^i_{j_1}}$ if $x^i_{j_1} \neq 0$. If $x^i_{j_1} = 0$ we set $a^i = \frac{\underline{a}+\overline{a}}{2}$. Subsequently, we choose the respective means

$$a := \sum_{(x^i, y^i) \in \mathcal{X}_I(C)} \frac{a^i}{|\mathcal{X}_I(C)|} \text{ and } b := \sum_{(x^i, y^i) \in \mathcal{X}_I(C)} \frac{b^i}{|\mathcal{X}_I(C)|}$$

as the values for branching. The idea of this construction is illustrated in Fig. 4. In our experiments we found that this simple strategy outperforms the initial idea of choosing $a := \frac{\underline{a}+\overline{a}}{2}$ and $b := \frac{\underline{b}+\overline{b}}{2}$ and due to the linear complexity it does not significantly slow down the branch and bound procedure.

4.5 Termination of the algorithm

An important issue to be addressed for a branch and bound algorithm over a continuous domain is the termination of the algorithm. The following theorem states that our algorithm does in fact terminate after a finite number of iterations for the first branching strategy.

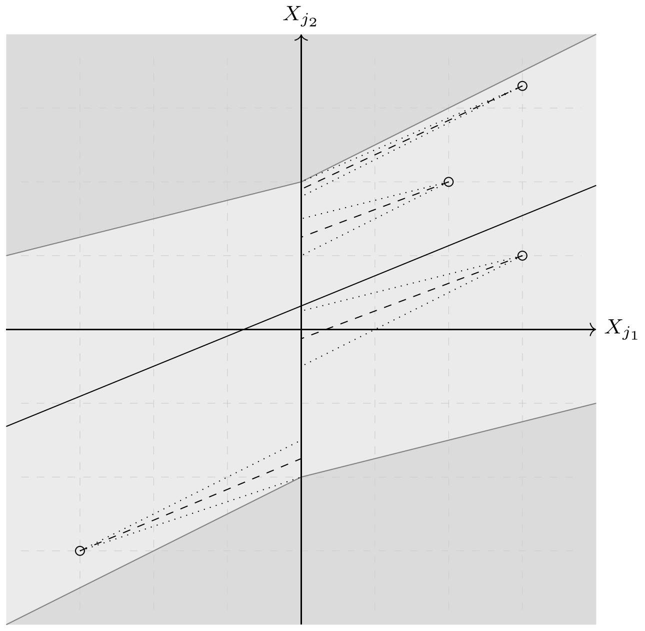


Fig. 4 The dashed lines illustrate the lines $a^i X_{j_1} + b^i = X_{j_2}$ and the solid line illustrates the line $aX_{j_1} + b = X_{j_2}$ from which the branching parameters are derived

Theorem 3 *If for each subproblem $C = [\underline{a}, \overline{a}] \times [\underline{b}, \overline{b}]$ the point $(\frac{\underline{a}+\overline{a}}{2}, \frac{\underline{b}+\overline{b}}{2})$ is chosen for branching, the algorithm terminates after a finite number of iterations.*

Proof To proof this theorem, we have to show that the branch and bound procedure terminates for each individual pair of attributes $j_1, j_2 \in \{1, \dots, m\}$ such that $j_1 \neq j_2$. More specifically, for each pair of attributes, the while loop at line 8 in Algorithm 2 must terminate after a finite number of iterations. Let's assume that this is not the case. Then, there exist attributes j_1, j_2 with $j_1 \neq j_2$ and a sequence $(C_k)_{k\in\mathbb{N}} = ([\underline{a}_k, \overline{a}_k] \times [\underline{b}_k, \overline{b}_k])_{k\in\mathbb{N}}$ of consecutive subproblems for that particular pair of attributes such that $C_{k+1} \subset C_k$ and none of these subproblems satisfies one of the node exploration criteria. As the first three node exploration criteria are not satisfied the sets $\mathcal{X}_I(C_k)$ contain at least two distinct points and these sets remain equal for $k \geq k_I$ for some specific $k_I \in \mathbb{N}$. W.l.o.g we can therefore assume that there exists $\tilde{\mathcal{X}} \subseteq \mathcal{X}$ such that $\mathcal{X}_I(C_k) = \tilde{\mathcal{X}}$ for all $k \in \mathbb{N}$. Due to the branching strategy, the sequence converges towards a single point $(\tilde{a}, \tilde{b}) \in \mathbb{R}^2$ and for each $\epsilon > 0$ there exists $k_\epsilon \in \mathbb{N}$ such that for each $k \geq k_\epsilon$ it holds that $C_k \subseteq [\tilde{a} - \epsilon, \tilde{a} + \epsilon] \times [\tilde{b} - \epsilon, \tilde{b} + \epsilon]$. As the fourth node exploration criterion is not satisfied, there exists at least one point $(x^i, y^i) \in \tilde{\mathcal{X}}$ such that $\tilde{a}x^i_{j_1} + \tilde{b} \neq x^i_{j_2}$. We define

$d := |\tilde{a}x^i_{j_1} + \tilde{b} - x^i_{j_2}|$. Clearly, it holds that $d > 0$ Then, for $\epsilon = \frac{d}{2(|x^i_{j_1}|+1)}$, and any $(a', b') \in [\tilde{a}-\epsilon, \tilde{a}+\epsilon] \times [\tilde{b}-\epsilon, \tilde{b}+\epsilon]$ we conclude:

– Case 1: $x^i_{j_1} \geq 0$ and $\tilde{a}x^i_{j_1} + \tilde{b} - x^i_{j_2} > 0$

$$\begin{aligned} a'x^i_{j_1} + b' - x^i_{j_2} &\geq (\tilde{a} - \epsilon)x^i_{j_1} + (\tilde{b} - \epsilon) - x^i_{j_2} \\ &\geq \tilde{a}x^i_{j_1} + \tilde{b} - x^i_{j_2} - \epsilon(x^i_{j_1} + 1) = d - \frac{d}{2} = \frac{d}{2} > 0 \end{aligned}$$

– Case 2: $x^i_{j_1} < 0$ and $\tilde{a}x^i_{j_1} + \tilde{b} - x^i_{j_2} > 0$

$$a'x^i_{j_1} + b' - x^i_{j_2} \geq \tilde{a}x^i_{j_1} + \tilde{b} - x^i_{j_2} - \epsilon(-x^i_{j_1} + 1) = d - \frac{d}{2} = \frac{d}{2} > 0$$

– Case 3: $x^i_{j_1} \geq 0$ and $\tilde{a}x^i_{j_1} + \tilde{b} - x^i_{j_2} < 0$

$$a'x^i_{j_1} + b' - x^i_{j_2} \leq \tilde{a}x^i_{j_1} + \tilde{b} - x^i_{j_2} + \epsilon(x^i_{j_1} + 1) = -d + \frac{d}{2} = -\frac{d}{2} < 0$$

– Case 4: $x^i_{j_1} < 0$ and $\tilde{a}x^i_{j_1} + \tilde{b} - x^i_{j_2} < 0$

$$a'x^i_{j_1} + b' - x^i_{j_2} \leq \tilde{a}x^i_{j_1} + \tilde{b} - x^i_{j_2} + \epsilon(-x^i_{j_1} + 1) = -d + \frac{d}{2} = -\frac{d}{2} < 0$$

Thus, in all of the four cases there is no $(a', b') \in [\tilde{a} - \epsilon, \tilde{a} + \epsilon] \times [\tilde{b} - \epsilon, \tilde{b} + \epsilon]$ such that $a'x^i_{j_1} + b' = x^i_{j_2}$. Hence, $(x^i, y^i) \notin \mathcal{X}_I(C_k)$ for all $k \geq k_\epsilon$ which contradicts the assumption that $(x^i, y^i) \in \tilde{\mathcal{X}}$ which is equivalent to $(x^i, y^i) \in \mathcal{X}_I(C_k)$ for all $k \in \mathbb{N}$. □

The second branching strategy presented in Section 4.4 should be regarded as a suggestion to improve the runtime of the algorithm in practice. The idea is purely heuristic and for that reason it is provided without theoretical justification. Nevertheless, we want to point out the fact that due to Theorem 3 we can ensure termination for any branching strategy by combining it appropriately with Branching Strategy 1.

4.6 Upper and lower bounds

Upper bounds The computation of an upper bound for the currently observed subproblem can be carried out by choosing arbitrary parameters a and b such that $(a, b) \in C$ and compute the impurity of the split $aX_{j_1} + b \leq X_{j_2}$. In our implementation we choose the values used for branching the subproblem.

Lower bounds For the lower bound of a subproblem C we exploit the concavity of the impurity measures. In fact, it turns out that the weighted impurity of the two invariants is a lower bound for any rule $aX_{j_1} + b \leq X_{j_2}$ with $(a, b) \in C$.

Theorem 4 *Let I denote a frequency-weighted impurity measure and let $\hat{I}$ denote the corresponding impurity measure for a bivariate oblique split on the training set $\mathcal{X}$. For a subproblem $C \subseteq \mathbb{R}^2$ let n_L and n_U denote the vector of absolute frequencies of class labels of the sets $\mathcal{X}_L(C)$ and $\mathcal{X}_U(C)$, respectively. Further, let $\theta := aX_{j_1} + b \leq X_{j_2}$ with $(a, b) \in C$ denote a valid rule within the bounds of the subproblem Then, it holds that*

$$I(n_L) + I(n_U) \leq \hat{I}(\theta, \mathcal{X})$$

Proof The rule θ partitions $\mathcal{X}_I(C)$ into two subsets $\mathcal{X}_I^-(C), \mathcal{X}_I^+(C)$ with class frequencies m_L and m_U. It divides the overall set of training samples $\mathcal{X}$ into two subsets $\mathcal{X}_L$ and $\mathcal{X}_U$ such that $\mathcal{X}_L = \mathcal{X}_I^-(C) \dot{\cup} \mathcal{X}_L(C)$ and $\mathcal{X}_U = \mathcal{X}_I^+(C) \dot{\cup} \mathcal{X}_U(C)$. Hence, the impurity of the split is

$$\hat{I}(\theta, \mathcal{X}) = I(n_L + m_L) + I(n_U + m_U).$$

Using strict concavity of I, we have

$$\begin{aligned} I(n_L+m_L)+I(n_U + m_U) &\geq I(n_L)+I(m_L)+I(n_U)+I(m_U) \quad (1) \\ &\geq I(n_L) + I(n_U). \quad (2) \end{aligned}$$

□

4.7 Improved lower bounds for specific impurity measures

In this section we present two improved lower bounds for the popular impurity measures entropy and gini impurity. The improvements are based on the work of [23] who summarizes useful properties of concave impurity measures.

Theorem 4 states that the weighted impurity of the invariants is a lower bound for any rule of the examined subproblem C. In its proof, we ignored the sum $I(m_L) + I(m_U)$ to get from (1) and (2). Thus, if we can lower bound this sum by some value greater than zero, we can improve the lower bound of the subproblem. Note, that this expression corresponds to the impurity of a feasible partition of the points in $\mathcal{X}_I(C)$ into two subsets. Recalling Theorem 1, we know that there exists an optimal partition of these points that does not split any class. Those optimal partitions, however, can be very different depending on the underlying impurity measure.

For the entropy, Breiman shows that an optimal partition would distribute the points evenly among the subsets. More precisely, a partition $(\mathcal{X}_L, \mathcal{X}_U)$ of $\mathcal{X}_I(C)$ such that $\mathcal{X}_L \dot{\cup} \mathcal{X}_U = \mathcal{X}_I(C)$ with respective vectors of absolute frequencies of the class labels m_L and m_U which does not split any class minimizes the weighted entropy $E(m_L) + E(m_U)$ if it minimizes $\max\{|\mathcal{X}_L|, |\mathcal{X}_U|\}$. If m denotes the vector of absolute frequencies of class labels of $\mathcal{X}_I(C)$ the result can be derived from the following equation:

$$E(m_L)+E(m_U) = -\sum_{r\in\{L,U\}} |\mathcal{X}_r| \sum_{\substack{j=1,\dots,k:\\ m_{r_j}=m_j}} \frac{m_j}{|\mathcal{X}_r|}\log\frac{m_j}{|\mathcal{X}_r|}$$
$$= -\sum_{j=1}^{k} m_j\log m_j + \sum_{r\in\{L,U\}} |\mathcal{X}_r|\log|\mathcal{X}_r|$$

The first equation stems from the fact either m_{Lj} or m_{Rj} is equal to m_j as the we only consider partitions which do not split any class. Unfortunately, determining such an optimal partition is equivalent to solving the well-known NP-hard partition problem. Nevertheless, the second part of the equation is clearly bounded by the expression $|\mathcal{X}_I(C)|\log\frac{|\mathcal{X}_I(C)|}{2}$ which motivates the following theorem.

Theorem 5 *Let E denote the frequency weighted entropy impurity measure and let $\hat{E}$ denote the corresponding impurity measure for a bivariate oblique split on the training set $\mathcal{X}$. For a subproblem $C\subseteq\mathbb{R}^2$ let n_L and n_U denote the vector of absolute frequencies of the class labels in the invariants. Further, let m denote the vector of absolute frequencies of $\mathcal{X}_I(C)$ and $M := |\mathcal{X}_I(C)|$. For any rule $\theta := aX_{j_1}+b\le X_{j_2}$ such that $(a,b)\in C$ it holds that*

$$\max\left\{\sum_{r\in\{L,U\}} E(n_r),\ \sum_{r\in\{L,U\}} E(n_r) - \sum_{j=1}^{k} m_j\log m_j + M\log\frac{M}{2}\right\} \le \hat{E}(\theta,\mathcal{X}).$$

For the gini impurity Breiman proves that an optimal partition separates the maximum class from the other classes. This observation leads to the following theorem.

Theorem 6 *Let G denote the frequency weighted gini impurity measure and let $\hat{G}$ denote the corresponding impurity measure for a bivariate oblique split on the training set $\mathcal{X}$. Let C, n_L, n_U and m as in Theorem 5. Furthermore, let $j^* := \arg\max_{j=1,\dots,k} m_j$. For any rule $\theta := aX_{j_1}+b\le X_{j_2}$ such that $(a,b)\in C$ it holds that*

$$\sum_{r\in\{L,U\}} G(n_r) + G((m_1,\dots,m_{j^*-1},0,m_{j^*+1},\dots,m_k)) \le \hat{G}(\theta,\mathcal{X}).$$

These improved lower bounds are especially useful in the early stage of the branch and bound procedure when the number of points in the invariants is still quite low. In this stage, the lower bound stated in Theorem 4 is quite small and the additionally added term will have a high proportion of the value of the improved bound. At a later stage, when the invariants contain many points the original term which corresponds to the lower bound stated in Theorem 4 becomes dominant as the invariants already yield a good picture of any feasible split of the subproblem.

5 Extensions for the tree building algorithm

5.1 Early stopping

As the outer loop of the decision tree induction algorithm (see Algorithm 1) is a greedy approach, global optimality of the splits might sometimes not be absolutely essential and one might want to speed up the algorithm instead. Nevertheless, even in these situations it is desirable to be close to the global optimum and to have a global quality guarantee. Heuristic approaches which yield local optima usually don't provide such a guarantee and it is not clear whether the local optimal solution's impurity is close to the best value possible. Branch and bound on the other hand, although belonging to the class of exact algorithms for solving optimization problems, can also be employed heuristically while providing a global quality guarantee. Instead of pruning subproblems only when the lower bound lb is smaller than or equal to the upper bound ub, one can prune whenever $lb \ge (1-\epsilon)ub$ for $\epsilon\in[0,1]$. For such nodes, no significant improvement is possible. Although this might lead to pruning the optimal solution, we still know that an optimal solution's objective value can only improve the final solution by a factor of $(1-\epsilon)$. That way the decision tree induction can be sped up while ensuring that the bivariate splits' are provably near optimal.

5.2 Parallelization

Another benefit of branch and bound algorithms is the fact that they are easily parallelizable to make the best use of modern multi-core hardware. Each core can process an individual subproblem while keeping the current incumbent solution as a shared resource. For our specific problem of finding the best bivariate oblique split, we propose parallelization at attribute level, i.e., the cores are assigned tuples of attributes and run a full branch and bound procedure for them (steps 4-27 of Algorithm 2).

5.3 Categorical data

Bivariate oblique splits are only applicable for numerical data. However, in many application the data to be analyzed contains categorical features with no specific ordering. For such situations we propose to use a so-called one-hot encoding. For each category, we add an additional feature which is equal to one if the data sample belongs to the respective category and zero, otherwise. The transformed dataset can then be processed by our presented algorithm.

5.4 Decision tree pruning

When decision trees are fully branched, they are usually overfitted. This means that they describe the training data perfectly but are incapable of predicting well on unseen data. A common technique to overcome this problem is to use pruning methods. An overview over different pruning methods is presented in [24]. These can be categorized into two classes, pre- and post-pruning methods. Pre-pruning methods stop the tree building procedure when certain criteria are met. For example, if the reduction of the impurity is below a certain threshold. Post-pruning methods, on the other hand, are applied after the tree has been fully grown. If nodes of the tree satisfy certain criteria, they are pruned. These techniques are independent of the type of split used for building the tree. Therefore, they are all applicable with our method and similar results as for other decision tree induction methods are expected in terms of prediction accuracy.

6 Evaluation

6.1 Experimental setup

In this section, we provide a comprehensive evaluation of our proposed method. We compare our method (BiDT) to our own implementation of the well-known greedy top-down univariate decision tree building method (DT) which is also implemented by CART and C4.5 and to the OC1 decision tree induction algorithm [12]. The latter is probably the most popular oblique decision tree induction method which uses a randomized hill climbing approach to determine local optimal multidimensional hyperplanes to divide the feature space. Using our own implementation for the univariate trees ensures maximal comparability to the bivariate oblique trees as we are basically just omitting the branch and bound procedure. We implemented our algorithm in C++ and we carried out all of the experiments on a computer equipped with an Intel Xeon E3-1231v3 @3.40GHz (4 Cores), 32GB DDR3 RAM running Ubuntu 18.04.

For our experiments we test the three methods on 20 popular datasets from the UCI Learning Repository [25]. The collection contains datasets of variable size ranging from 80 to 25010 data points, 6 to 256 attributes and 2 to 16 classes. An overview over the examined datasets is presented in Table 1. Note, that we report the number of attributes after the application of the one-hot encoding scheme for categorical features and we dropped samples with missing values beforehand.

We compare the algorithms with and without an additional post-pruning step. The chosen pruning method is minimal cost-complexity pruning introduced by [8]. We perform 10-fold cross validation to retrieve the following three quality measures:

Table 1 Dataset description

Dataset	Data points	Attributes	Classes
breast_cancer_wisconsin	683	9	2
chess_krvskp	3196	73	2
congressional_voting	232	16	2
dermatology	358	34	6
ecoli	336	7	8
glass	214	9	6
ionosphere	351	34	2
monks_problems_1	124	6	2
optical_recognition	3823	63	16
parkinsons	195	22	2
pen_based_recognition	7494	16	10
pima_indian_diabetes	768	8	2
poker_hand	25010	10	10
seeds	210	7	3
semeion	1593	256	10
spect_heart	80	22	2
statlog_heart	270	13	2
statlog_landsat	4435	36	6
tic_tac_toe	958	27	2
wine	178	13	3

Table 2 Comparison of accuracy between unpruned BiDT, OC1 and DT

Dataset	Accuracy (%)		
	BiDT	OC1	DT
breast_cancer_wisconsin	**94.74**±3.06	94.0±3.48	94.15±1.45
chess_krvskp	**99.56**±0.29	98.53±0.47	99.56±0.4
congressional_voting	**94.87**±4.96	93.53±3.5	94.84±5.41
dermatology	**97.21**±3.2	95.56±3.06	94.42±3.34
ecoli	77.13±4.99	81.6±4.78	**82.91**±5.15
glass	**72.53**±9.47	69.2±9.49	68.81±12.14
ionosphere	**90.58**±5.58	89.46±3.14	88.02±2.52
monks_problems_1	**100.0**±0.0	89.49±8.92	81.28±11.95
optical_recognition	**94.76**±1.55	94.68±1.25	94.71±1.4
parkinsons	**87.24**±7.33	85.68±8.32	84.16±12.33
pen_based_recognition	96.54±0.53	**96.68**±0.5	96.38±0.49
pima_indian_diabetes	68.5±6.53	69.16±5.79	**71.23**±7.07
poker_hand	**85.17**±1.43	54.18±0.88	49.89±1.01
seeds	**95.71**±3.33	91.91±6.41	91.9±6.04
semeion	**76.96**±3.03	75.27±2.74	74.82±3.6
spect_heart	68.75±16.06	**71.25**±14.84	67.5±8.29
statlog_heart	**75.93**±5.56	74.81±5.69	74.81±6.99
statlog_landsat	85.59±1.46	**85.66**±1.23	84.44±2.32
tic_tac_toe	94.68±2.05	**94.78**±2.6	93.73±2.45
wine	**90.46**±7.05	88.3±9.09	88.86±10.52
Average Rank	**1.4**	2.1	2.5

- Out-of-sample accuracy
- Tree size in terms of the number of leaf nodes
- Required time for building the trees

We report the runtime of our bivariate oblique decision tree induction algorithm if the branch and bound procedures are carried out by a single thread as well as when they are carried out in parallel by 4 threads simultaneously (see Section 5.2).

Following Demšar's proposal [26] for comparing multiple classifiers we perform a Friedman test (see [27, 28]) to check whether there is a significant difference for at least two of the methods and subsequently, if a significant difference is detected, we execute the Holm test (see [29]) with BiDT as a control method for a post-hoc analysis.

The Friedman test is a non-parametric statistical test based on ranks to detect differences across measurements taken by three or more different methods. For each individual dataset the performances are ranked from best to worst, i.e., the best classifier has rank one, the second best has rank two and so forth. Ties are broken by assigning the average rank. The null-hypothesis being tested is that the classifiers perform equally well and the differences are due to random error. Thus, if the null-hypothesis is true the observed ranks should be equal. If the p-value computed by the Friedman test is below a certain significance level α we reject the null-hypothesis and conclude that at least two methods differ significantly.

If a significant difference is detected by the Friedman test, the Holm test let's us decide whether the control method BiDT performs significantly better or worse than the other algorithms at a given significance level α. For each of the methods that BiDT should be compared to, the null-hypotheses is that the method's performance is equal to BiDT's. The p-values for these null-hypotheses are computed and ordered from lowest to highest. To control the family-wise error rate, the Holm test adjusts these p-values by multiplying it with $(k + 1 - i)$ where k denotes

Table 3 Holm post-hoc test for accuracy of unpruned trees

Algorithm	Rank	Holm adjusted p-value	Hypothesis
OC1	2.1	0.03	Rejected
DT	2.5	1×10^{-3}	Rejected
Control method: BiDT, Rank: 1.4			

Table 4 Comparison of accuracy between pruned BiDT, OC1 and DT

Dataset	Accuracy (%)		
	BiDT	OC1	DT
breast_cancer_wisconsin	**96.34**±1.99	95.02±2.3	95.76±2.88
chess_krvskp	**99.5**±0.42	98.12±1.04	99.25±0.58
congressional_voting	**97.01**±2.7	94.84±3.76	**97.01**±2.7
dermatology	**95.24**±3.18	94.44±3.91	92.75±3.85
ecoli	78.78±4.71	**82.77**±3.95	81.07±4.86
glass	72.85±6.32	63.53±9.14	**74.62**±8.42
ionosphere	**92.58**±4.28	86.04±3.93	88.04±5.53
monks_problems_1	**98.4**±3.21	85.32±16.53	79.04±15.96
optical_recognition	94.79±1.49	94.45±1.34	**94.94**±1.23
parkinsons	**87.18**±8.67	87.16±6.94	84.08±9.64
pen_based_recognition	**96.66**±0.71	96.18±0.57	95.61±0.49
pima_indian_diabetes	**74.24**±6.74	73.44±4.79	73.32±6.44
poker_hand	**86.6**±1.7	60.22±2.27	54.57±1.75
seeds	**94.76**±4.49	94.29±4.15	91.9±7.39
semeion	**75.58**±4.5	75.26±3.09	74.83±4.02
spect_heart	65.0±20.0	**67.5**±18.71	60.0±18.37
statlog_heart	74.44±5.75	78.52±3.23	**81.85**±4.81
statlog_landsat	**86.47**±1.93	86.02±1.52	85.95±1.56
tic_tac_toe	**94.78**±2.04	93.12±3.64	92.89±3.61
wine	90.46±5.72	**93.27**±6.04	87.22±9.31
Average Rank	**1.425**	2.15	2.425

the number of methods different from the control method and i denotes the position in the ordering. Thus, smaller p-values are increased more strongly than higher ones. Then, starting from the most significant null-hypothesis (based on the original p-values) the adjusted p-values are compared to the significance level until it is exceeded for the first time. All null-hypotheses before that are rejected and the others are not and we can conclude which methods in fact differ significantly from the control method.

In our experiments we require a significance level of $\alpha = 0.05$ for both, the Friedman and the Holm test.

6.2 Comparison of accuracy

In this section, we compare the accuracy of the three methods. An overview of our results for the unpruned trees can be found in Table 2 and for the pruned trees in Table 4. We report the average accuracy together with the standard deviation resulting from the 10-fold cross validation. The best value for each dataset is highlighted in bold. The last line of the table reports the average rank computed over all the datasets.

For the unpruned decision trees BiDT is in first place with a rank of 1.4, OC1 is in second place with a rank of 2.1 and DT is in the third place with a rank of 2.5 (Table 2). The Friedman test confirms that there is a significant difference for at least two average ranks with a p-value of $2 \times 10^{-3} < \alpha$. A summary of the subsequent Holm test with BiDT as control method is presented in Table 3 and we can see that both OC1's and DT's accuracy differs significantly from BiDT's. Thus, we conclude that BiDT is the most accurate of the three methods for the datasets at hand.

We see similar results for the pruned decision trees. BiDT has the lowest rank of 1.425, OC1 has rank 2.15 and DT has

Table 5 Holm post-hoc test for accuracy of pruned trees

Algorithm	Rank	Holm adjusted p-value	Hypothesis
OC1	2.15	0.02	Rejected
DT	2.425	3×10^{-3}	Rejected
Control method: BiDT, Rank: 1.425			

Table 6 Comparison of tree size between unpruned BiDT, OC1 and DT

Dataset	Leaves		
	BiDT	OC1	DT
breast_cancer_wisconsin	**15.7**±1.19	16.1±1.04	26.4±1.8
chess_krvskp	29.4±1.1	**11.2**±4.19	47.8±1.83
congressional_voting	7.7±0.9	**3.3**±0.46	10.4±0.66
dermatology	**8.8**±0.6	9.1±1.3	15.8±1.54
ecoli	**31.0**±1.9	34.2±1.83	45.0±2.28
glass	**25.7**±1.33	29.1±1.81	37.2±1.47
ionosphere	**14.1**±1.36	15.3±2.05	21.7±2.33
monks_problems_1	**5.2**±0.4	6.0±1.61	18.5±3.56
optical_recognition	**71.9**±2.17	89.3±3.13	102.1±2.47
parkinsons	**8.8**±0.6	12.8±1.89	12.5±1.43
pen_based_recognition	114.8±2.69	**98.8**±6.84	204.7±6.37
pima_indian_diabetes	**73.0**±4.37	74.8±4.85	107.8±6.76
poker_hand	**1642.86**±82.21	3596.79±62.97	6391.36±105.85
seeds	**6.4**±0.49	9.0±0.89	12.8±1.08
semeion	**119.6**±4.54	143.1±5.39	185.6±4.25
spect_heart	**15.2**±0.75	16.9±1.76	22.2±1.25
statlog_heart	**20.5**±1.42	20.7±2.33	36.8±1.94
statlog_landsat	**175.7**±5.66	257.9±8.14	301.4±7.89
tic_tac_toe	35.2±2.53	**14.2**±5.06	64.6±6.44
wine	**5.5**±0.8	6.7±1.0	9.8±1.94
Average Rank	**1.2**	1.85	2.95

rank 2.425 (Table 4). The Friedman test returns a p-value of 4×10^{-3} and therefore, we reject the null-hypothesis that the methods perform equally. The Holm test, which is summarized in Table 5, confirms that both OC1 and DT are in fact less accurate than BiDT.

6.3 Comparison of tree size

We now present our findings on the tree sizes obtained by the three algorithms. The results for the unpruned trees are presented in Tables 6 and 8 summarizes the results for the pruned trees. Again, the best value for each dataset is highlighted in bold and the average rank is stated in the last line of the respective table.

For the unpruned trees we see that BiDT has the lowest rank of 1.2, OC1 has a rank of 1.85 and DT has the highest rank of 2.95 (Table 6) and this difference is confirmed to be significant by the Friedman test with a p-value of 2×10^{-7}. The post-hoc analysis which is summarized in Table 7 rejects both null-hypotheses which allows the conclusion that BiDT produces smaller trees than OC1 and DT.

For the pruned trees we get a different result. BiDT has a slightly higher rank of 1.625 than OC1 which has a rank of 1.55. DT has the highest rank with a value 2.825 (Table 8). The Friedman test confirms that there is a significant difference between at least two methods with a p-value of 3×10^{-5}. The post-hoc Holm test, summarized in Table 9, only confirms that the difference between BiDT and DT is significant while it does not provide evidence for a significant difference between BiDT and OC1.

6.4 Comparison of induction times

Lastly, we compare the induction times of the methods. We include the multi-threaded variant of BiDT, which is abbreviated by BiDTMT, to check whether a noticable

Table 7 Holm post-hoc test for size of unpruned trees

Algorithm	Rank	Holm adjusted p-value	Hypothesis
OC1	1.85	0.04	Rejected
DT	2.95	6×10^{-8}	Rejected
Control method: BiDT, Rank: 1.2			

Table 8 Comparison of tree size between pruned BiDT, OC1 and DT

Dataset	Leaves		
	BiDT	OC1	DT
breast_cancer_wisconsin	**3.0**±1.79	3.4±2.37	11.5±6.17
chess_krvskp	19.2±2.84	**4.6**±3.88	27.4±5.39
congressional_voting	**2.0**±0.0	2.1±0.3	**2.0**±0.0
dermatology	**5.7**±0.49	6.0±0.77	8.1±1.64
ecoli	14.0±7.07	**11.4**±5.37	17.8±6.48
glass	11.9±3.94	**11.7**±8.23	13.0±6.08
ionosphere	**4.2**±1.73	5.0±2.57	7.5±5.82
monks_problems_1	5.5±0.67	**3.5**±1.02	8.7±4.17
optical_recognition	**34.0**±13.25	43.5±20.71	49.9±25.35
parkinsons	**2.6**±1.2	3.1±1.92	3.7±1.85
pen_based_recognition	85.0±11.8	**65.6**±16.47	149.9±33.02
pima_indian_diabetes	10.3±4.78	**10.1**±13.97	24.6±15.94
poker_hand	504.5±62.47	**129.79**±52.37	775.64±123.74
seeds	3.4±0.49	**3.1**±0.3	4.5±1.91
semeion	**61.7**±18.54	90.0±34.39	96.7±35.94
spect_heart	3.5±1.22	**2.4**±0.92	3.4±2.8
statlog_heart	5.6±2.07	**3.2**±1.47	6.1±2.26
statlog_landsat	**46.5**±36.28	79.7±40.28	78.0±20.42
tic_tac_toe	14.3±5.49	**9.6**±9.6	38.6±13.55
wine	**3.6**±0.49	4.2±1.54	4.3±1.42
Average Rank	1.625	**1.55**	2.825

difference can be achieved through parallelization. As suggested in Section 5.2 parallelization is carried out at attribute level and we employ a total of 4 CPU cores. For the comparison of induction times, we don't include a comparison of the pruned trees as the post-pruning method is equal for all three methods and its runtime is neglectable in comparison to the time necessary to grow the trees. Thus, our findings would be equivalent to the results for the unpruned trees. Table 10 summarizes the induction times of our experiments.

Looking at the results, we see that DT has rank 1, BiDTMT has rank 2.2, OC1 has rank 3.3 and BiDT has rank 3.5. This result obviously reveals a significant difference as DT is the fastest algorithm for all datasets. Therefore, the Friedman test yields a p-value of 2×10^{-10}. The Holm test which is summarized in Table 11 confirms that DT is faster than BiDT. Moreover, the difference between BiDT and BiDTMT is found significant while there is no evidence for a significant difference between BiDT and OC1.

6.5 Discussion

Overall these results show the benefits of our proposed method. The inclusion of an additional attribute allows bivariate trees to capture the structure of the data better than univariate trees which results in a higher prediction accuracy for the majority of the datasets. Moreover, due to the global optimality of the splits, our bivariate decision tree algorithm manages to be more accurate than OC1 although the latter is not restricted to two attibutes per split. Pruning had no noticable effect on this result.

Without pruning BiDT manages to induce trees with a fewer amount of leaf nodes than DT and OC1. While it is not suprising that bivariate splits lead to smaller trees than

Table 9 Holm post-hoc test for size of pruned trees

Algorithm	Rank	Holm adjusted p-value	Hypothesis
OC1	1.55	0.81	Not Rejected
DT	2.825	3×10^{-4}	Rejected
Control method: BiDT, Rank: 1.625			

Table 10 Comparison of induction time between unpruned BiDT, BiDTMT, OC1 and DT

Dataset	Time (ms)			
	BiDT	BiDTMT	OC1	DT
breast_cancer_wisconsin	133.83±33.25	45.42±10.78	322.28±57.67	**8.35**±1.57
chess_krvskp	7777.9±479.84	2262.02±139.18	3596.21±753.99	**282.66**±15.67
congressional_voting	22.58±3.22	7.16±0.95	61.25±2.03	**1.68**±0.13
dermatology	72.67±7.89	28.86±3.15	161.31±4.86	**9.59**±0.84
ecoli	95.61±14.66	34.39±5.29	223.82±15.94	**2.16**±0.04
glass	112.83±25.22	38.79±10.64	93.36±4.72	**1.77**±0.26
ionosphere	2852.69±1199.71	1151.15±460.47	339.52±25.29	**9.08**±1.32
monks_problems_1	3.52±0.73	1.42±0.37	24.26±2.63	**0.48**±0.08
optical_recognition	101283.28±30949.24	27191.31±8376.51	21076.68±5826.74	**164.77**±10.49
parkinsons	176.25±76.02	61.08±32.91	135.29±21.72	**4.61**±1.11
pen_based_recognition	24054.65±1496.51	6657.05±496.56	19119.74±1558.83	**99.43**±9.38
pima_indian_diabetes	821.33±328.66	314.57±142.13	405.67±63.61	**9.58**±1.28
poker_hand	78540.86±5205.81	21846.84±1413.42	91050.92±4019.96	**555.97**±94.26
seeds	21.55±10.22	10.6±2.8	73.21±4.12	**5.84**±1.48
semeion	94820.07±3832.04	26528.1±1679.51	7369.95±697.19	**557.37**±62.21
spect_heart	37.25±4.76	16.8±2.34	43.22±1.1	**1.42**±0.26
statlog_heart	115.35±18.44	38.28±6.35	125.38±13.84	**2.58**±0.2
statlog_landsat	87472.7±25084.71	25936.01±7732.71	15853.02±934.55	**245.72**±24.91
tic_tac_toe	544.45±16.72	206.18±15.01	303.37±59.52	**55.68**±3.49
wine	33.77±18.11	13.32±8.45	41.38±5.0	**1.11**±0.07
Average Rank	3.5	2.2	3.3	**1.0**

univariate splits, the fact that BiDT performs better than OC1 is quite remarkable. As bivariate splits are a special case of oblique splits, theoretically, oblique decision trees are always capable of dividing the feature space with an equal number or fewer splits. The only advantage of BiDT is that the bivariate splits are provably global optimal while OC1's splits are only local optimal. From our experiments we therefore conclude that global optimal bivariate splits are often more efficient at dividing the feature space than local optimal oblique splits. This result further raises the assumption that global optimal splits are in fact desirable despite the greedy nature of the outer loop of the top-down induction algorithm.

For the pruned trees, the difference in tree size between BiDT and DT can again be explained by the fact that bivariate splits are more general than univariate splits and pruning the trees does not yield an advantage for DT. What is more interesting is the fact that no significant difference is observed between BiDT and OC1, although the difference was significant for the unpruned trees. We assume that this result can also be explained by the fact that BiDT's splits are global optimal while OC1's splits are only local optimal. The global optimal bivariate splits seem to be stronger and therefore less likely to be pruned while many local optimal oblique splits are less important to ensure the accuracy of the decision trees and are therefore pruned. Although after pruning our bivariate trees might not be smaller than OC1's oblique decision trees, they still have the advantage of being more interpretable.

The difference in induction time between BiDT and DT can be explained by the fact determining optimal univariate splits is much less complex than finding optimal

Table 11 Holm post-hoc test for build-up time for unpruned trees

Algorithm	Rank	Holm adjusted p-value	Hypothesis
BiDTMT	2.2	3×10^{-3}	Rejected
OC1	3.3	0.62	Not Rejected
DT	1.0	3×10^{-9}	Rejected
Control method: BiDT, Rank: 3.5			

bivariate splits or oblique splits in general. No significant difference is observed between BiDT's and OC1's induction time and by looking at the results we suspect that they are in the same order of magnitude for many applications. Interestingly, there is a significant difference between BiDT's and BiDTMT's induction time. On average BiDTMT is 2.9 times faster than BiDT which allows the conclusion that our induction algorithm does in fact profit from parallelization. Parallelization seems to be especially benefitial for larger datasets such as "optical_recognition", "pen_based_recognition" or "poker_hand". For these datasets the induction time is reduced by a factor greater than 3.5 which is about the best we can expect, considering the fact that four CPU cores were employed in our experiments. The smallest speed-up is achieved for the "seeds" dataset, yet the induction time is still reduced by a factor of two.

7 Illustration of interpretability

In this section, we provide an illustrative example to investigate the interpretability of bivariate decision trees. For this evaluation we test our induction algorithm on the "monks_problem_1" dataset [30]. It is an artifical dataset consisting of 124 samples and 6 attributes. Attributes X_1, X_2, X_4 take values in $\{1, 2, 3\}$, attributes X_3, X_6 in $\{1, 2\}$ and X_5 in $\{1, 2, 3, 4\}$. The labels are assigned according to the logical rule $(X_1 = X2) \vee (X_5 = 1)$. A label of 1 means that the rule is satisfied and 0 means that it is not. The major advantage of using such a artificially crafted dataset for this exploration is that we can check whether the decision tree actually captures the underlying structure.

The bivariate decision tree obtained by our algorithm is shown in Fig. 5. As we can see, the root node's rule is univariate and equivalent to $X_5 \geq 1.5$. As all values are integer and greater than zero, the negation of this rule corresponds to $X_5 = 1$ and therefore, Node 2 comprises the positive samples that satisfy the second part of the disjunction to be learned. What is worth mentioning here is the observation that the split is in fact univariate which means that no bivariate split could reduce the impurity after the initial univariate bound is computed in Step 2 of Algorithm 2. This is actually desirable as the inclusion of another variable could cause misinterpretations or misclassifications. This shows that it is a reasonable choice to use the best univariate split as an initial solution.

The subtree rooted at Node 1 is responsible for the condition $X_1 = X2$. We can rewrite the conditions at Node 1 and 3 to $X1 \geq 0.75X_2$ and $X_1 \leq X_2 + 0.5$ which are equivalent to $X1 \geq X_2$ and $X_1 \leq X_2$ for $X_1, X_2 \in \{1, 2, 3\}$. Thus, both conditions are satisfied if and only if $X_1 = X_2$ and we can conclude that Node 5 comprises all positive samples with $X_1 = X_2$ and Node 4 and 6 hold all negative samples with $X_1 \neq X_2$. Therefore, Nodes 2 and 5, the ones with the positive labels, are reached if and only if the underlying rule is satisfied. Thus, this tree captures the logic of the dataset perfectly. Furthermore, it is also an oblique tree of minimal size as at least three splits are necessary to express the underlying rule.

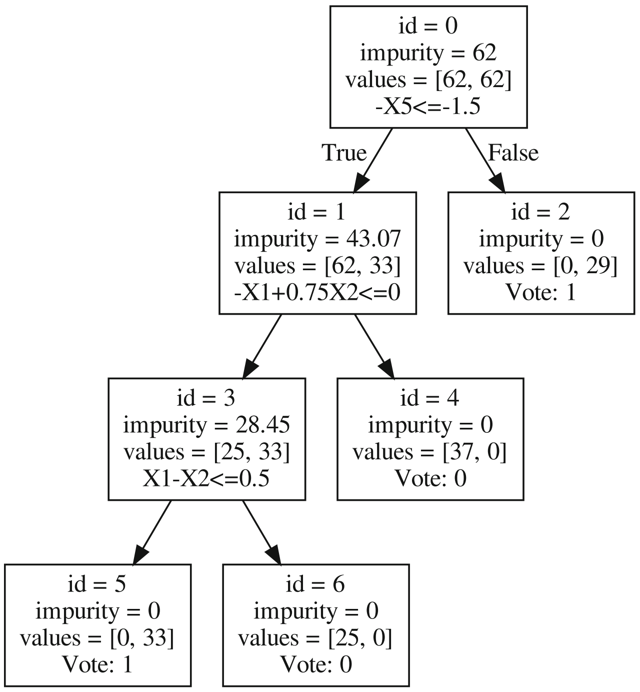


Fig. 5 Bivariate tree for the "monks_problem_1" dataset

Overall, this illustrative example shows how bivariate trees can capture the underlying structure of the data and how they can be interpreted with some basic algebraic reasoning.

8 Conclusion and future work

Interpretable machine learning becomes increasingly popular and decision trees play an import role in this context due to their easy to understand recursive nature. However, univariate decision trees are sometimes less accurate as other classification models. One reason for that is that they only use one feature per split such that complex interactions are not detected. This also often leads to huge trees which are also hard to interpret due to their size. Oblique decision trees can make up for this shortcoming at the cost of an increased induction time and the loss of interpretability when many features are involved in the splits. Moreover, the task of finding these oblique splits is far more complex and heuristics have to be applied to yield local optimal solutions with no global quality guarantee.

Bivariate decision trees based on our branch and bound algorithm overcome the issues of univariate decision trees while preserving the advantages of oblique trees. They are fairly interpretable due to the restriction to two attributes per split and due to the global optimality of the rules they are often more likely to capture the underlying structure of the data at hand. Their major disadvantage is the increased induction time compared to univariate trees. Our efficient branch and bound algorithm reduces this disadvantage effectively, especially due to the fact that it is easily parallizable and thus leverages the capabilities of modern hardware. For these reasons, they are a viable alternative to the commonly used univariate methods when extremely fast induction is less important and accurate yet interpretable trees are desired.

As a future work it would be interesting to test our proposed algorithm against different non-tree-based classification methods such as neural networks and support vector machines. Moreover, it would be interesting to investigate the advantages of bivariate decision trees in ensemble methods and to take advantage of their interpretability to analyze real world data.

Funding Open Access funding enabled and organized by Projekt DEAL.

References

1. Heath DG (1993) A geometric framework for machine learning. Ph.D. Thesis, Department of Computer Science, Johns Hopkins University
2. Schneider HJ, Friedrich N, Klotsche J, Pieper L, Nauck M, John U, Dorr M, Felix S, Lehnert H, Pittrow D et al (2010) The predictive value of different measures of obesity for incident cardiovascular events and mortality. J Clin Endocrinol Metab 95(4):1777–1785. https://doi.org/10.1210/jc.2009-1584
3. Arrieta AB, Díaz-Rodríguez N, Del Ser J, Bennetot A, Tabik S, Barbado A, García S, Gil-López S, Molina D, Benjamins R et al (2020) Explainable artificial intelligence (xai): Concepts, taxonomies, opportunities and challenges toward responsible ai. Inf Fusion 58:82–115. https://doi.org/10.1016/j.inffus.2019.12.012
4. Carvalho DV, Pereira EM, Cardoso JS (2019) Machine learning interpretability: A survey on methods and metrics. Electronics 8(8):832. https://doi.org/10.3390/electronics8080832
5. Guidotti R, Monreale A, Ruggieri S, Turini F, Giannotti F, Pedreschi D (2018) A survey of methods for explaining black box models. ACM Comput Surv 51(5):1–42. https://doi.org/10.1145/3236009
6. Blanco-Justicia A, Domingo-Ferrer J, Martínez S, Sánchez D (2020) Machine learning explainability via microaggregation and shallow decision trees. KNOWL-BASED SYST 194:105532. https://doi.org/10.1016/j.knosys.2020.105532
7. Rudin C (2019) Stop explaining black box machine learning models for high stakes decisions and use interpretable models instead. Nat Mach Intell 1(5):206–215. https://doi.org/10.1038/s42256-019-0048-x
8. Breiman L, Friedman JH, Olshen RA, Stone CJ (1984) Classification and regression trees. Wadsworth and Brooks, Monterey
9. Quinlan JR (1993) C4.5: programs for machine learning. Morgan Kaufmann Publishers, San Francisco
10. Hyafil L, Rivest RL (1976) Constructing optimal binary decision trees is np-complete. Inform Process Lett 5(1):15–17. https://doi.org/10.1016/0020-0190(76)90095-8
11. Heath D, Kasif S, Salzberg S (1993) Induction of oblique decision trees. In: Proceedings of the Thirteenth International Joint Conference on Artificial Intelligence. Morgan Kaufmann Publishers, pp 1002–1007
12. Murthy SK, Kasif S, Salzberg S (1994) A system for induction of oblique decision trees. J Artif Intell Res 2:1–32. https://doi.org/10.1613/jair.63
13. Cant´u-Paz E, Kamath C (2003) Inducing oblique decision trees with evolutionary algorithms. IEEE Trans Evol Comput 7(1):54–68. https://doi.org/10.1109/TEVC.2002.806857
14. Wickramarachchi DC, Robertson BL, Reale M, Price CJ, Brown J (2016) Hhcart: An oblique decision tree. Comput Stat Data Anal 96:12–23. https://doi.org/10.1016/j.csda.2015.11.006
15. López-Chau A, Cervantes J, López-García L, Lamont FG (2013) Fisher's decision tree. Expert Syst Appl 40(16):6283–6291. https://doi.org/10.1016/j.eswa.2013.05.044
16. Truong AKY (2009) Fast growing and interpretable oblique trees via logistic regression models. Ph.D. Thesis, Oxford
17. Bertsimas D, Dunn J (2017) Optimal classification trees. Mach Learn 106(7):1039–1082. https://doi.org/10.1007/s10994-017-5633-9
18. Verwer S, Zhang Y (2019) Learning optimal classification trees using a binary linear program formulation. In: Proceedings of the AAAI Conference on Artificial Intelligence, vol 33. AAAI Press, pp 1625–1632. https://doi.org/10.1609/aaai.v33i01.33011624
19. Blanquero R, Carrizosa E, Molero-Río C, Morales DR (2020) Sparsity in optimal randomized classification trees. Eur J Oper Res 284(1):255–272. https://doi.org/10.1016/j.ejor.2019.12.002
20. Lubinsky D (1994) Classification trees with bivariate splits. Appl Intell 4(3):283–296. https://doi.org/10.1007/BF00872094
21. Bioch JC, van der Meer O, Potharst R (1997) Bivariate decision trees. In: Komorowski J, Zytkow J (eds) European Symposium on Principles of Data Mining and Knowledge Discovery. Springer, Berlin, pp 232–242. https://doi.org/10.1007/3-540-63223-9_122
22. Coppersmith D, Hong SJ, Hosking JRM (1999) Partitioning nominal attributes in decision trees. Data Min Knowl Discov 3(2):197–217. https://doi.org/10.1023/A:1009869804967
23. Breiman L (1996) Some properties of splitting criteria. Mach Learn 24(1):41–47. https://doi.org/10.1023/A:1018094028462
24. Mingers J (1989) An empirical comparison of pruning methods for decision tree induction. Mach Learn 4(2):227–243. https://doi.org/10.1023/A:1022604100933
25. Dheeru D, Karra Taniskidou E (2017) UCI machine learning repository. http://archive.ics.uci.edu/ml, Accesed 2 October 2020
26. Demšar J (2006) Statistical comparisons of classifiers over multiple data sets. J Mach Learn Res 7:1–30

27. Friedman M (1937) The use of ranks to avoid the assumption of normality implicit in the analysis of variance. J Amer Statist Assoc 32(200):675–701. https://doi.org/10.1080/01621459.1937.10503522
28. Friedman M (1940) A comparison of alternative tests of significance for the problem of m rankings. Ann Math Stat 11(1):86–92. https://doi.org/10.1214/aoms/1177731944
29. Holm S (1979) A simple sequentially rejective multiple test procedure. Scand J Stat 6(2):65–70
30. Thrun SB, Bala J, Bloedorn E, Bratko I, Cestnik B, Cheng J, Jong KD, Dzeroski S, Fahlman SE, Fisher D, Hamann R, Kaufman K, Keller S, Kononenko I, Kreuziger J, Michalski RS, Mitchell T, Pachowicz P, Reich Y, Vafaie H, Welde WVD, Wenzel W, Wnek J, Zhang J (1991) The monk's problems a performance comparison of different learning algorithms. Technical Report, CMU-CS-91-197. Carnegie Mellon University, Pittsburgh