Learning Optimal Oblique Decision Trees with (Max)SAT

Florent Avellaneda
Université du Québec à Montréal (UQAM), Montréal, Canada
avellaneda.florent@uqam.ca

Abstract

Decision trees are widely used in machine learning for their interpretability and effectiveness in classification tasks. Traditional axis-parallel decision trees partition data using single-feature thresholds at each node, but they often struggle to represent complex, non-axis-aligned decision boundaries efficiently. This limitation can result in unnecessarily large and less interpretable trees. Oblique decision trees address this limitation by using linear combinations of features at each node, allowing a more natural representation of complex decision boundaries while maintaining interpretability through sparse linear combinations. However, learning optimal oblique decision trees poses a significant computational challenge, as existing methods predominantly rely on suboptimal greedy heuristics. In this paper, we propose a novel approach to learning globally optimal oblique decision trees by reformulating the problem as a (Max)SAT instance. By leveraging state-of-the-art (Max)SAT solvers, our method efficiently explores the solution space to identify optimal trees. Experiments on benchmark datasets demonstrate that our approach generates optimal oblique decision trees within reasonable computational time for small to medium-sized datasets.

1 Introduction

Decision trees are one of the most widely used models in machine learning, valued for their interpretability, simplicity, and competitive performance on various classification tasks [Breiman *et al.*, 1984; Quinlan, 1986]. A decision tree partitions the input space into regions by iteratively applying simple rules. Each leaf corresponds to a class, and the path from the root to a leaf represents a conjunction of conditions that leads to that classification.

Traditional decision tree learning methods, such as ID3 [Quinlan, 1986], C4.5 [Quinlan, 1993], and CART [Breiman *et al.*, 1984], employ an axis-parallel splitting approach, where each decision node tests a single feature against a threshold, resulting in splits that are perpendicular to one of the feature axes. In these methods, each internal node splits the data based on a single feature and a threshold. While effective and computationally efficient, axis-parallel trees have inherent limitations. Specifically, when the decision boundary between classes is not aligned with the feature axes, these trees may require numerous splits to approximate the boundary, resulting in larger, less interpretable, and potentially less accurate models.

Oblique decision trees overcome this limitation by allowing internal nodes to split the data using a linear combination of features rather than relying on single-feature splits [Heath *et al.*, 1993]. This flexibility enables the trees to naturally capture non-axis-parallel decision boundaries, often leading to smaller, more accurate models. Early work on oblique decision trees, such as the OC1 algorithm [Murthy *et al.*, 1994], employed randomized searches to identify hyperplanes. While these approaches offered improvements over axis-parallel trees, they relied on greedy heuristics, lacking guarantees of global optimality.

Recent advances in optimal decision tree learning have highlighted the potential of combinatorial optimization techniques. Mixed-integer linear programming [Bertsimas and Dunn, 2017; Verwer and Zhang, 2019], SAT solving [Avellaneda, 2020a; Verwer and Zhang, 2019; Shati *et al.*, 2021; Narodytska *et al.*, 2018], and dynamic programming [Demirović *et al.*, 2022; Aglin *et al.*, 2020; Nijssen and Fromont, 2007] have been successfully applied to derive provably optimal axis-parallel trees. However, extending these methods to oblique decision trees presents unique challenges due to the continuous nature of hyperplane parameters and the combinatorial complexity of optimizing tree structures. Although some progress has been made using linear programming for this task [Zhu *et al.*, 2020; Bertsimas and Dunn, 2017], to the best of our knowledge, the potential of (Max)SAT encoding remains unexplored.

In this paper, we introduce a novel approach for learning optimal oblique decision trees using (Max)SAT solvers. Our method encodes the learning problem as a (Max)SAT instance, enabling modern solvers to find globally optimal solutions. This approach combines guaranteed optimality with practical efficiency for reasonably sized datasets.

The paper is structured as follows. Section 2 introduces background on oblique decision trees. Section 3 describes our encoding and solving approach. Section 4 reports experimental results. Finally, Section 5 concludes the paper.

2 Preliminaries

Let $E = \{e_0, \ldots, e_{n-1}\}$ be a set of n *examples*. Each example $e_i \in \mathbb{R}^m$ is a vector representing the values of m *features* $F = \{f_0, \ldots, f_{m-1}\}$. For an example $e \in E$ and a feature $f \in F$, we use the notation $e[f]$ to represent the value of feature f for the example e. The *label class* for each example e is given by the *true label function* $\gamma : E \to \mathcal{Y}$, where $\mathcal{Y} = \{0, \ldots, c-1\}$ is the set of possible class labels.

Definition 1 (Binary Tree Structure)**.** *A* binary tree structure $\mathcal{T}$ *is a tuple* $(\mathcal{N}_B, \mathcal{N}_L, \delta, p, l, r)$ *where:*

- $\mathcal{N}_B$ *and* $\mathcal{N}_L$ *are disjoint finite sets of* branching nodes *and* leaf nodes*, respectively.*
- $\delta \in \mathcal{N}_B \cup \mathcal{N}_L$ *is the* root node*.*
- $p : (\mathcal{N}_B \cup \mathcal{N}_L) \setminus \{\delta\} \to \mathcal{N}_B$ *is the* parent function*, mapping each non-root node to its parent.*
- $l : \mathcal{N}_B \to \mathcal{N}_B \cup \mathcal{N}_L$ *is the* left child function*.*
- $r : \mathcal{N}_B \to \mathcal{N}_B \cup \mathcal{N}_L$ *is the* right child function*.*

Additionally, the following conditions must hold:

- ***(Parent-Child Consistency)*** *For every branching node $n \in \mathcal{N}_B$, we have $p(l(n)) = n$ and $p(r(n)) = n$.*
- ***(Tree Structure)*** *Every node $n \in \mathcal{N}_B \cup \mathcal{N}_L$ is reachable from the root through a unique path. Formally, there exists a unique sequence of nodes $n_0, \ldots, n_k$ such that $n_0 = \delta$, $n_k = n$, and for all $i \in \{0, \ldots, k-1\}$, $n_{i+1} = l(n_i)$ or $n_{i+1} = r(n_i)$.*

We define the *depth* of a binary tree structure $\mathcal{T}$ as the length of the longest path from the root to a leaf node.

Definition 2 (Oblique Decision Tree)**.** *An* oblique decision tree $\mathcal{D}$ *is a tuple* $\mathcal{D} = (\mathcal{T}, \theta, \lambda)$*, where:*

- $\mathcal{T} = (\mathcal{N}_B, \mathcal{N}_L, \delta, p, l, r)$ *is a binary tree structure.*
- $\theta : \mathcal{N}_B \to (\mathbb{R}^m \times \mathbb{R})$ *is a function that assigns to each branching node $n \in \mathcal{N}_B$ a decision rule, represented by the tuple $\theta(n) = (w_n, t_n)$. Here, $w_n \in \mathbb{R}^m$ is a weight vector and $t_n \in \mathbb{R}$ is a threshold.*
- $\lambda : \mathcal{N}_L \to \mathcal{Y}$ *is a function that assigns to each leaf node $n \in \mathcal{N}_L$ a prediction $\lambda(n) \in \mathcal{Y}$.*

To classify an example e using an oblique decision tree $\mathcal{D}$, we denote the resulting classification as $\mathcal{D}(e)$. This classification is determined by the function $cl(\mathcal{D}, \delta, e)$, where δ represents the root node of the tree. The function cl is defined recursively as follows:

$$cl(\mathcal{D}, n, e) = \begin{cases} \lambda(n) & \text{if } n \in \mathcal{N}_L, \\ cl(\mathcal{D}, l(n), e) & \text{if } w_n \cdot e \geq t_n, \\ cl(\mathcal{D}, r(n), e) & \text{otherwise.} \end{cases}$$

The accuracy of an oblique decision tree $\mathcal{D}$ on a set of examples E with true labels γ is formally defined as:

$$Accuracy(\mathcal{D}, E, \gamma) = \frac{1}{|E|} \sum_{e \in E} \mathbb{I}(\mathcal{D}(e) = \gamma(e))$$

Definition 3 (Optimal Oblique Decision Tree)**.** *Given a maximum depth $d \in \mathbb{N}$, a set of examples E, and a true label function γ, an oblique decision tree $\mathcal{D}^*$ is optimal with respect to depth d if the depth of $\mathcal{D}^*$ is equal or less than d and for any other oblique decision tree $\mathcal{D}'$ such that depth$(\mathcal{D}') \leq p$, $Accuracy(\mathcal{D}', E, \gamma) \leq Accuracy(\mathcal{D}^*, E, \gamma)$*

In this paper, we focus on sparse oblique decision trees where each decision rule $\theta(n) = (w_n, t_n)$ is constrained such that the weight vector w_n contains at most two non-zero components.

3 Constraint-Based Learning of Decision Trees

Given a set of examples E and a true label function $\gamma : E \to \mathcal{Y}$, our objective is to learn an optimal oblique decision tree $\mathcal{D}^*$ that minimizes classification error while respecting a maximum depth constraint d. Building on the SAT encoding for non-binary features proposed in [Shati *et al.*, 2021], we introduce a novel encoding for learning optimal oblique decision trees.

To formalize the problem, we define the following set of Boolean variables:

- $S_{e,n}$: True if the example e is directed towards the left child, if it passes through branching node n.
- $Z_{e,n}$: True if the example e terminates at leaf node n.
- $C_{n,c}$: True if the leaf node n is assigned to the class c.
- A_{n,f_1,f_2}: True if the feature f_1 and f_2 are selected for the split at branching node n.

For each each $n \in \mathcal{N}_B$ and every $f_1, f_2, f_1', f_2' \in F$ such that $(f_1, f_2) \neq (f_1', f_2')$, we add the clause:

$$\neg A_{n,f_1,f_2} \vee \neg A_{n,f_1',f_2'} \tag{1}$$

These clauses ensure that each branching node selects at most one pair of features for splitting.

For each $n \in \mathcal{N}_B$, we add the clause:

$$\bigvee_{f_1, f_2 \in F} A_{n,f_1,f_2} \tag{2}$$

These clauses ensure that each branching node selects at least one pair of features for splitting.

For each $n \in \mathcal{N}_L$, each $e \in E$ and each $n' \in A_l(n)$, we add the clause:

$$\neg Z_{e,n} \vee S_{e,n'} \tag{3}$$

With $A_l(n)$ the set of ancestors of the leaf node n such that n is a descendant of their left branch.

For each $n \in \mathcal{N}_L$, each $e \in E$ and each $n' \in A_r(n)$, we add the clause:

$$\neg Z_{e,n} \vee \neg S_{e,n'} \tag{4}$$

With $A_r(n)$ the set of ancestors of the leaf node n such that n is a descendant of their right branch.

For each $n \in \mathcal{N}_L$ and each $e \in E$, we add the clause:

$$Z_{e,n} \vee \bigvee_{n' \in A_l(n)} \neg S_{e,n'} \vee \bigvee_{n' \in A_r(n)} S_{e,n'} \tag{5}$$

The clauses (3) and (4) ensure that if an example e reaches a leaf node n, it must have followed the correct path from the root to n, as dictated by the variables S. Conversely, the clauses (5) ensure that if an example e does not terminate at a leaf node n, then at least one of its ancestor nodes must have redirected e to a different path.

For every $c_1, c_2 \in \mathcal{Y}$ and each $n \in \mathcal{N}_L$, we add the clause:

$$\neg C_{n,c_1} \vee \neg C_{n,c_2} \tag{6}$$

These clauses ensure that each leaf node is assigned to a single class.

For each example $e \in E$ and each $n \in \mathcal{N}_L$, we add the clause:

$$\neg Z_{e,n} \vee C_{n,\gamma(e)} \tag{7}$$

These clauses ensure that if an example e reaches a leaf node n, the leaf node n must be assigned the correct class as specified by the true label function γ.

Note that if clauses (7) are treated as hard clauses, then we construct a SAT formula that seeks a decision tree fully consistent with the dataset, i.e., achieving 100% accuracy. If such a tree does not exist and we aim to find the tree that minimizes classification errors, we can switch to MaxSAT by treating clauses (7) as soft clauses. The same applies to the SMT setting: we can use MaxSMT to minimize classification errors when no perfectly consistent tree exists.

At this point, we need to add constraints to S and A to ensure that, given the features present at each branching node, the partition represented by S can be achieved via linear separation using only two features. We propose two solutions to implement these constraints. The first approach utilizes an (Max)SMT solver to encode them directly, while the second involves encoding them natively in (Max)SAT.

3.1 Linear separation with (Max)SMT

Satisfiability Modulo Theories (SMT) is an extension of SAT that allows for solving satisfiability problems with constraints from specific theories, such as linear arithmetic. By employing an SMT solver, we can encode the linear separation constraints directly within the theory of linear arithmetic, which simplifies the process and leverages the solver's built-in capabilities for handling such theories. Its extension, MaxSMT, further allows for the inclusion of soft clauses.

To encode the possible splits at each branching node, we introduce real variables $x_{n,1}, x_{n,2}$, and $x_{n,3}$ for each branching node $n \in \mathcal{N}_B$. These variables represent the coefficients of the linear inequality that defines the split at node n.

Specifically, if the variable A_{n,f_1,f_2} is true, indicating that features f_1 and f_2 are used at node n, then the split at node n is defined by the linear inequality:

$$x_{n,1}F_{f_1} + x_{n,2}F_{f_2} + x_{n,3} \geq 0.$$

Here, F_{f_1} and F_{f_2} represent the values of features f_1 and f_2, respectively. This linear equation determines how data points are partitioned at the branching node based on the selected features and the learned coefficients.

Thus, for each branching node $n \in \mathcal{N}_B$, each pair of features $f_1, f_2 \in F$, and each example $e \in E$, we add the following constraints:

$$\neg A_{n,f_1,f_2} \vee \neg S_{e,n} \vee (x_{n,1}e[f_1] + x_{n,2}e[f_2] + x_{n,3} \geq 0) \tag{8}$$

$$\neg A_{n,f_1,f_2} \vee S_{e,n} \vee (x_{n,1}e[f_1] + x_{n,2}e[f_2] + x_{n,3} < 0) \tag{9}$$

These constraints ensure that if the pair of features f_1 and f_2 is selected for the split at branching node n, then example e is directed to the left child if the linear inequality is satisfied, and to the right child otherwise.

3.2 Linear Separation with (Max)SAT

At each branching node n, if A_{n,f_1,f_2} is true, the split at this node, represented by $S_{e,n}$, must correspond to a linear separation of the examples based on features f_1 and f_2. This reduces to finding a linear separator in a 2D plane, where each example is a point with coordinates given by its values of f_1 and f_2. A naive approach would consider all possible linear cuts and add constraints to ensure each example is directed to the correct child node based on the chosen cut. However, this method is inefficient due to the excessive number of variables and constraints required.

To address this, we propose an alternative approach that reduces the number of variables and constraints needed to encode the problem. We introduce two fictitious points at infinity in the projective plane, defined as $P_A = (1, 1, 0)$ and $P_B = (-1, -1, 0)$. In the projective plane, a point is represented by a triplet (x, y, z). When $z \neq 0$, the point corresponds to the Euclidean coordinates $(x/z, y/z)$. When $z = 0$, the point is considered to lie "at infinity" in a specific direction. Given three points p_1, p_2 and p_3, we denote by triangle(p_1, p_2, p_3) the convex hull of these three points.

Then, our method relies on the following theorem:

Theorem 1. *Let A and B be two sets of points. These sets are linearly separable if and only if the following two conditions are satisfied:*

- *For every triplet of points (p_1, p_2, p_3) with $p_1, p_2 \in A$ and $p_3 \in B$, $p_3 \notin$ triangle(p_1, p_2, P_A).*
- *For every triplet of points (p_1, p_2, p_3) with $p_1, p_2 \in B$ and $p_3 \in A$, $p_3 \notin$ triangle(p_1, p_2, P_B).*

Where P_A and P_B are two points at infinity in the projective plane, defined as $P_A = (1, 1, 0)$ and $P_B = (-1, -1, 0)$, or vice versa.

Proof. ($\Rightarrow$) Suppose that A and B are linearly separable. Then, there exist real coefficients a, b, and c such that:

$$\forall (x, y) \in A, ax+by+c \geq 0 \quad \text{and} \quad \forall (x, y) \in B, ax+by+c < 0.$$

Since P_A and P_B are points at infinity in opposite directions, only one of them lies in the half-plane $ax + by + c \geq 0$. Without loss of generality, assume P_A lies in this half-plane. Consider any triplet (p_1, p_2, p_3) with $p_1, p_2 \in A$ and $p_3 \in B$. The triangle formed by p_1, p_2, and P_A, is entirely contained within the half-plane $ax + by + c \geq 0$. Since p_3 lies in B and the half-plane $ax + by + c < 0$, it cannot lie within $triangle(p_1, p_2, P_A)$. Therefore, $p_3 \notin triangle(p_1, p_2, P_A)$. A similar argument applies for B and P_B.

($\Leftarrow$) Conversely, suppose the conditions hold but A and B are not linearly separable. This means their convex hulls intersect. Consequently, there exists a point p that belongs to both Conv(A) and Conv(B).
This implies that there exist points $p_1, p_2 \in A$ and $p_3, p_4 \in B$ such that p lies within both $triangle(p_1, p_2, P_A)$ and $triangle(p_3, p_4, P_B)$. Since the sides leading to P_A and P_B are parallel (as P_A and P_B are points at infinity in opposite directions), and the triangles have a non-empty intersection, this intersection must contain at least one of the points p_1, p_2, p_3, or p_4. This contradiction implies that A and B must be linearly separable. □

This theorem allows us to encode the linear separation problem by stating that if two points p_1 and p_2 belong to the same class A, then any point p_3 that lies within the triangle formed by p_1, p_2, and the point at infinity P_A must also belong to class A. A similar reasoning applies for class B with the point at infinity P_B.
Therefore, for each branching node $n \in \mathcal{N}_B$, each pair of features $f_1, f_2 \in F$, and every triplet of examples $e_1, e_2, e_3 \in E$ such that p_3 lies inside the triangle formed by p_1, p_2, and P_A, with $p_1 = (e_1[f_1], e_1[f_2]), p_2 = (e_2[f_1], e_2[f_2])$, and $p_3 = (e_3[f_1], e_3[f_2])$, we add the clause:

$$\neg A_{n,f_1,f_2} \vee \neg S_{e_1,n} \vee \neg S_{e_2,n} \vee S_{e_3,n}. \tag{10}$$

Similarly, each branching node $n \in \mathcal{N}_B$, every pair of features $f_1, f_2 \in F$, and every triplet of examples $e_1, e_2, e_3 \in E$ such that $p_3 \in$ triangle(p_1, p_2, P_B), with $p_1 = (e_1[f_1], e_1[f_2]), p_2 = (e_2[f_1], e_2[f_2])$, and $p_3 = (e_3[f_1], e_3[f_2])$, we add the clause:

$$\neg A_{n,f_1,f_2} \vee S_{e_1,n} \vee S_{e_2,n} \vee \neg S_{e_3,n}. \tag{11}$$

Equations (10) and (11) ensure that if the pair of features f_1, f_2 is chosen for separation at branching node n, S will split the examples such that a linear separation exists.

Minimizing the Number of Clauses

Equations (10) and (11) can introduce a substantial number of clauses into our model, potentially increasing computational complexity. To enhance efficiency, we aim to minimize the number of clauses by eliminating redundant constraints.
Let $p_1 = (e_1[f_1], e_1[f_2])$, $p_2 = (e_2[f_1], e_2[f_2])$, $p_3 = (e_3[f_1], e_3[f_2])$, and $p'_3 = (e'_3[f_1], e'_3[f_2])$, four points such that $p_3 \in$ triangle(p_1, p_2, P_A), $p'_3 \in$ triangle(p_1, p_2, P_A), and $p'_3 \in$ triangle(p_1, p_3, P_A) (see Figure 1). In this configuration, if we have already added the constraints (10) for the triplets (e_1, e_2, e_3) and (e_1, e_3, e'_3), it becomes unnecessary to add the same constraints for the triplet (e_1, e_2, e'_3). The reason is that the constraints for (e_1, e_2, e'_3) are implied by those for (e_1, e_2, e_3) and (e_1, e_3, e'_3). A similar argument applies when $p'_3 \in$ triangle(p_2, p_3, P_A), as well as for the corresponding case involving the point P_B.
Similarly, examples sharing identical values for features f_1 and f_2 cannot be separated by a linear decision boundary based on these features. Thus, it suffices to retain a single representative example from each group of duplicates for our constraints.

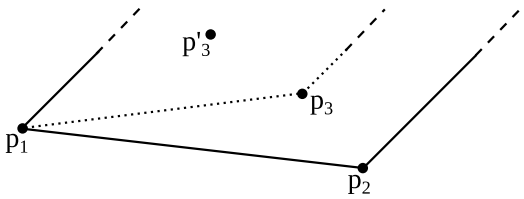

Figure 1: Example for constraints simplification.

To formalize this reduction, we define two functions based on a total order $\leq$ on the set of examples E. The function *firsts*(f_1, f_2) selects, for each unique combination of feature values (v_1, v_2), the minimal example according to the total order. Formally, $firsts(f_1, f_2) = \{e \in E \mid \forall e' \in E, e'[f_1] = e[f_1] \wedge e'[f_2] = e[f_2] \Rightarrow e' \geq e\}$ For each $e \in$ *firsts*(f_1, f_2), the function *doublons*(e) returns the set of examples that share the same feature values as e but are greater than e in the total order. Formally: *doublons*$(e) = \{e' \in E \mid e'[f_1] = e[f_1],\ e'[f_2] = e[f_2],\ e' > e\}$.
Using these functions, we can now replace (10) and (11) by the following constraints.
For each branching node $n \in \mathcal{N}_B$, each pair of features $f_1, f_2 \in F$, and each example $e \in firsts(f_1, f_2)$, we add the clauses:

$$\bigwedge_{e' \in doublons(e)} \neg A_{n,f_1,f_2} \vee \neg S_{e,n} \vee S_{e',n} \tag{12}$$

$$\bigwedge_{e' \in doublons(e)} \neg A_{n,f_1,f_2} \vee S_{e,n} \vee \neg S_{e',n} \tag{13}$$

These constraints ensure that examples with identical values for features f_1 and f_2 are directed toward the same child.
We define the projection $\pi_{f_1,f_2}(e) = (e[f_1], e[f_2])$. For each branching node $n \in \mathcal{N}_B$, each pair of features $f_1, f_2 \in F$, and each pair of examples $e_1, e_2 \in$ firsts(f_1, f_2), we select only one example $e_3 \in$ firsts(f_1, f_2) such that $\pi_{f_1,f_2}(e_3) \in$ triangle$(\pi_{f_1,f_2}(e_1), \pi_{f_1,f_2}(e_2), P_A)$, and for all $e'_3 \in$ firsts(f_1, f_2), if $\pi_{f_1,f_2}(e'_3) \in$ triangle$(\pi_{f_1,f_2}(e_1), \pi_{f_1,f_2}(e_2), P_A)$, then $\pi_{f_1,f_2}(e'_3) \in$ triangle$(\pi_{f_1,f_2}(e_1), \pi_{f_1,f_2}(e_3), P_A) \cup$ triangle$(\pi_{f_1,f_2}(e_2), \pi_{f_1,f_2}(e_3), P_A)$.
We then add the clause:

$$\neg A_{n,f_1,f_2} \vee \neg S_{e_1,n} \vee \neg S_{e_2,n} \vee S_{e_3,n} \tag{14}$$

For each branching node $n \in \mathcal{N}_B$, each pair of features $f_1, f_2 \in F$, and each pair of examples $e_1, e_2 \in$ firsts(f_1, f_2), we select only one example $e_3 \in$ firsts(f_1, f_2) such that $\pi_{f_1,f_2}(e_3) \in$ triangle$(\pi_{f_1,f_2}(e_1), \pi_{f_1,f_2}(e_2), P_B)$, and for all $e'_3 \in$ firsts(f_1, f_2), if $\pi_{f_1,f_2}(e'_3) \in$ triangle$(\pi_{f_1,f_2}(e_1), \pi_{f_1,f_2}(e_2), P_B)$, then $\pi_{f_1,f_2}(e'_3) \in$ triangle$(\pi_{f_1,f_2}(e_1), \pi_{f_1,f_2}(e_3), P_B) \cup$ triangle$(\pi_{f_1,f_2}(e_2), \pi_{f_1,f_2}(e_3), P_B)$.
We then add the clause:

$$\neg A_{n,f_1,f_2} \vee S_{e_1,n} \vee S_{e_2,n} \vee \neg S_{e_3,n} \tag{15}$$

The clauses (14) and (15) ensure that if the pair of features f_1, f_2 is chosen for splitting at node n, then the assignment of examples to the left or right child by S corresponds to a linear separation based on these features.

Datasets	# Exemples	# Features	# classes
Balance-scale	625	4	3
BreastCancer	116	9	3
Car	1727	6	2
Cryptotherapy	90	6	2
Heart_60+	309	13	2
Heart	1025	13	2
Immunotherapy	90	7	2
Iris	150	4	3
Mouse	70	5	2
Person ... Index	500	3	6
SIRTUIN6	100	6	2
Wine	178	13	3
Zoo	101	16	7

Table 1: Characteristics of real-world datasets used.

3.3 Decoding Decision Trees from Solutions

The solutions obtained from the (Max)SMT or (Max)SAT encodings allow us to construct an oblique decision tree that minimizes classification errors. However, translating these solutions into an explicit tree is nontrivial. While it is relatively easy to determine which examples reach each node, identifying a linear separation equations at the branching nodes presents a challenge.

In the SAT-based approach, the linear separation equations at the branching nodes are not explicitly provided. The SMT-based approach produces valid equations but does not necessarily optimize the separation between classes.

To address this, we employ a hard-margin Support Vector Machine (SVM) to compute optimal separating hyperplanes at each branching node [Vapnik, 2013]. Hard-margin SVMs find the hyperplane that maximizes the margin between classes, reducing the risk of misclassification on unseen data.

Reconstructing the oblique decision tree involves two steps.

Step 1. Identifying Examples at Each Node

The variables $S_{e,n}$ from the solution indicate the path each example e takes through the tree. For a node n, we determine the examples that reach it by examining the values of $S_{e,n}$. An example reaches node n if it satisfies the branching decisions along the path from the root to n. This step is straightforward and involves tracing the decisions encoded by $S_{e,n}$ for each example.

Step 2. Computing Optimal Separating Hyperplanes

For each branching node n, we compute the optimal separating hyperplane using a hard-margin SVM. Examples reaching node n are labeled $+1$ if directed to the left child, and -1 if directed to the right. Then, using these labeled examples and the selected features F_{f_1} and F_{f_2} at node n, we compute the optimal separating hyperplane defined by:

$$x_{n,1}F_{f_1} + x_{n,2}F_{f_2} + x_{n,3} = 0,$$

where $x_{n,1}, x_{n,2}$, and $x_{n,3}$ are SVM-derived coefficients. This hyperplane represents the decision boundary at node n.

4 Experimentation

Our experimental evaluation assesses the computational feasibility and effectiveness of our proposed (Max)SAT and (Max)SMT encodings for learning optimal oblique decision trees. While oblique decision trees are known for their superior accuracy and more compact model sizes compared to axis-parallel trees, particularly on datasets with non-linear decision boundaries [Bertsimas and Dunn, 2017], our research specifically addresses the challenge of finding provably optimal solutions. We focused on two key aspects: the computational complexity of achieving optimality and the structural characteristics of the resulting optimal trees. We implemented our methods in C++[1] and utilized the EvalMaxSAT solver [Avellaneda, 2020b] for MaxSAT, the Z3 solver [De Moura and Bjørner, 2008] for SMT and LibSVM [Chang and Lin, 2011] to compute the hard-margin SVM.

4.1 Experimental Setup

All experiments were conducted under controlled conditions using a single thread of a AMD EPYC 7532 (Zen 2) processor (2.4 GHz) with 32 GB of RAM. To ensure practical feasibility, we imposed a maximum runtime limit of three hours per experiment. In our results tables, experiments exceeding this time limit are denoted by "–".

4.2 Scope and Limitations

Our evaluation deliberately focuses on the fundamental challenge of learning optimal oblique decision trees rather than on techniques for enhancing generalization performance. Consequently, we did not perform exhaustive comparisons of test set accuracy against heuristic methods. Advanced techniques such as pruning, regularization of oblique splits, and ensemble methods were beyond the scope of our research.

4.3 Real-World Datasets

We evaluated our approach using 13 small to medium-sized real-world datasets from the UCI Machine Learning Repository [Kelly *et al.*, 2023], as detailed in Table 1. These datasets represent diverse classification tasks, exhibiting variations in the number of instances, features, and classes.

To evaluate our optimal oblique tree learning methods, we compared them with three distinct approaches:

- *Optimal ADT* [Shati *et al.*, 2021]: The current state-of-the-art method for learning optimal axis-parallel decision trees using (Max)SAT solvers. This comparison allows us to isolate the computational overhead introduced by learning oblique splits while maintaining optimality.
- *HHCART* [Wickramarachchi *et al.*, 2016]: A heuristic algorithm for learning oblique decision trees, implemented in the `scikit-obliquetree` Python package [ECNU, 2021]. This serves as a reference against a practical and fast heuristic approach.
- *IAI* [Interpretable AI, 2024]: A commercial implementation based on the optimal decision tree framework [Bertsimas and Dunn, 2017]. While IAI's theoretical

[1] https://github.com/FlorentAvellaneda/InferOptimalObliqueDT

framework addresses optimal trees, their implementation employs heuristic approximations for practical runtime considerations. We acknowledge IAI's provision of an academic license for our experimental work.

Although we originally planned to include comparisons with the mixed-integer programming approach for optimal oblique decision tree learning [Zhu *et al.*, 2020], we were unable to obtain access to their implementation.

Our experimental results, presented in Table 2, demonstrate that both our SMT-based and MaxSAT-based encodings consistently achieve superior training accuracy, validating our theoretical guarantee of discovering globally optimal oblique decision trees. Notably, the MaxSAT encodings solved with EvalMaxSAT demonstrated markedly better runtime performance compared to the SMT encodings solved with Z3.

To address scalability limitations with larger datasets, we conducted additional experiments using the incomplete MaxSAT solver NuWLS [Chu *et al.*, 2023], configuring it with timeouts of 5 minutes and 1 hour. Although these configurations do not guarantee optimality, NuWLS provided high-quality solutions within the specified time limits. The results closely approximated those of the complete methods and, crucially, NuWLS successfully processed larger datasets where exact methods exceeded the 3-hour timeout threshold. In these cases, NuWLS solutions frequently outperformed heuristic approaches in terms of training accuracy.

Our comparative analysis revealed that our prototype consistently produced decision trees with superior training accuracy compared to both heuristic methods and the optimal axis-parallel method. However, HHCART and IAI, while significantly faster (typically completing in under one second) traded off solution quality for speed, as evidenced by their lower training accuracies.

Notably, IAI's implementation includes non-optional heuristic optimizations for runtime performance, which prevents direct measurement of the time required to find provably optimal decision trees. Moreover, in their original research [Bertsimas and Dunn, 2017], which focused primarily on demonstrating superior out-of-sample accuracy rather than computational efficiency, they reported that finding an optimal oblique decision tree of depth 2 for the Wine dataset required approximately 5 minutes. By contrast, our method achieves the same result in 40 seconds.

4.4 Synthetic Datasets

To complement our real-world dataset analysis with controlled evaluations, we conducted systematic experiments using synthetic datasets generated through a structured protocol. This approach allowed us to precisely measure the performance of our encodings under specific, controlled conditions.

In our first experimental series, illustrated in Figure 2, we examined the relationship between dataset size and computational efficiency. We generated synthetic datasets using random oblique decision trees with consistent parameters: four features, each with 100 possible values, two classes, and a tree depth of three. The dataset sizes varied to observe how computational time scales with the number of examples. The results demonstrated a polynomial growth in execution time as the dataset size increased. Notably, the SAT encoding consistently outperformed the SMT encoding by several orders of magnitude in terms of runtime efficiency.

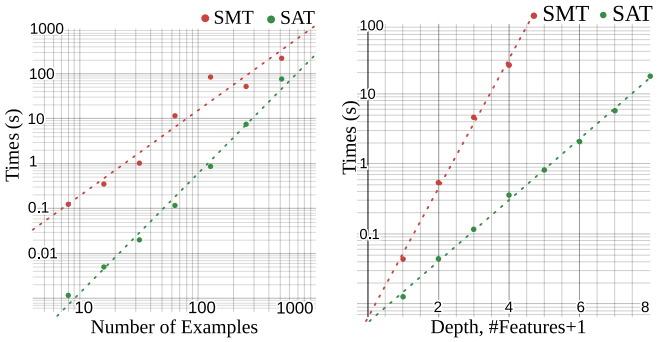

Figure 2: Time to find an optimal oblique decision tree using SMT and SAT encodings on synthetic datasets (average over 10 runs): (Left) as a function of the number of training examples; (Right) as a function of the depth (number of features + 1).

In the second experimental series, presented in Figure 2, we investigated the impact of tree depth on computational complexity. We maintained a constant training dataset size of 64 examples and systematically increased the tree depth from low to higher values. Correspondingly, we adjusted the number of features to be equal to the tree depth plus one (i.e., number of features $=$ depth $+ 1$) for each iteration. The experiments revealed an exponential growth in execution time for both encodings as tree depth increased. However, the SAT encoding exhibited a significantly slower growth rate compared to the SMT encoding, indicating better scalability with respect to tree depth.

5 Conclusion

In this paper, we presented a novel approach for learning optimal oblique decision trees by formulating the problem as instances of (Max)SAT and SMT. Our novel contributions are a theoretical framework encoding the oblique decision tree learning problem as (Max)SAT or SMT, an efficient SAT encoding that uses geometric properties of linear separability, and empirical validation showing our method finds provably optimal oblique decision trees on small to medium-sized datasets. Notably, our SAT encoding significantly outperforms the SMT encoding in computational efficiency, often by several orders of magnitude.

Despite these advancements, computational complexity remains a significant challenge, especially for larger datasets and deeper trees. While our SAT-based approach scales better than SMT, both exhibit exponential growth in execution time with increasing tree depth due to the combinatorial nature of finding globally optimal decision trees. To mitigate scalability issues, we used incomplete MaxSAT solvers, which, although not guaranteeing optimality, provided high-quality solutions within practical time bounds for larger datasets. Our experiments show that incomplete MaxSAT solvers often outperform heuristic methods in training accuracy while operating within reasonable computational times.

Depth	HHCART	IAI	Optimal ADT	Our SMT	Our SAT		
			EvalMaxSAT (3h)	Z3 (3h)	NuWLS (5min)	NuWLS (1h)	EvalMaxSAT (3h)
Person Gender Height Weight Index							
d=1	46.2%	64.2%	51.6% ($\leq$ 1s)	**64.8%** (59s)	**64.8%**	**64.8%**	**64.8% (15s)**
d=2	58.8%	84.2%	61.0% (540s)	**90.6%** (5800s)	**90.6%**	**90.6%**	**90.6% (98s)**
d=3	67.2%	90.0%	–	–	**98.0%**	**98.0%**	**98.0%** (350s)
d=4	77.6%	96.4%	–	–	**99.6%**	**99.6%**	**99.6%** (340s)
Balance-scale							
d=1	63.5%	70.0%	63.5% (0.64s)	**70.1%** (38s)	**70.1%**	**70.1%**	**70.1% (2.8s)**
d=2	63.5%	**82.9%**	71.7% (56s)	–	**82.9%**	**82.9%**	**82.9%** (2800s)
d=3	75.2%	87.2%	–	–	**89.3%**	**89.3%**	–
BreastCancer							
d=1	67.2%	71.6%	71.6% (0.11s)	–	**79.3%**	**79.3%**	**79.3% (22s)**
d=2	73.3%	87.8%	82.8% (12s)	–	**90.5%**	**90.5%**	–
d=3	75.0%	79.3%	91.4% (260s)	–	**98.3%**	**98.3%**	–
Car							
d=1	71.3%	70.0%	71.3% (2.5s)	**84.1%** (100s)	**84.1%**	**84.1%**	**84.1% (45s)**
d=2	82.4%	89.3%	85.5% (160s)	–	90.4%	**90.7%**	–
d=3	85.5%	95.8%	89.5% (7900s)	–	96.4%	**96.9%**	–
Cryptotherapy							
d=1	82.2%	**91.1%**	85.6% (0.015s)	**91.1%** (7.2s)	**91.1%**	**91.1%**	**91.1%** (0.22s)
d=2	82.2%	96.7%	94.4% (0.20s)	**97.8%** (1800s)	**97.8%**	**97.8%**	**97.8% (2.1s)**
d=3	86.6%	97.8%	98.9% (0.51s)	**100%** (15s)	**100%**	**100%**	**100% (1.3s)**
Heart_60+							
d=1	75.4%	80.6%	75.4% (0.43s)	**80.9%** (3000s)	**80.9%**	**80.9%**	**80.9% (63s)**
d=2	79.0%	87.1%	81.9% (110s)	–	87.1%	**90.0%**	–
d=3	82.3%	92.6%	90.6% (2200s)	–	99.0%	99.0%	**100% (290s)**
Heart							
d=1	76.0%	**78.4%**	76.0% (8.0s)	–	**78.4%**	**78.4%**	**78.4%** (3100s)
d=2	76.0%	**85.2%**	79.6% (4500s)	–	**85.2%**	**85.2%**	–
d=3	82.0%	**88.7%**	–	–	85.6%	85.9%	–
Immunotherapy							
d=1	78.9%	**88.9%**	86.7% (0.025s)	**88.9%** (52s)	**88.9%**	88.9%	**88.9%** (0.64s)
d=2	78.9%	86.7%	91.1% (0.82s)	–	**95.6%**	**95.6%**	**95.6% (67s)**
d=3	86.7%	86.7%	95.6% (2.5s)	**100%** (2200s)	**100%**	**100%**	**100% (2.2s)**
Iris							
d=1	**66.7%**	**66.7%**	**66.7%** (0.039s)	**66.7%** (7.4s)	**66.7%**	**66.7%**	**66.7%** (0.53s)
d=2	94.7%	96.0%	96.0% (0.15s)	**98.7%** (190s)	**98.7%**	**98.7%**	**98.7% (1.5s)**
d=3	96.0%	98.0%	99.3% (0.17s)	**100%** (3.2s)	**100%**	**100%**	**100% (0.70s)**
Mouse							
d=1	94.3%	94.3%	94.3% (0.0045s)	**97.1%** (0.33s)	**97.1%**	**97.1%**	**97.1% (0.014s)**
d=2	94.3%	94.3%	97.1% (0.026s)	**98.6%** (6.6s)	**98.6%**	**98.6%**	**98.6% (0.11s)**
d=3	97.1%	94.3%	98.6% (0.031s)	**100%** (0.72s)	**100%**	**100%**	**100% (0.047s)**
SIRTUIN6							
d=1	79.0%	86.0%	86.0% (0.039s)	**89.0%** (1700s)	**89.0%**	**89.0%**	**89.0% (1.3s)**
d=2	79.0%	90.0%	92.0% (0.89s)	–	**95.0%**	**95.0%**	**95.0% (130s)**
d=3	86.0%	90.0%	96.0% (5.0s)	–	**100%**	**100%**	**100% (14s)**
Wine							
d=1	59.0%	**71.3%**	69.7% (0.22s)	–	**71.3%**	**71.3%**	**71.3%** (660s)
d=2	83.7%	96.6%	96.6% (1.1s)	–	**99.4%**	**99.4%**	**99.4% (41s)**
d=3	83.7%	98.3%	**100% (0.42s)**	–	**100%**	**100%**	**100%** (39s)
Zoo							
d=1	45.5%	**60.4%**	**60.4%** (0.067s)	**60.4%** (1.2s)	**60.4%**	**60.4%**	**60.4%** (0.094s)
d=2	73.3%	82.2%	82.2% (0.36s)	83.2% (91s)	**83.2%**	**83.2%**	**83.2% (0.50s)**
d=3	85.1%	92.1%	94.1% (1.1s)	**100%** (3.1s)	**100%**	**100%**	**100% (0.49s)**

Table 2: Evaluating training accuracy and construction time of decision trees on real-world datasets.

Acknowledgements

We gratefully acknowledge the support of the Natural Sciences and Engineering Research Council of Canada (NSERC) [funding reference number RGPIN-2023-04468] and Fonds de recherche du Québec - Nature et technologies (FRQNT) [funding reference number 345904 : https://doi.org/10.69777/345904]. The benchmarks were carried out with support from Calcul Québec (calculquebec.ca) and the Digital Research Alliance of Canada (alliancecan.ca).

References

[Aglin *et al.*, 2020] Gaël Aglin, Siegfried Nijssen, and Pierre Schaus. Learning optimal decision trees using caching branch-and-bound search. In *Proceedings of the AAAI conference on artificial intelligence*, volume 34, pages 3146–3153, 2020.

[Avellaneda, 2020a] Florent Avellaneda. Efficient inference of optimal decision trees. In *Proceedings of the AAAI conference on artificial intelligence*, volume 34, pages 3195–3202, 2020.

[Avellaneda, 2020b] Florent Avellaneda. A short description of the solver evalmaxsat. *MaxSAT Evaluation*, 8:364, 2020.

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

[Breiman *et al.*, 1984] L. Breiman, J. Friedman, C.J. Stone, and R.A. Olshen. *Classification and Regression Trees*. Taylor & Francis, 1984.

[Chang and Lin, 2011] Chih-Chung Chang and Chih-Jen Lin. Libsvm: a library for support vector machines. *ACM transactions on intelligent systems and technology (TIST)*, 2(3):1–27, 2011.

[Chu *et al.*, 2023] Yi Chu, Shaowei Cai, and Chuan Luo. Nuwls: improving local search for (weighted) partial maxsat by new weighting techniques. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 37, pages 3915–3923, 2023.

[De Moura and Bjørner, 2008] Leonardo De Moura and Nikolaj Bjørner. Z3: An efficient smt solver. In *International conference on Tools and Algorithms for the Construction and Analysis of Systems*, pages 337–340. Springer, 2008.

[Demirović *et al.*, 2022] Emir Demirović, Anna Lukina, Emmanuel Hebrard, Jeffrey Chan, James Bailey, Christopher Leckie, Kotagiri Ramamohanarao, and Peter J Stuckey. Murtree: Optimal decision trees via dynamic programming and search. *Journal of Machine Learning Research*, 23(26):1–47, 2022.

[ECNU, 2021] ECNU. Oblique decision tree in python. https://github.com/zhenlingcn/scikit-obliquetree, 2021.

[Heath *et al.*, 1993] David Heath, Simon Kasif, and Steven Salzberg. Induction of oblique decision trees. In *IJCAI*, volume 1993, pages 1002–1007. Citeseer, 1993.

[Interpretable AI, 2024] LLC Interpretable AI. Interpretable ai documentation, 2024.

[Kelly *et al.*, 2023] Markelle Kelly, Rachel Longjohn, and Kolby Nottingham. The uci machine learning repository. *URL https://archive. ics. uci. edu*, 2023.

[Murthy *et al.*, 1994] Sreerama K Murthy, Simon Kasif, and Steven Salzberg. A system for induction of oblique decision trees. *Journal of artificial intelligence research*, 2:1–32, 1994.

[Narodytska *et al.*, 2018] Nina Narodytska, Alexey Ignatiev, Filipe Pereira, and Joao Marques-Silva. Learning optimal decision trees with sat. In *International Joint Conference on Artificial Intelligence 2018*, pages 1362–1368. Association for the Advancement of Artificial Intelligence (AAAI), 2018.

[Nijssen and Fromont, 2007] Siegfried Nijssen and Elisa Fromont. Mining optimal decision trees from itemset lattices. In *Proceedings of the 13th ACM SIGKDD international conference on Knowledge discovery and data mining*, pages 530–539, 2007.

[Quinlan, 1986] J. Ross Quinlan. Induction of decision trees. *Machine learning*, 1:81–106, 1986.

[Quinlan, 1993] J. Ross Quinlan. *C4.5: Programs for Machine Learning*. Morgan Kaufmann Publishers Inc., 1993.

[Shati *et al.*, 2021] Pouya Shati, Eldan Cohen, and Sheila McIlraith. Sat-based approach for learning optimal decision trees with non-binary features. In *27th International Conference on Principles and Practice of Constraint Programming (CP 2021)*. Schloss-Dagstuhl-Leibniz Zentrum für Informatik, 2021.

[Vapnik, 2013] Vladimir Vapnik. *The nature of statistical learning theory*. Springer science & business media, 2013.

[Verwer and Zhang, 2019] Sicco Verwer and Yingqian Zhang. Learning optimal classification trees using a binary linear program formulation. In *Proceedings of the AAAI conference on artificial intelligence*, volume 33, pages 1625–1632, 2019.

[Wickramarachchi *et al.*, 2016] Darshana Chitraka Wickramarachchi, Blair Lennon Robertson, Marco Reale, Christopher John Price, and Jennifer Brown. Hhcart: an oblique decision tree. *Computational Statistics & Data Analysis*, 96:12–23, 2016.

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