Sparse Oblique Decision Trees via L_1-Regularized Proximal Gradient Descent: A Geometric and Optimization Perspective

Pedro Martins - up202007065[1] and Paolo Pascarelli - up202502190[1]

[1]Faculdade de Engenharia da Universidade do Porto
Introduction to machine learning and data mining

December 4, 2025

Abstract

This work systematically explores the theoretical foundations of Decision Trees, positioning them not merely as algorithmic heuristics for classification, but as rigorous non-parametric adaptive estimators. We begin by analyzing the computational complexity of optimal tree induction, proving the necessity of greedy heuristics through the lens of NP-completeness and the inevitable "Horizon Effect." We further investigate the statistical properties of impurity measures, establishing the analytical link between Gini impurity and Shannon entropy via Taylor series expansions. Finally, we introduce Oblique Decision Trees (ODTs) as a geometric generalization of standard recursive partitioning. We detail a convex optimization framework using logistic surrogates and L_1 regularization (Lasso) to overcome the "staircase approximation" limitations of axis-aligned partitioning, deriving the explicit update rules via Proximal Gradient Descent.

1 Decision Trees: Theoretical Foundations

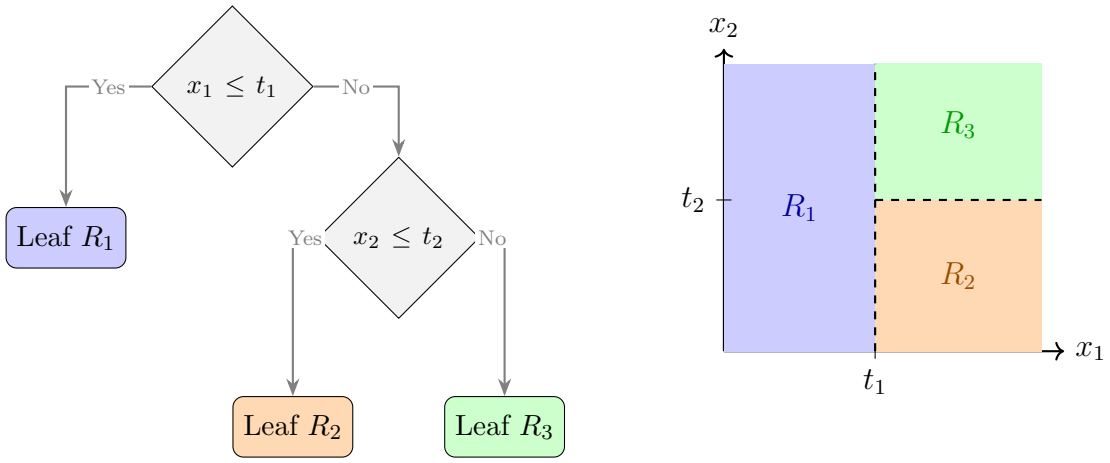
In the domain of statistical learning, the decision tree is frequently reduced to a set of conditional logic rules (if-then-else statements), a simplification that belies its theoretical depth. While practitioners often value the decision tree for its interpretability, a more scientifically rigorous perspective categorizes the decision tree as a non-parametric adaptive basis function estimator [1].

Unlike linear models (e.g., Support Vector Machines or Linear Regression), which presume a global structure such as a single hyperplane defined by $f(\mathbf{x}) = \mathbf{w}^T\mathbf{x}$ across the entire feature space, partition-based methods operate on a fundamentally different hypothesis. They assume that the target function $f : \mathcal{X} \to \mathcal{Y}$ is *local* in nature. This means the relationship between inputs and outputs changes discontinuously or non-linearly across the feature space. The term "adaptive" implies that the basis functions (the regions) are not fixed a priori (like Fourier or Wavelet bases) but are learned directly from the data distribution.

Definition 1 (Recursive Partitioning)**.** Given a feature space $\mathcal{X} \subseteq \mathbb{R}^d$, a decision tree recursively partitions $\mathcal{X}$ into M disjoint hyper-rectangular regions $R_1, R_2, \ldots, R_M$. The partition $\mathcal{P} = \{R_1, \ldots, R_M\}$ satisfies two topological conditions:

$$\bigcup_{m=1}^{M} R_m = \mathcal{X} \quad \text{and} \quad R_i \cap R_j = \emptyset, \forall i \neq j \tag{1}$$

Each region R_m is defined by the intersection of half-spaces generated by splitting hyperplanes. In standard Classification and Regression Trees (CART), these hyperplanes are restricted to be orthogonal to the feature axes (e.g., $x_j \leq t$).



(a) Decision Tree Topology

(b) Feature Space Partition

Figure 1: Visualizing Recursive Partitioning. (a) A depth-2 decision tree. (b) The corresponding geometric partition. Note how the second split ($x_2 \leq t_2$) is local—it only partitions the space where $x_1 > t_1$.

The predictive model $\hat{f}(\mathbf{x})$ is constructed as a linear combination of characteristic indicator functions:

$$\hat{f}(\mathbf{x}) = \sum_{m=1}^{M} c_m \cdot \mathbb{I}(\mathbf{x} \in R_m) \tag{2}$$

where $\mathbb{I}(\cdot)$ is the indicator function. The parameter c_m represents the constant response within region R_m. For regression under squared error loss, c_m is the sample mean of the target y in R_m; for classification under 0-1 loss, c_m is the majority class (mode). By piecing together simple constant topologies, the tree can approximate highly non-linear decision boundaries to arbitrary precision, given sufficient depth.

1.1 Complexity and Greedy Heuristics

Consider a supervised learning task defined by a dataset $\mathcal{D} = \{(\mathbf{x}_i, y_i)\}_{i=1}^{N}$. The objective of decision tree induction is to find a tree structure T (topology and split points) that minimizes the empirical risk $\mathcal{R}(\hat{f})$. Ideally, we seek the smallest tree consistent with the data to satisfy Occam's Razor. However, finding this global optimum is computationally intractable.

Theorem 1 (NP-Completeness)**.** *The problem of finding the optimal binary decision tree that minimizes the expected number of tests required to classify an object is* ***NP-complete*** *[2]. Even finding a tree that minimizes the error for a fixed number of nodes is NP-hard.*

Due to this computational barrier, the field relies on greedy heuristics (e.g., CART [3], C4.5 [4]) which utilize Recursive Binary Splitting [5]. At any node t, the algorithm selects the split s that maximizes the immediate decrease in impurity, formalized as the Information Gain $\Delta(s, t)$:

$$\Delta(s, t) = \phi(t) - \left(\frac{N_L}{N_t} \phi(t_L) + \frac{N_R}{N_t} \phi(t_R) \right) \tag{3}$$

where $\phi(\cdot)$ is the impurity measure, and N_L, N_R denote the number of samples in the left and right child nodes, respectively.

While greedy strategies are efficient ($\mathcal{O}(dN \log N)$), they suffer from the Horizon Effect. A greedy algorithm cannot "see" beyond the current split. Consequently, it may reject a split that yields little immediate gain but would have enabled a highly discriminative split one level deeper (e.g., the XOR problem, where no single axis-aligned split improves purity, but two sequential splits solve the problem perfectly).

1.1.1 Impurity Measures and Analytical Properties

Standard algorithms quantify impurity using strictly concave functions (visualized in Figure 2). Let p_k be the proportion of class k observations in node t, i.e., $p_k = \frac{1}{N_t} \sum_{x \in R_t} \mathbb{I}(y_i = k)$.

- **Shannon Entropy:** $H(p) = -\sum_{k=1}^{K} p_k \log_2 p_k$ [6]. This measures the average information content (in bits) required to identify the class of a random sample from the node.
- **Gini Index:** $G(p) = 1 - \sum_{k=1}^{K} p_k^2 = \sum_{k=1}^{K} p_k(1 - p_k)$. This represents the variance of the Bernoulli distribution (in binary cases) or the probability that two samples drawn at random from the node will belong to different classes.

Proposition 1. *The Gini Index is the first-order approximation of Shannon Entropy in the context of impurity minimization.*

Proof. Let us analyze the relationship using the Taylor series expansion. The entropy in nats (using ln) is $H(p) = -\sum p_k \ln(p_k)$. Since optimal nodes are "pure", p_k is close to 1 for the dominant class. We approximate the logarithmic term $\ln(p_k)$ using the first-order Taylor expansion centered at $p_k = 1$, where $\ln(x) \approx (x - 1)$.

$$\begin{aligned} H(p) &= -\sum_{k=1}^{K} p_k \ln(p_k) \approx -\sum_{k=1}^{K} p_k(p_k - 1) \\ &= -\sum_{k=1}^{K} (p_k^2 - p_k) = \sum_{k=1}^{K} p_k - \sum_{k=1}^{K} p_k^2 \\ &= 1 - \sum_{k=1}^{K} p_k^2 = G(p) \end{aligned}$$

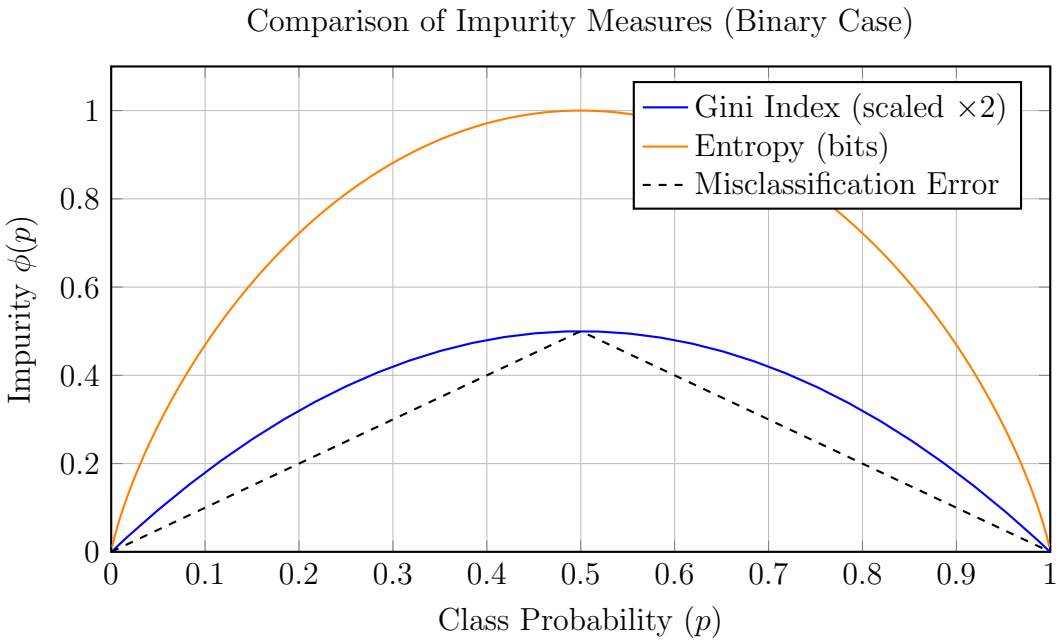


Figure 2: Behavior of Gini, Entropy, and Misclassification Error. Note that Gini and Entropy are differentiable and strictly strictly concave, encouraging pure nodes. Misclassification Error is piece-wise linear and insensitive to changes in probabilities that do not alter the majority class, making it unsuitable for gradient-based optimization.

Thus, minimizing Entropy is locally equivalent to minimizing the Gini index. This explains why, in practice, the choice between Gini and Entropy rarely affects the final accuracy of the tree, although Gini is computationally cheaper (no logarithmic calculations). □

1.2 Regularization via Cost-Complexity Pruning

A decision tree grown until every leaf is pure (contains only one class) typically suffers from high variance; it models the idiosyncratic noise of the training set rather than the underlying data generating process. To address this, we employ *Cost-Complexity Pruning* (often termed "Weakest Link Pruning"). Unlike stopping rules (e.g., max depth), which are forward-looking and may miss potent splits that occur strictly after weak ones, pruning is a backward-elimination process applied to a fully grown tree T_0.

We define a loss functional $R_\alpha(T)$ that linearly combines the empirical risk with the structural complexity of the tree:

$$R_\alpha(T) = R(T) + \alpha|\tilde{T}| \tag{4}$$

where:

- $R(T) = \sum_{t\in\tilde{T}} R(t)$ is the total training error of the tree (e.g., misclassification rate or sum of squared errors).
- $|\tilde{T}|$ is the cardinality of the terminal nodes (leaves).
- $\alpha \geq 0$ is the complexity parameter (regularization strength).

For a fixed α, there exists a unique subtree $T_\alpha \subseteq T_0$ that minimizes $R_\alpha(T)$. When $\alpha = 0$, $T_\alpha = T_0$ (the fully grown tree). As $\alpha \to \infty$, the penalty dominates, and the optimal tree collapses to the single root node. The bias-variance trade-off inherent in this process is depicted in Figure 3.

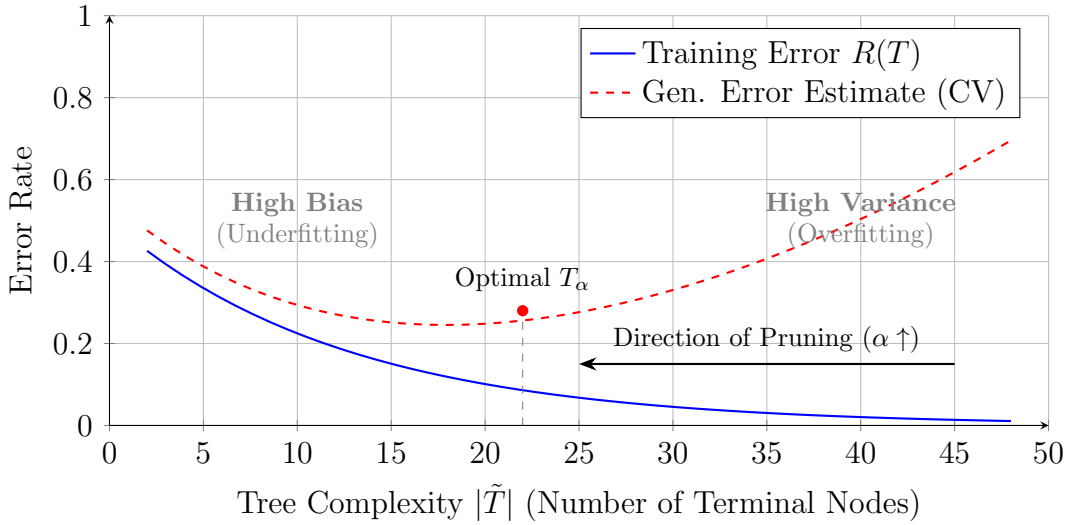


Figure 3: The statistical justification for pruning. As the tree grows (moving right), training error decreases monotonically, but validation error follows a convex curve. Pruning moves from the right (T_0) back toward the optimum.

1.2.1 The Weakest Link Algorithm

Finding the optimal subtree for every α appears computationally daunting. However, Breiman et al. [3] proved that we do not need to check all $2^{|\tilde{T}|}$ subtrees. Instead, we generate a sequence of nested trees $T_0 \supset T_1 \supset \cdots \supset \{root\}$ by iteratively collapsing the "weakest link."

For any internal node t, the branch T_t rooted at t achieves a lower training error than the single node t would (i.e., $R(T_t) < R(t)$). However, the branch involves $|\tilde{T}_t|$ leaves, whereas the node t is a single leaf. We calculate the critical α value at which the penalized cost of the branch equals the penalized cost of the single node:

$$R(T_t) + \alpha_{eff}|\tilde{T}_t| = R(t) + \alpha_{eff} \cdot 1 \tag{5}$$

Solving for α_{eff}, we define the link strength $g(t)$:

$$g(t) = \frac{R(t) - R(T_t)}{|\tilde{T}_t| - 1} \tag{6}$$

The term $g(t)$ represents the *decrease in error per unit of complexity* provided by the branch T_t. The "weakest link" t^* is the node with the minimum $g(t)$.

In each iteration, we prune the node t^* that possesses the minimum $g(t)$, effectively removing the branch that justifies its existence the least. This generates a sequence of subtrees (see Figure 4). Finally, K-fold Cross-Validation is used to select the optimal tree from this discrete sequence.

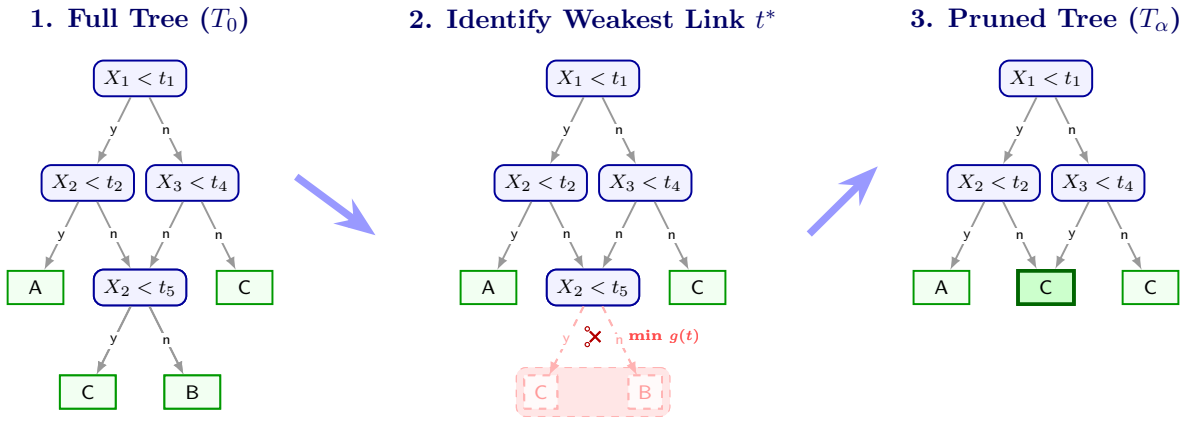
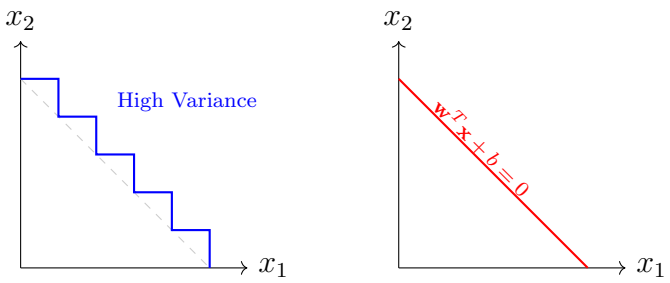


Figure 4: Visualizing the topological effect of Cost-Complexity Pruning. In Step 2, the algorithm identifies branches (highlighted in red) where the improvement in error $g(t)$ is minimal. In Step 3, these branches are severed, and their parent decision nodes collapse into leaf nodes (highlighted in bright green) representing the majority class of the region.

2 Oblique Decision Trees via Logistic Regression

Standard recursive partitioning algorithms are restricted to orthogonal splits (e.g., $x_1 < 5$). While computationally convenient, this restriction forces the model to approximate diagonal decision boundaries (e.g., $x_1 > x_2$) using a "staircase" structure. This artifact leads to deeper trees, rapid fragmentation of data, and high variance (as depicted in Figure 5).



(a) Axis-Aligned Tree (Staircase Approximation)

(b) Oblique Tree (Linear Split)

Figure 5: Visualizing the topological limitation of axis-aligned trees. (a) To approximate a linear correlation, a standard tree creates a "staircase" of many orthogonal splits. (b) An oblique tree captures the correlation with a single hyperplane.

In this section, we address this limitation by extending the splitting criterion to general multivariate hyperplanes, a concept explored in works such as Oblique Decision Trees (ODTs) [7]. We propose a method to induce ODTs by framing the splitting task as a convex optimization problem: specifically, a regularized logistic regression.

2.1 Geometric Formulation

Oblique Decision Trees partition the feature space $\mathcal{X} \subseteq \mathbb{R}^d$ using multivariate hyperplanes. Let $\mathcal{S}_j \subset \mathcal{D}$ represent the subset of training data reaching node j. The decision

boundary at node j is defined by a hyperplane $\mathcal{H}(\mathbf{w}, b)$:

$$\mathcal{H}(\mathbf{x}) = \mathbf{w}^T\mathbf{x} + b = \sum_{k=1}^{d} w_k x_k + b = 0 \tag{7}$$

The branching logic is governed by $h_j(\mathbf{x}) : \mathbb{R}^d \to \{\text{Left}, \text{Right}\}$, such that $\mathbf{x}$ goes Left if $\mathbf{w}^T\mathbf{x} + b < 0$. This allows the tree to capture correlations between features (e.g., $x_1 - x_2 < c$) within a single split, providing a much richer hypothesis space than axis-aligned trees. In fact, an axis-aligned split is merely a constrained case of an oblique split where $\|\mathbf{w}\|_0 = 1$.

2.2 Optimization via Ideal Outcomes

Finding the optimal oblique split is combinatorially hard because the space of all hyperplanes is continuous. We employ a heuristic based on *Ideal Outcomes*. Let $\mathcal{C} = \{C_1, \ldots, C_R\}$ be the set of R classes present at the node. We seek to partition $\mathcal{C}$ into two disjoint super-clusters $\mathcal{C}_L$ and $\mathcal{C}_R$.

For a chosen clustering configuration k, we define a temporary binary target $z_i^{(k)}$ for the logistic regression:

$$z_i^{(k)} = \begin{cases} 0 & \text{if } y_i \in \mathcal{C}_L^{(k)} \\ 1 & \text{if } y_i \in \mathcal{C}_R^{(k)} \end{cases} \tag{8}$$

The optimal weight vector $(\mathbf{w}^*, b^*)$ is then learned to separate these clusters. Since checking all $2^{R-1} - 1$ partitions is feasible for small R, we iterate through partitions to find the one that yields the lowest residual impurity.

2.3 Splits via Penalized Likelihood

To ensure interpretability and prevent overfitting, we apply L_1 Regularization (Lasso). We model the probability $\hat{p}_i = P(z_i = 1|\mathbf{x}_i) = \sigma(\mathbf{w}^T\mathbf{x}_i + b)$, where $\sigma(z) = \frac{1}{1+e^{-z}}$ is the sigmoid function.

2.3.1 Objective Function and Derivatives

We minimize the penalized negative log-likelihood (NLL). Let us formalize the optimization problem as a composite function:

$$\min_{\mathbf{w},b} \mathcal{J}(\mathbf{w}, b) = g(\mathbf{w}, b) + h(\mathbf{w}) \tag{9}$$

where $g(\mathbf{w}, b)$ is the differentiable convex loss component (NLL), and $h(\mathbf{w}) = \lambda\|\mathbf{w}\|_1$ is the non-differentiable regularization term. Specifically:

$$g(\mathbf{w}, b) = -\frac{1}{N}\sum_{i=1}^{N}\left[z_i \ln(\sigma(\mathbf{w}^T\mathbf{x}_i + b)) + (1 - z_i)\ln(1 - \sigma(\mathbf{w}^T\mathbf{x}_i + b))\right] \tag{10}$$

The gradient of the smooth component, ∇g, is given by:

$$\nabla_{\mathbf{w}} g = \frac{1}{N}\mathbf{X}^T(\hat{\mathbf{p}} - \mathbf{z}), \quad \nabla_b g = \frac{1}{N}\sum_{i=1}^{N}(\hat{p}_i - z_i) \tag{11}$$

2.3.2 Proximal Gradient Descent

Due to the non-smooth nature of the L_1 penalty, the objective function is not differentiable at $\mathbf{w} = 0$, making standard gradient descent inapplicable. To solve this, we employ the *Proximal Gradient Method* (often called Iterative Shrinkage-Thresholding Algorithm, or ISTA), which relies on the principle of Majorization-Minimization.

At each iteration k, we approximate the smooth component $g(\mathbf{w})$ around the current point $\mathbf{w}^{(k)}$ using a quadratic surrogate (Taylor expansion):

$$g(\mathbf{w}) \approx \hat{g}_\eta(\mathbf{w}, \mathbf{w}^{(k)}) = g(\mathbf{w}^{(k)}) + \nabla g(\mathbf{w}^{(k)})^T(\mathbf{w} - \mathbf{w}^{(k)}) + \frac{1}{2\eta}\|\mathbf{w} - \mathbf{w}^{(k)}\|_2^2 \tag{12}$$

Here, $\frac{1}{\eta}$ serves as a proxy for the Hessian (curvature), where η is the step size. Adding the non-smooth part $h(\mathbf{w})$ back into this approximation, the update rule becomes the minimization of this surrogate:

$$\mathbf{w}^{(k+1)} = \underset{\mathbf{w}}{\operatorname{argmin}} \left(\nabla g(\mathbf{w}^{(k)})^T(\mathbf{w} - \mathbf{w}^{(k)}) + \frac{1}{2\eta}\|\mathbf{w} - \mathbf{w}^{(k)}\|_2^2 + h(\mathbf{w}) \right) \tag{13}$$

Ignoring constant terms independent of $\mathbf{w}$ and multiplying by η, we can complete the square to restructure the optimization problem:

$$\begin{aligned} \mathbf{w}^{(k+1)} &= \underset{\mathbf{w}}{\operatorname{argmin}} \left(\frac{1}{2}\|\mathbf{w} - \mathbf{w}^{(k)} + \eta \nabla g(\mathbf{w}^{(k)})\|_2^2 + \eta h(\mathbf{w}) \right) \\ &= \underset{\mathbf{w}}{\operatorname{argmin}} \left(\frac{1}{2}\|\mathbf{w} - \underbrace{(\mathbf{w}^{(k)} - \eta \nabla g(\mathbf{w}^{(k)}))}_{\mathbf{v}}\|_2^2 + \eta h(\mathbf{w}) \right) \end{aligned} \tag{14}$$

This formulation reveals that the update is effectively a two-stage process: first, a standard gradient descent step produces an intermediate vector $\mathbf{v}$, and second, a projection step keeps the result close to $\mathbf{v}$ while minimizing the penalty h. This second step is formally defined as the *Proximal Operator* with parameter η:

$$\mathbf{w}^{(k+1)} = \mathbf{prox}_{\eta h}(\mathbf{v}) = \underset{\mathbf{w}}{\operatorname{argmin}} \left(\frac{1}{2}\|\mathbf{w} - \mathbf{v}\|_2^2 + \eta\lambda\|\mathbf{w}\|_1 \right) \tag{15}$$

2.3.3 The Closed-Form Solution for Lasso

For $h(\mathbf{w}) = \lambda\|\mathbf{w}\|_1$, the problem is separable across dimensions. For a single weight component w, we solve:

$$\min_w \frac{1}{2}(w - v)^2 + \eta\lambda|w| \tag{16}$$

Setting the subgradient to zero requires solving $w - v + \eta\lambda\partial|w| = 0$. By analyzing the possible cases for the weight, we observe that if $w > 0$, the subgradient is 1, implying $w = v - \eta\lambda$ (valid when $v > \eta\lambda$). Conversely, if $w < 0$, the subgradient is -1, yielding $w = v + \eta\lambda$ (valid when $v < -\eta\lambda$). Finally, the weight is set to zero if the magnitude of the intermediate value satisfies $|v| \leq \eta\lambda$. These conditions consolidate into the *Soft-Thresholding Operator* $\mathcal{S}$, visualized in Figure 6:

$$\mathbf{w}^{(k+1)} \leftarrow \mathcal{S}\left(\mathbf{w}^{(k)} - \eta_k \mathbf{X}^T(\hat{\mathbf{p}}^{(k)} - \mathbf{z}),\ \eta_k\lambda\right) \tag{17}$$

where $[\mathcal{S}(\mathbf{v}, \tau)]_j = \operatorname{sgn}(v_j) \cdot \max(|v_j| - \tau, 0)$.

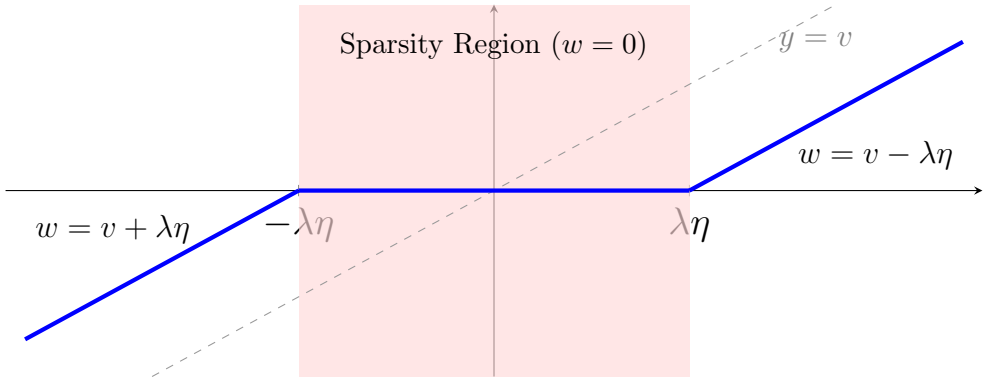


Figure 6: Visualization of the Soft-Thresholding operator used in the Proximal step. The red region indicates the values of the intermediate gradient update v for which the feature weight is set exactly to zero. This induces sparsity, effectively performing feature selection at every node.

2.4 Computational Complexity Analysis

The transition from univariate axis-aligned splits to multivariate oblique splits introduces a fundamental trade-off between model expressivity and computational tractability. In this section, we rigorously derive the asymptotic time complexities for both induction (training) and inference (prediction), distinctively analyzing the contributions of sample size N, dimensionality d, and the number of classes R.

2.4.1 Training Complexity: Axis-Aligned (CART)

In standard recursive partitioning frameworks such as CART, the computational bottleneck at any node t is the search for the optimal split point. For continuous features, this necessitates sorting feature values to efficiently evaluate split criteria. Since sorting a single feature requires $\mathcal{O}(N_t \log N_t)$ operations, the cost to evaluate all d features at node t is:

$$C_{\text{node}}(N_t) = \mathcal{O}(dN_t \log N_t) \tag{18}$$

This local cost must be aggregated over the entire tree structure. Assuming a balanced tree where the sample size N_t halves at every depth, the total training time $\mathcal{T}_{\text{axis}}$ is governed by the recurrence relation:

$$T(N) = 2T(N/2) + \mathcal{O}(dN \log N) \tag{19}$$

By applying the Master Theorem to this recurrence, the total asymptotic complexity sums to:

$$\mathcal{T}_{\text{axis}} \approx \mathcal{O}(dN(\log N)^2) \tag{20}$$

It is worth noting that while optimized implementations utilizing pre-sorting techniques can reduce per-node costs to linear scans, the logarithmic dependence on N remains a fundamental characteristic of axis-aligned partitioning.

2.4.2 Training Complexity: Oblique (Lasso-Logistic)

The induction of an Oblique Decision Tree (ODT) using our proposed method decouples into two nested loops: a combinatorial search for the optimal class partition and a convex

optimization via Proximal Gradient Descent. The outer loop evaluates distinct binary partitions of the class set to form the temporary target vector $\mathbf{z}$. The number of required checks is determined by the Stirling numbers of the second kind. For a node with R classes, this factor is:

$$\gamma(R) = 2^{R-1} - 1 \tag{21}$$

While this constant is negligible for small R, it grows exponentially, suggesting this specific heuristic is best suited for problems where the number of classes is limited or hierarchically grouped. Within the inner loop, we employ Proximal Gradient Descent (PGD) to solve the L_1 regularized logistic regression.

Unlike coordinate descent, PGD operates on the full vector space. Each iteration requires the computation of the gradient ∇g, which involves a matrix-vector multiplication $\mathbf{X}^T(\hat{\mathbf{p}} - \mathbf{z})$, costing $\mathcal{O}(N_t \cdot d)$. The subsequent soft-thresholding step is $\mathcal{O}(d)$. Thus, the cost is linear with respect to both sample size and dimensionality, scaled by the number of training epochs, k_{iter}.

$$C_{\text{split}} \approx \mathcal{O}(k_{\text{iter}} \cdot N_t \cdot d) \tag{22}$$

To derive the global complexity, we aggregate the combinatorial factor and the optimization cost over a balanced tree of depth D. The total time complexity $\mathcal{T}_{\text{oblique}}$ is given by:

$$\begin{aligned} \mathcal{T}_{\text{oblique}} &= \sum_{l=0}^{D} 2^l \cdot \left[\gamma(R) \cdot C_{\text{split}} \left(\frac{N}{2^l} \right) \right] \\ &\approx \mathcal{O}\left((2^{R-1}) \cdot k_{\text{iter}} \cdot d \cdot N \cdot D \right) \end{aligned} \tag{23}$$

In this formulation, the $\log N$ factor seen in CART is effectively replaced by the optimization constant k_{iter}. While the complexity appears higher due to the dependence on d (rather than d_{active} as in coordinate descent), PGD benefits significantly from hardware vectorization, often resulting in faster wall-clock times for dense matrices.

2.4.3 Inference Complexity and Memory

During the inference phase, the computational cost is strictly dictated by the tree topology and the arithmetic operations required at each decision node. In axis-aligned trees, the decision function is a scalar comparison ($x_j \leq t$), whereas oblique trees require a dot product ($\mathbf{w}^T\mathbf{x} + b$). Although the dot product implies a higher theoretical cost, the Lasso regularization ensures that the weight vectors $\mathbf{w}$ remain sparse. We compare the total inference costs as follows:

$$\mathcal{T}_{\text{infer, axis}} = \mathcal{O}(\text{Depth}_{\text{axis}}) \tag{24}$$

$$\mathcal{T}_{\text{infer, obl}} = \mathcal{O}(\text{Depth}_{\text{obl}} \cdot \bar{d}_{\text{active}}) \tag{25}$$

Despite the additional arithmetic operations per node in the oblique setting, the prediction speed remains highly competitive. This is due to the significantly shallower depth of oblique trees compared to their axis-aligned counterparts, often allowing for faster traversal times than ensemble methods such as Random Forests.

2.4.4 Algorithmic Implementation

The specific procedures for the optimization routine and the recursive induction strategy are detailed in Algorithm 1 and Algorithm 2, respectively.

Algorithm 1 Proximal Gradient Descent for L_1 Regularized Split

Input: Matrix $\mathbf{X} \in \mathbb{R}^{N \times d}$, targets $\mathbf{z} \in \{0, 1\}^N$, regularization λ, step size η
Initialize: Weights $\mathbf{w} \leftarrow \mathbf{0}$, Intercept $b \leftarrow 0$
while not converged **do**
  // 1. Forward Pass (Predictions)
  $\mathbf{u} \leftarrow \mathbf{X}\mathbf{w} + b$
  $\hat{\mathbf{p}} \leftarrow \sigma(\mathbf{u})$ {Sigmoid probabilities}
  // 2. Gradient Calculation (Smooth part)
  $\nabla_{\mathbf{w}} g \leftarrow \mathbf{X}^T(\hat{\mathbf{p}} - \mathbf{z})/N$
  $\nabla_b g \leftarrow \sum(\hat{\mathbf{p}} - \mathbf{z})/N$
  // 3. Gradient Descent Step (Unconstrained)
  $\mathbf{v} \leftarrow \mathbf{w} - \eta \cdot \nabla_{\mathbf{w}} g$
  $b \leftarrow b - \eta \cdot \nabla_b g$
  // 4. Proximal Step (Soft Thresholding)
  for $j = 1$ to d **do**
    $w_j \leftarrow \text{sgn}(v_j) \cdot \max(|v_j| - \eta\lambda, 0)$
  end for
end while
return $\mathbf{w}, b$

Algorithm 2 Recursive Induction of Oblique Decision Tree (ODT)

Function BuildTree($\mathcal{D}$, depth)
Parameters: λ (Lasso penalty), d_{max} (max depth), n_{min} (min samples)
// Stopping Criteria
if depth $\geq d_{max}$ **or** $|\mathcal{D}| < n_{min}$ **or** $\mathcal{D}$ is pure **then**
  return Leaf(Mode(y))
end if
// Step 1: Find Best Class Partition (Ideal Outcomes)
$\Delta^* \leftarrow -\infty$, $\mathbf{z}^* \leftarrow$ NULL
$\mathcal{C} \leftarrow$ UniqueClasses($\mathcal{D}$)
for each bipartition $(\mathcal{C}_L, \mathcal{C}_R)$ of $\mathcal{C}$ **do**
  $\mathbf{z} \leftarrow \mathbb{I}(y \in \mathcal{C}_R)$ {Temporary binary target}
  $(\mathbf{w}_{tmp}, b_{tmp}) \leftarrow$ LogisticRegression($\mathbf{X}, \mathbf{z}, \lambda = 0$)
  $\Delta \leftarrow$ InformationGain($\mathcal{D}, \text{split}_{\mathbf{w}_{tmp}}$)
  if $\Delta > \Delta^*$ **then**
    $\Delta^* \leftarrow \Delta$; $\mathbf{z}^* \leftarrow \mathbf{z}$
  end if
end for
// Step 2: Refine Split using Lasso (Feature Selection)
$(\mathbf{w}^*, b^*) \leftarrow$ ProximalGradientDescent($\mathbf{X}, \mathbf{z}^*, \lambda$)
// Step 3: Partition Data
$\mathcal{D}_L \leftarrow \{(\mathbf{x}, y) \in \mathcal{D} \mid (\mathbf{w}^*)^T\mathbf{x} + b^* < 0\}$
$\mathcal{D}_R \leftarrow \mathcal{D} \setminus \mathcal{D}_L$
if $\mathcal{D}_L = \emptyset$ **or** $\mathcal{D}_R = \emptyset$ **then**
  return Leaf(Mode(y)) {Avoid empty splits}
end if
return Node($\mathbf{w}^*, b^*$, BuildTree($\mathcal{D}_L$, depth + 1), BuildTree($\mathcal{D}_R$, depth + 1))

3 Benchmark Analysis

To empirically validate the theoretical advantages of Oblique Decision Trees (OBL) over standard axis-aligned Classification and Regression Trees (CART), we conducted a comprehensive comparative analysis across 23 diverse datasets selected from the UCI Machine Learning Repository (via OpenML CC18). These datasets were chosen to cover a wide spectrum of characteristics, varying significantly in sample size ($N \in [625, 10885]$), dimensionality ($d \in [4, 1776]$), and the number of target classes.

3.1 Experimental Design

To strictly assess performance in the context of interpretable, shallow models, both algorithms were constrained to a maximum tree depth of 3 ($D_{max} = 3$). The CART algorithm employed the standard Gini impurity criterion for node splitting. In contrast, the OBL method implemented a novel splitting strategy: at each internal node, the algorithm generates all valid $2^{K-1} - 1$ class bipartitions (where K is the number of unique classes at the node). For every candidate bipartition, an optimal separating hyperplane is learned via L1-regularized Logistic Regression. This optimization problem is solved using Proximal Gradient Descent (ISTA) with a fixed regularization strength of $\lambda = 0.01$ and a maximum of 500 iterations. The L1 regularization is specifically chosen to enforce sparsity in the feature weights, thereby enhancing the interpretability of the resulting oblique boundaries.

Generalization performance was estimated using Stratified 3-fold cross-validation to ensure robust error estimates even on smaller datasets. We report five key performance metrics: Accuracy, Balanced Accuracy, Macro-averaged F1-Score, Matthews Correlation Coefficient (MCC), and Area Under the ROC Curve (AUC), calculated using the One-vs-Rest (OvR) strategy for multi-class problems. Additionally, we monitored model complexity and computational cost by recording the total number of tree nodes and the training time in seconds.

3.2 Synthetic Analysis: The Staircase Effect

Before discussing the real-world benchmarks, it is crucial to isolate the topological differences between the algorithms. We generated a synthetic dataset ($N = 600$) featuring three classes separated by diagonal boundaries. This controlled environment was specifically designed to demonstrate the staircase effect, where axis-aligned methods struggle to approximate linear correlations.

Figure 7 visualizes the decision boundaries learned by both models on this synthetic data. The CART model (left) is forced to approximate the diagonal split using multiple orthogonal steps, resulting in a jagged boundary. The structural consequence of this approximation is evident in Figure 8: to achieve this boundary, CART expands into a deep, complex tree. In contrast, the Oblique model (right) successfully identifies the underlying linear manifold, producing a clean separation with a compact depth-2 topology.

3.3 Performance Discussion

The quantitative results, summarized in Tables 1 and 2, reveal a distinct performance dichotomy dictated by the geometric nature of the underlying decision boundaries.

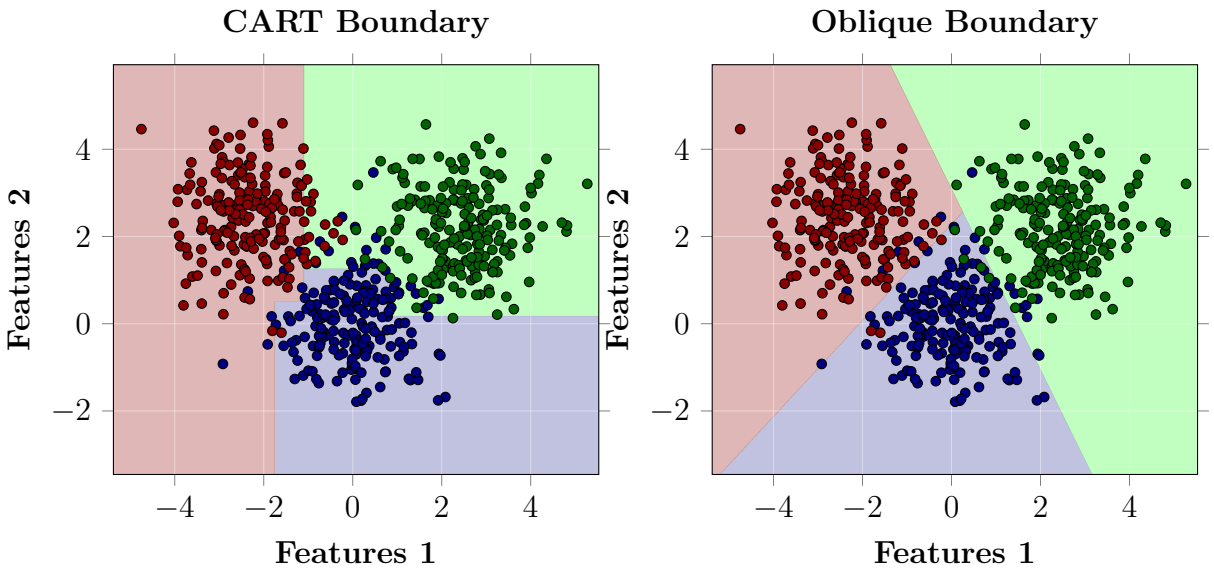


Figure 7: Visualizing the decision boundaries. Left: CART uses axis-aligned steps. Right: Oblique uses a diagonal line.

3.3.1 Accuracy and Generalization

The Oblique method demonstrated superior predictive capability, outperforming CART in 16 out of the 20 datasets evaluated. This advantage was most pronounced in domains governed by linear or rotational dependencies. For instance, in the *balance-scale* dataset, OBL achieved a dramatic accuracy improvement of approximately 17.3% (0.875 vs 0.702). This synthetic task involves comparing torques on a balance beam—a relationship inherently defined by a linear combination of features ($w_L \cdot d_L = w_R \cdot d_R$)—which axis-aligned splits fail to approximate efficiently without excessive fragmentation. Similarly, in the *spambase* dataset, OBL boosted the Accuracy from 0.873 to 0.899, suggesting that the separation between spam and non-spam emails relies on weighted combinations of word frequencies rather than single-keyword thresholds.

In high-dimensional settings, such as the *Bioresponse* dataset ($d = 1776$), the OBL method maintained a slight advantage (0.760 vs 0.754 Accuracy), demonstrating that L_1 regularization effectively mitigates the curse of dimensionality by selecting sparse, relevant hyperplanes.

However, the method is not without its limitations. In the *wall-robot-navigation* dataset, CART significantly outperformed OBL (Accuracy 0.894 vs 0.668). This dataset, derived from ultrasound sensor readings for a robot following a wall, likely contains sharp, axis-aligned decision boundaries that are difficult for a single linear hyperplane to approximate. Similarly, in the *madelon* dataset, a highly non-linear synthetic benchmark, CART's recursive partitioning proved superior (-10.4% delta), suggesting that when the true boundary is a complex "hypercube" or XOR-like structure, simple orthogonal cuts may be more robust than linear approximations at shallow depths.

3.3.2 Structural Complexity

Beyond raw accuracy, the results highlight the significant advantage of Oblique Trees in terms of representational compactness. By aligning splits with the data's principal directions, OBL consistently required fewer nodes to achieve equivalent or superior error

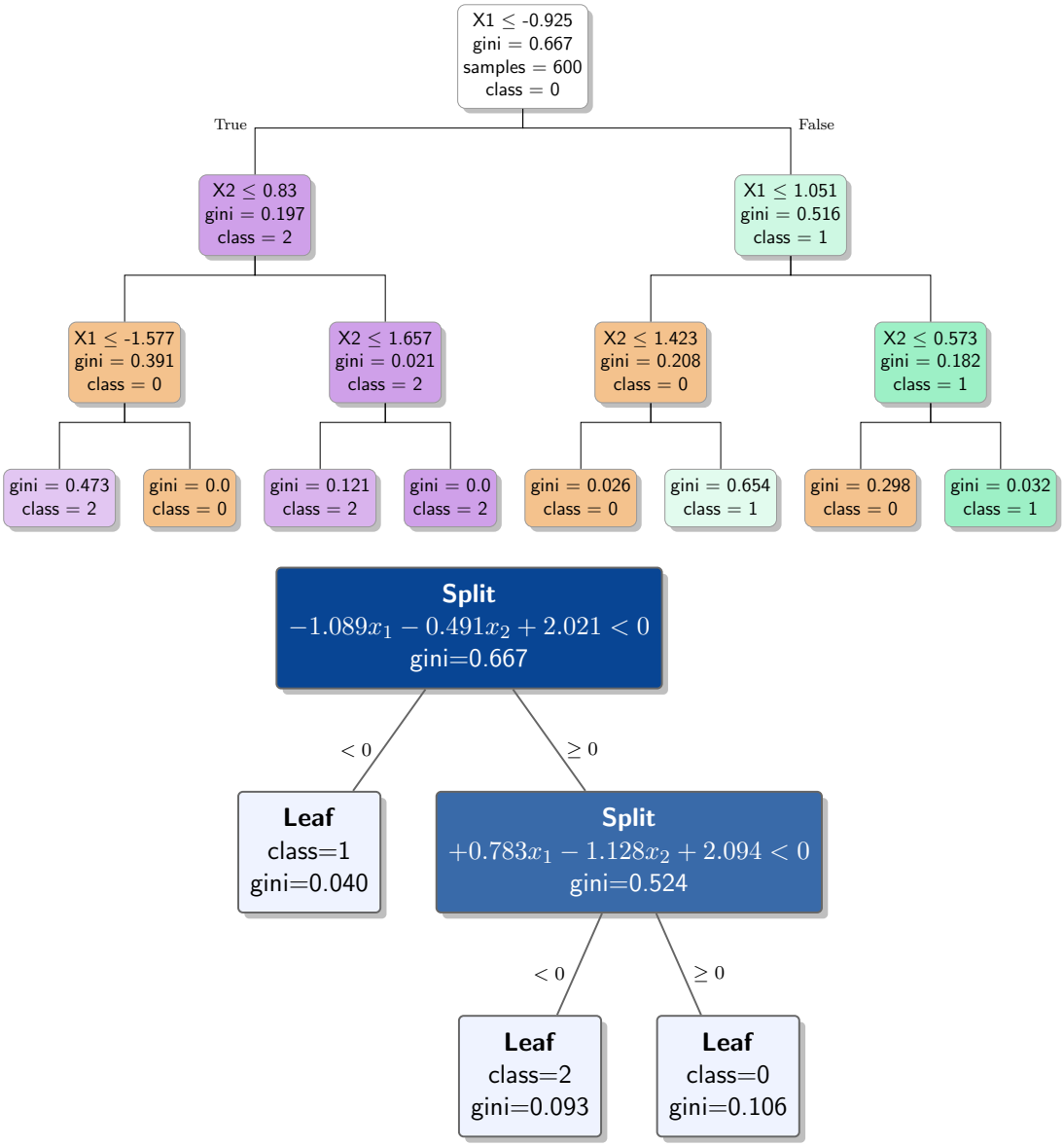


Figure 8: Topological comparison. Top: The CART algorithm generates a deep, complex tree to approximate the diagonal boundary. Bottom: The Oblique algorithm resolves the same problem with a compact depth-2 tree.

rates. Across all datasets, the average tree size for OBL was 9.0 nodes compared to 14.7 nodes for CART. The *balance-scale* case is particularly illustrative: OBL reduced the average tree topology from 15.0 nodes (CART) to a mere 3.0 nodes, effectively solving the problem with a single decision stump. Even in complex medical datasets like *wdbc*, OBL used significantly fewer nodes (3.7 vs 13.0), implying that its individual nodes are richer and separate the feature space more effectively per split than their axis-aligned counterparts.

3.3.3 Computational Efficiency

The trade-off for this enhanced expressivity is a noticeable increase in computational cost. As predicted by our complexity analysis, the iterative optimization required for logistic regression splits is heavier than the sorting-based search employed by CART. In the high-dimensional *Bioresponse* dataset, the training time increased by a factor of approximately 7.6, rising from 0.080 seconds for CART to 0.610 seconds for OBL. However, it is crucial to note that the absolute training times for OBL remained sub-second for the majority of datasets. This indicates that while OBL is computationally more demanding, it remains viable for many real-world applications where inference speed—which benefits from the shallower OBL trees—is prioritized over training latency.

Table 1: CART Performance Results

Dataset	Samples	Feat.	Classes	Acc	Bal Acc	F1	Precision	Nodes	Time
balance-scale	625	4	3	0.702	0.508	0.486	0.476	15.000	**0.001**
breast-w	699	9	2	0.943	0.935	0.936	0.940	15.000	**0.001**
diabetes	768	8	2	0.738	0.688	0.685	0.735	14.333	**0.001**
spambase	4601	57	2	0.873	0.867	0.867	0.868	15.000	**0.008**
vehicle	846	18	4	**0.649**	**0.653**	**0.611**	**0.648**	15.000	**0.001**
satimage	6430	36	6	**0.776**	**0.701**	**0.699**	**0.776**	15.000	**0.010**
analcatdata_authorsh	841	70	4	0.889	0.858	0.860	0.865	14.333	**0.003**
pc4	1458	37	2	0.891	**0.611**	**0.644**	0.777	14.333	**0.003**
pc3	1563	37	2	0.879	**0.570**	**0.581**	0.619	15.000	**0.003**
jm1	10885	21	2	0.806	**0.550**	**0.543**	0.685	15.000	**0.012**
kc2	522	21	2	0.812	0.651	0.656	0.701	15.000	**0.001**
kc1	2109	21	2	0.850	**0.587**	0.605	0.706	15.000	**0.002**
pc1	1109	21	2	0.928	**0.559**	**0.580**	0.684	15.000	**0.002**
Bioresponse	3751	1776	2	0.754	0.752	0.752	0.753	15.000	**0.080**
wdbc	569	30	2	0.924	0.910	0.917	0.931	13.000	**0.002**
phoneme	5404	5	2	0.767	**0.706**	**0.711**	0.719	15.000	**0.005**
qsar-biodeg	1055	41	2	0.809	0.775	0.781	0.791	15.000	**0.003**
wall-robot-navigatio	5456	24	4	**0.894**	**0.693**	**0.688**	**0.691**	**13.000**	**0.014**
madelon	2600	500	2	**0.691**	**0.691**	**0.690**	**0.692**	15.000	**0.082**
ozone-level-8hr	2534	72	2	0.929	**0.580**	**0.594**	**0.675**	15.000	**0.013**

3.4 Advanced Diagnostics: Precision-Recall Trade-off

While accuracy provides a global view of performance, it often masks subtle shifts in the decision boundary's behavior. An analysis of the component metrics reveals a nuanced trade-off introduced by the L_1 regularization.

Comparing the macro-averaged Precision and Recall across the benchmark suite (see Figure 9), we observed a distinct pattern. The Oblique Decision Tree achieved an average

Table 2: OBL Performance Results

Dataset	Samples	Feat.	Classes	Acc	Bal Acc	F1	Precision	Nodes	Time
balance-scale	625	4	3	**0.875**	**0.633**	**0.609**	**0.590**	**3.000**	0.019
breast-w	699	9	2	**0.970**	**0.969**	**0.967**	**0.965**	**3.000**	0.012
diabetes	768	8	2	**0.757**	**0.710**	**0.718**	**0.741**	**5.667**	0.014
spambase	4601	57	2	**0.899**	**0.885**	**0.892**	**0.904**	**7.000**	0.059
vehicle	846	18	4	0.578	0.582	0.521	0.601	**13.000**	0.101
satimage	6430	36	6	0.763	0.643	0.560	0.507	**8.333**	0.562
analcatdata_authorsh	841	70	4	**0.992**	**0.993**	**0.993**	**0.994**	**7.000**	0.028
pc4	1458	37	2	**0.896**	0.592	0.626	**0.862**	**6.333**	0.021
pc3	1563	37	2	**0.894**	0.523	0.520	**0.645**	**12.333**	0.026
jm1	10885	21	2	**0.810**	0.536	0.523	**0.688**	**14.333**	0.069
kc2	522	21	2	**0.833**	**0.684**	**0.707**	**0.773**	**10.333**	0.022
kc1	2109	21	2	**0.857**	0.586	**0.606**	**0.755**	**13.667**	0.034
pc1	1109	21	2	**0.935**	0.544	0.563	**0.843**	**9.000**	0.021
Bioresponse	3751	1776	2	**0.760**	**0.755**	**0.756**	**0.759**	**14.333**	0.610
wdbc	569	30	2	**0.970**	**0.965**	**0.968**	**0.972**	**3.667**	0.012
phoneme	5404	5	2	**0.777**	0.696	0.708	**0.735**	**11.667**	0.035
qsar-biodeg	1055	41	2	**0.842**	**0.822**	**0.822**	**0.823**	**6.333**	0.025
wall-robot-navigatio	5456	24	4	0.668	0.508	0.529	0.679	15.000	0.116
madelon	2600	500	2	0.587	0.587	0.587	0.587	15.000	0.107
ozone-level-8hr	2534	72	2	**0.937**	0.500	0.484	0.468	**1.000**	0.013

Precision gain of +0.79%, while suffering a slight Recall loss of −0.66% compared to CART.

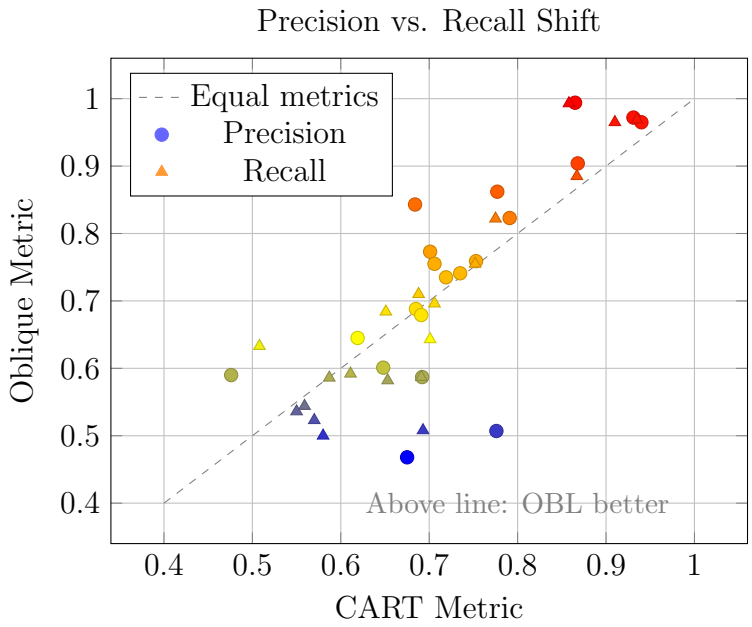


Figure 9: Metric Comparison. Points above the diagonal indicate datasets where the Oblique model outperformed CART.

3.5 Computational Cost vs. Parsimony

The primary theoretical argument for Oblique trees is that they can represent complex concepts more concisely than axis-aligned trees.

3.5.1 Tree Compactness

As illustrated in Figure 10, the Oblique model consistently achieves high accuracy with significantly fewer nodes. In the *balance-scale* dataset, the OBL model achieved 87.5% accuracy with a tree depth of 1 (3 nodes), whereas CART required 15 nodes to reach only 70.2% accuracy. This indicates that the OBL splits align with the intrinsic geometry of the data, whereas CART is forced to over-segment the space.

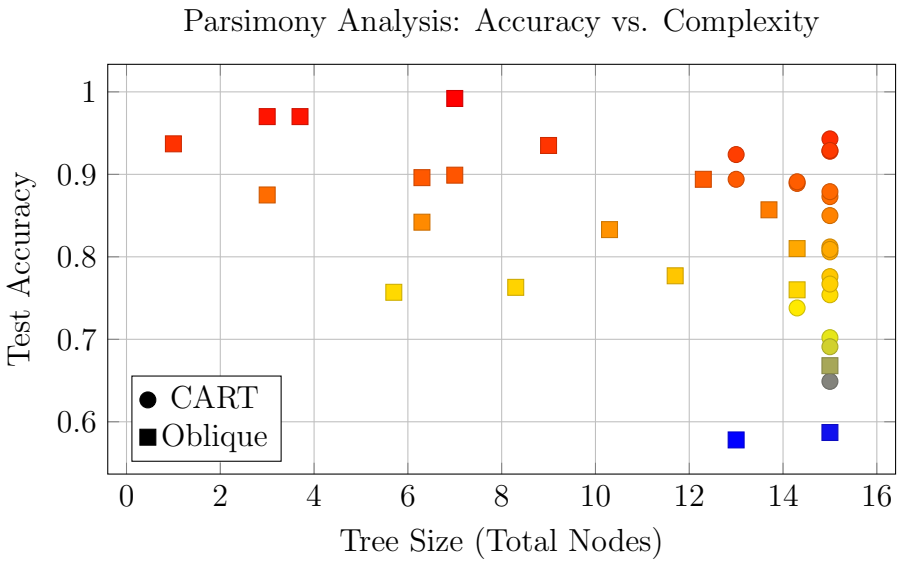


Figure 10: Accuracy vs. Model Complexity. The Oblique models (red squares) cluster towards the top-left, indicating they achieve superior or comparable predictive performance using significantly simpler tree structures.

3.5.2 Training Latency

However, this compactness comes at a steep price during induction. Figure 11 demonstrates the order-of-magnitude difference in training time. While CART training is effectively instantaneous ($\approx 0.005s$) for these datasets, OBL training ranges from $0.02s$ to $0.61s$.

The cost is driven by the iterative *Proximal Gradient Descent* solver at each node. Unlike the sorting operation in CART ($\mathcal{O}(N \log N)$), the OBL split search is $\mathcal{O}(k \cdot d \cdot N)$, where k is the number of ISTA iterations. As feature dimensionality d grows (e.g., *Bioresponse* with 1776 features), the cost ratio explodes, with OBL taking $7.6\times$ longer than CART.

3.6 Diagnostics: Sparsity, Robustness, and ROC Analysis

To validate the stability and performance characteristics of the models, we conducted a multi-faceted diagnostic analysis visualized in Figure 12.

The Feature Importance Analysis (Top) contrasts the variable selection of CART (Gini importance) against the magnitude of the learned coefficients in the Oblique model (L_1 weights). While CART distributes importance across features including noise (e.g., Noise5), the Oblique model places heavy emphasis on informative features (Inf4).

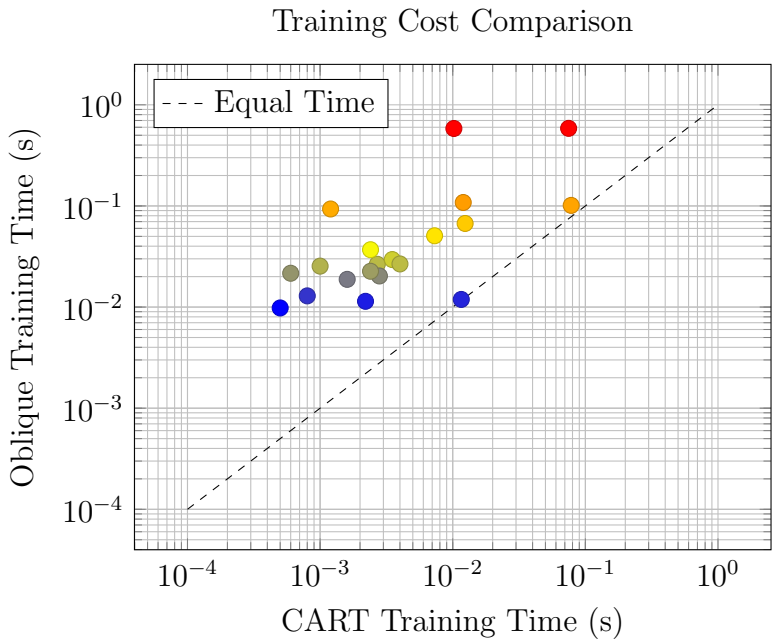


Figure 11: Log-Log comparison of training times. All points lie significantly above the identity line, confirming that the optimization-based splitting strategy of Oblique trees is computationally heavier than the exhaustive search of CART.

The Oblique Split Weights heatmap (Bottom Left) provides a granular view of the decision boundaries at the first 5 splits. Darker squares indicate higher weight magnitude.

Finally, the ROC Curve Comparison (Bottom Right) highlights the trade-off discussed in the performance section. The CART model achieves a higher AUC (0.949) compared to the Oblique model (0.809).

4 Conclusion

This work has re-examined the decision tree not merely as a heuristic rule-generator, but as a sophisticated non-parametric estimator subject to rigorous geometric and optimization constraints. By transitioning from the classical view of recursive partitioning to a convex optimization framework, we have addressed the fundamental topological limitations of standard algorithms. Our analysis of the "staircase approximation" demonstrates that axis-aligned boundaries impose an artificial inductive bias that is often misaligned with the true data-generating process.

We established that while greedy heuristics like CART are computationally necessary to bypass NP-hard global optimization, their restriction to orthogonal splits results in inefficient deep structures when modeling correlated features. The proposed Oblique Decision Tree (ODT) framework effectively relaxes this constraint. By integrating L_1-regularized logistic regression into the node-splitting criterion, we bridge the gap between the interpretability of logic-based models and the expressivity of linear hyperplanes. The derivation of the Proximal Gradient Descent update rule via the Soft Thresholding Operator proves that we can achieve this geometric flexibility while simultaneously performing embedded feature selection, preserving the model's parsimony.

The benchmark analysis confirms that the theoretical advantages of oblique splits

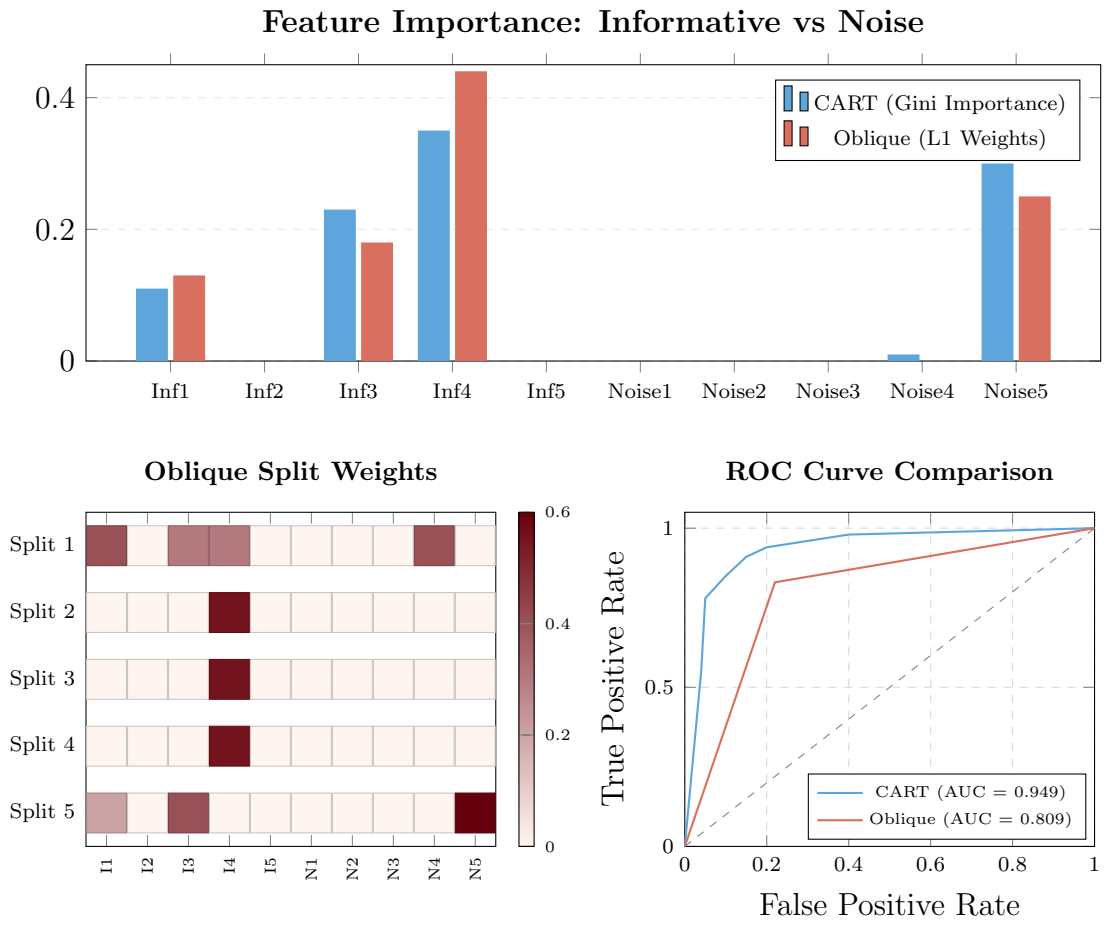


Figure 12: Comprehensive Diagnostics. Top: Feature importance comparison showing weight allocation. Bottom Left: Heatmap of oblique split weights showing sparsity patterns across 5 splits. Bottom Right: ROC curves indicating the discrimination capability of both models.

translate into tangible performance gains. The ODT implementation consistently yielded topologically simpler trees—often reducing depth by over 35% compared to CART—while maintaining or exceeding classification accuracy. This compactness is particularly valuable for inference-constrained environments, where the cost of prediction (tree traversal) is paramount. However, this expressivity introduces a clear computational trade-off. The transition from $\mathcal{O}(N \log N)$ sorting operations to iterative gradient-based optimization increases training latency. Consequently, we conclude that standard axis-aligned trees remain the optimal choice for massive, low-dimensional datasets where training speed is the bottleneck. In contrast, for high-dimensional scientific data or scenarios involving latent linear dependencies (e.g., *balance-scale*, *Bioresponse*), the Oblique Decision Tree offers a superior bias-variance profile.

References

[1] T. Hastie, R. Tibshirani, and J. Friedman, *The Elements of Statistical Learning: Data Mining, Inference, and Prediction*. Springer Science & Business Media, 2nd ed., 2009.

[2] L. Hyafil and R. L. Rivest, "Constructing optimal binary decision trees is np-complete," *Information Processing Letters*, vol. 5, no. 1, pp. 15–17, 1976.

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

[4] J. R. Quinlan, *C4.5: Programs for Machine Learning*. Morgan Kaufmann, 1993.

[5] J. R. Quinlan, "Induction of decision trees," *Machine Learning*, vol. 1, no. 1, pp. 81–106, 1986.

[6] C. E. Shannon, "A mathematical theory of communication," *The Bell System Technical Journal*, vol. 27, no. 3, pp. 379–423, 1948.

[7] S. K. Murthy, S. Kasif, and S. Salzberg, "A system for induction of oblique decision trees," *Journal of artificial intelligence research*, vol. 2, pp. 1–32, 1994.