Optimal Sparse Decision Trees

Xiyang Hu[1], Cynthia Rudin[2], Margo Seltzer[3*]

xiyanghu@cmu.edu, cynthia@cs.duke.edu, mseltzer@cs.ubc.ca

[1]Carnegie Mellon University, [2]Duke University, [3]The University of British Columbia

[*]Author names are in alphabetic order.

Abstract

- Decision Trees: Since early 80's, extremely popular form for interpretable ML models.
- Existing algorithms use greedy splitting and pruning, no optimality.
- OSDT is the first practical algorithm for construction of optimal decision trees for binary variables.
- OSDT combines analytical bounds, computational caching, and fast bit-vector operations to efficiently prune the search space.

Notation

We focus on binary classification, and our decision trees are Boolean functions.

- A tree can be expressed in terms of only its leaves.
- A leaf, p_k, is the classification rule of the path from the root to leaf k.
- Each p_k corresponds to its $\hat{y}_k^{(\text{leaf})}$, for $k = 1, \ldots, H$.
- We represent a decision tree, d as $(d_{un}, \delta_{un}, d_{\text{split}}, \delta_{\text{split}}, K, H)$, where $d_{un} = (p_1, \ldots, p_K)$ are the unchanged leaves of d, $\delta_{un} = (\hat{y}_1^{(\text{leaf})}, \ldots, \hat{y}_K^{(\text{leaf})}) \in \{0,1\}^K$ are the predicted labels of leaves d_{un}, $d_{\text{split}} = (p_{K+1}, \ldots, p_H)$ are the leaves we are going to split, and $\delta_{\text{split}} = (\hat{y}_{K+1}^{(\text{leaf})}, \ldots, \hat{y}_H^{(\text{leaf})}) \in \{0,1\}^{H-K}$ are the predicted labels of leaves d_{split}.

Objective Function

For a tree $d = (d_{un}, \delta_{un}, d_{\text{split}}, \delta_{\text{split}}, K, H)$, we define its objective function as a combination of the misclassification error and a sparsity penalty on the number of leaves:

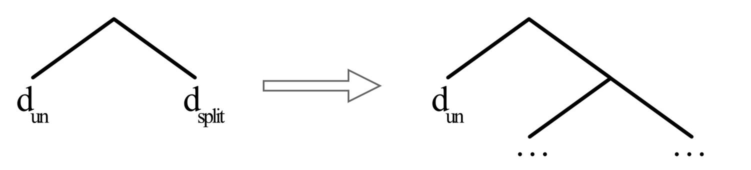
$$R(d, \mathbf{x}, \mathbf{y}) = \ell(d, \mathbf{x}, \mathbf{y}) + \lambda H(d). \tag{1}$$

$H(d)$ is the number of leaves in the tree d. $R(d, \mathbf{x}, \mathbf{y})$ is a regularized empirical risk. The loss $\ell(d, \mathbf{x}, \mathbf{y})$, is the misclassification error of d, *i.e.*, the fraction of training data with incorrectly predicted labels.

Optimization Framework

We minimize the objective function based on a branch-and-bound framework. We prove a series of useful bounds that work together to eliminate a large part of the search space.

Hierarchical objective lower bound

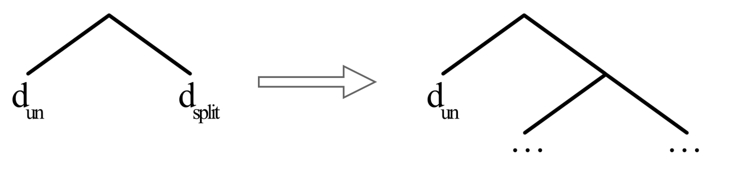


Tree T | any child tree t of Tree T

lower bound b(T) | Objective R(t)≥b(T)

Optimization Framework Cont'd

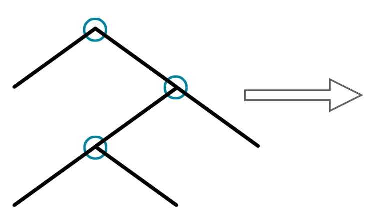
Objective lower bound with one-step lookahead



Tree T | any child tree t of Tree T

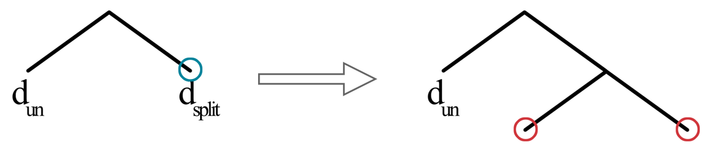
If b(T)+λ≥R^c | Objective R(t)≥b(T)+λ≥R^c

Lower bound on node support



In an optimal tree, for any internal node n, its support supp(n)≥2λ.

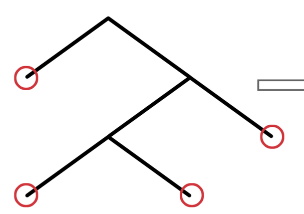
Lower bound on incremental classification accuracy



Tree T | get two new leaves

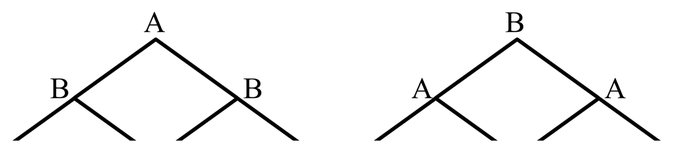
split one leaf | Incremental accuracy≥λ

Leaf accurate support bound



In an optimal tree, for any leaf l, its classification accuracy≥λ.

Leaf permutation bound



Tree T_1 | Tree T_2

T_1 and T_2 are actually the same, only need to evaluate one of them.

Equivalent points bound

index	Feature A	Feature B	Feature C	Feature D	...	Lable
1	1	0	1	1	...	0
2	1	0	1	1	...	1
3	1	0	1	1	...	1
4	1	0	1	1	...	1
5	1	0	1	1	...	0
...	...	...	...	...	...	...
n	1	0	1	1	...	1

For a given dataset, if there are multiple samples with exactly the same features but different labels, then no matter how we build our classifier, we will always predict some of these points incorrectly.

Algorithm

The loss can be decomposed into two parts corresponding to the unchanged leaves and the leaves to be split:

- $\ell(d, \mathbf{x}, \mathbf{y}) \equiv \ell_p(d_{un}, \delta_{un}, \mathbf{x}, \mathbf{y}) + \ell_q(d_{\text{split}}, \delta_{\text{split}}, \mathbf{x}, \mathbf{y})$, where $d_{un} = (p_1, \ldots, p_K)$, $\delta_{un} = (\hat{y}_1^{(\text{leaf})}, \ldots, \hat{y}_K^{(\text{leaf})})$, $d_{\text{split}} = (p_{K+1}, \ldots, p_H)$ and $\delta_{\text{split}} = (\hat{y}_{K+1}^{(\text{leaf})}, \ldots, \hat{y}_H^{(\text{leaf})})$;
- $\ell_p(d_{un}, \delta_{un}, \mathbf{x}, \mathbf{y}) = \frac{1}{N}\sum_{n=1}^{N}\sum_{k=1}^{K} \text{cap}(x_n, p_k) \wedge \mathbb{1}[\hat{y}_k^{(\text{leaf})} \neq y_n]$ is the proportion of data in the unchanged leaves that are misclassified;
- $\ell_p(d_{\text{split}}, \delta_{\text{split}}, \mathbf{x}, \mathbf{y}) = \frac{1}{N}\sum_{n=1}^{N}\sum_{k=K+1}^{H} \text{cap}(x_n, p_k) \wedge \mathbb{1}[\hat{y}_k^{(\text{leaf})} \neq y_n]$ is the proportion of data in the leaves we are going to split that are misclassified.
- Define a lower bound $b(d_{un}, \mathbf{x}, \mathbf{y})$ on the objective by leaving out the latter loss,

$$b(d_{un}, \mathbf{x}, \mathbf{y}) \equiv \ell_p(d_{un}, \delta_{un}, \mathbf{x}, \mathbf{y}) + \lambda H \leq R(d, \mathbf{x}, \mathbf{y}), \tag{2}$$

where the leaves d_{un} are kept and the leaves d_{split} are going to be split. Here, $b(d_{un}, \mathbf{x}, \mathbf{y})$ gives a lower bound on the objective of *any* child tree of d.

Algorithm 1 Incremental branch-and-bound for learning optimal decision trees.

Input: Objective function $R(d, \mathbf{x}, \mathbf{y})$, objective lower bound $b(d_{un}, \mathbf{x}, \mathbf{y})$, set of features $S = \{s_m\}_{m=1}^{M}$, training data $(\mathbf{x}, \mathbf{y}) = \{(x_n, y_n)\}_{n=1}^{N}$, regularization parameter λ
Output: Provably optimal decision tree d^* with minimum objective R^*

$d^c \leftarrow ((), (), (), \delta_{\text{split}}, 0, 0)$ ▷ For simplicity, cold start with an empty tree
$R^c \leftarrow R(d^c, \mathbf{x}, \mathbf{y})$ ▷ Initialize current best objective
$Q \leftarrow \text{queue}([d^c])$ ▷ Initialize queue with the empty tree
while Q not empty **do** ▷ Optimization complete when the queue is empty
  $d = (d_{un}, \delta_{un}, d_{\text{split}}, \delta_{\text{split}}, K, H) \leftarrow Q.\text{pop}()$ ▷ Remove tree d from the queue
  for every possible combination of features to split d_{split} **do**
    split d_{split} and get new leaves d_{new}
    CheckLeafSupportAndAccuracyBound(d_{new})
    for each possible subset d'_{split} of d_{new} **do**
      $d'_{un} = d_{un} \cup (d_{\text{new}} \setminus d'_{\text{split}})$
      $d' = (d'_{un}, \delta'_{un}, d'_{\text{split}}, \delta'_{\text{split}}, K', H')$ ▷ **Branch**: Generate child tree d'
      $\Delta_{un} = \frac{1}{N}\sum_{n=1}^{N}\sum_{k=K+1}^{K'} \text{cap}(x_n, p_k) \wedge \mathbb{1}[\hat{y}_k^{(\text{leaf})} \neq y_n]$
      $b(d'_{un}, \mathbf{x}, \mathbf{y}) \leftarrow b(d_{un}, \mathbf{x}, \mathbf{y}) + \lambda(H' - H) + \Delta_{un}$
      $\Delta_R = \frac{1}{N}\sum_{n=1}^{N}\sum_{k=K'+1}^{H'} \text{cap}(x_n, p_k) \wedge \mathbb{1}[\hat{y}_k^{(\text{leaf})} \neq y_n]$
      $R(d', \mathbf{x}, \mathbf{y}) \leftarrow b(d'_{un}, \mathbf{x}, \mathbf{y}) + \Delta_R$
      if $R(d', \mathbf{x}, \mathbf{y}) < R^c$ **then**
        $(d^c, R^c) \leftarrow (d', R(d', \mathbf{x}, \mathbf{y}))$ ▷ Update d^c and R^c
      end if
      if $b(d'_{un}, \mathbf{x}, \mathbf{y}) + \sum_{l \in d'_{\text{split}}} b_0(l, \mathbf{x}, \mathbf{y}) + \lambda \times L(d'_{\text{split}}) < R^c$ **then**
        ▷ **Bound**: Apply bound from **Theorem 1**, combined with **Equivalent points bound (Theorem 14)** and **Lookahead bound (Lemma 2)**
        $Q.\text{push}(d')$ ▷ Add d' to the queue
      end if
    end for
  end for
end while
$(d^*, R^*) \leftarrow (d^c, R^c)$ ▷ Identify provably optimal tree and objective

Incremental Computation

During the execution of our algorithm, for each tree d, we compute the lower bound $b(d_{un}, \mathbf{x}, \mathbf{y})$ of the tree based on its unchanged leaves d_{un} and the corresponding objective $R(d, \mathbf{x}, \mathbf{y})$ of the tree. Given the hierarchical nature of the parent-children relationship, we *incrementally* compute the objective function and the lower bound throughout the brand-and-bound execution of the algorithm. Together, these ideas save >97% execution time.

Experiments

- Accuracy and optimality

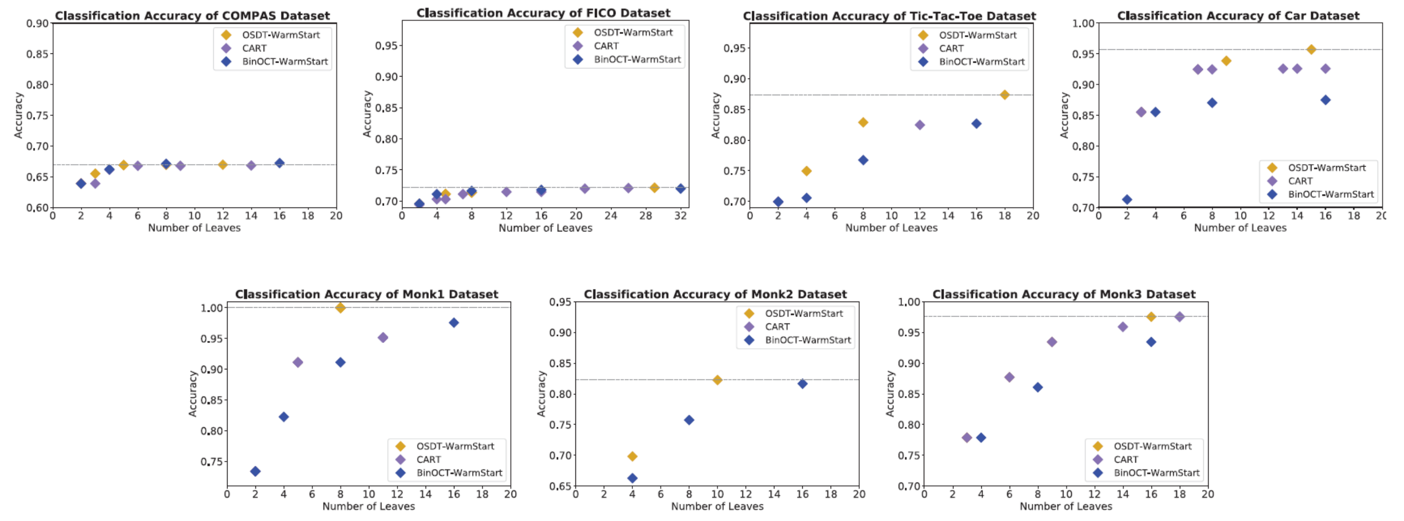


Figure: Training accuracy of OSDT, CART, BinOCT on different data (time limit: 30min). Horizontal lines indicate the accuracy of the best OSDT tree. On most datasets, all trees of BinOCT and CART are below this line.

- Convergence

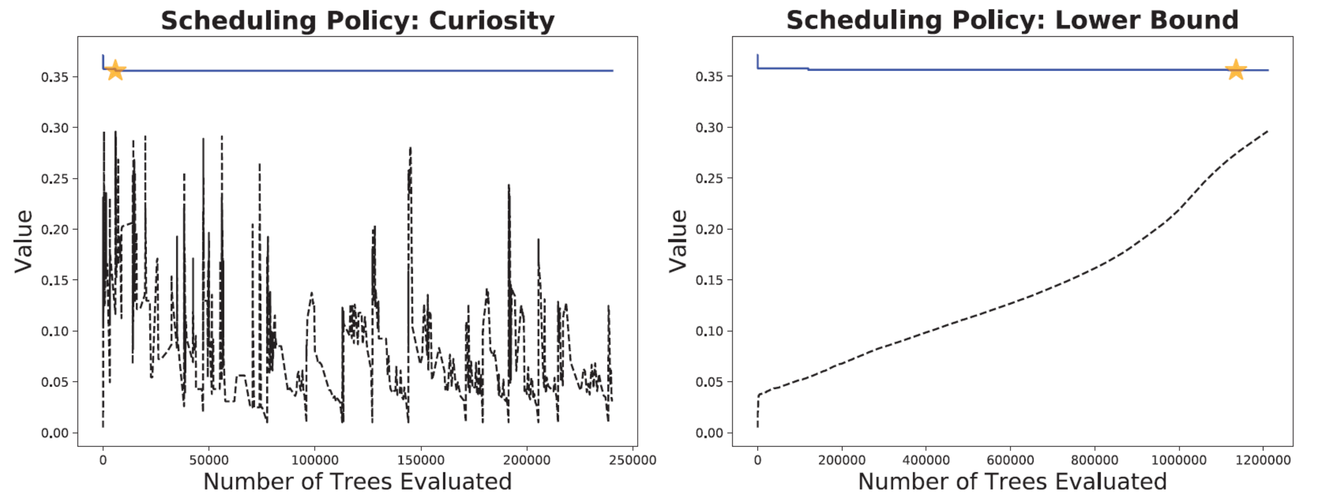


Figure: Example OSDT execution traces (COMPAS Dataset, $\lambda = 0.005$). Lines are the objective value and dashes are the lower bound for OSDT. For each scheduling policy, we mark the time to optimum and the optimal objective value using a star.

- Scalability

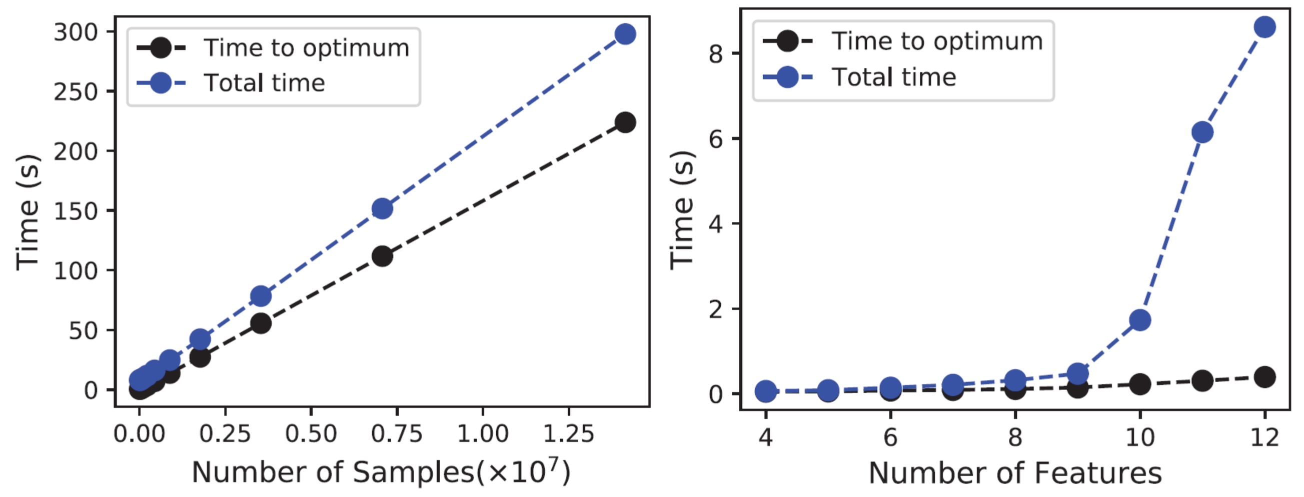


Figure: Scalability with respect to number of samples and number of features using (multiples of) the ProPublica data set. ($\lambda = 0.005$).

Sample Trees

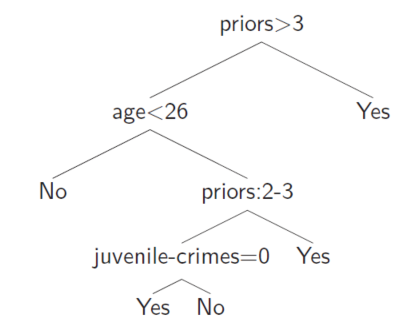
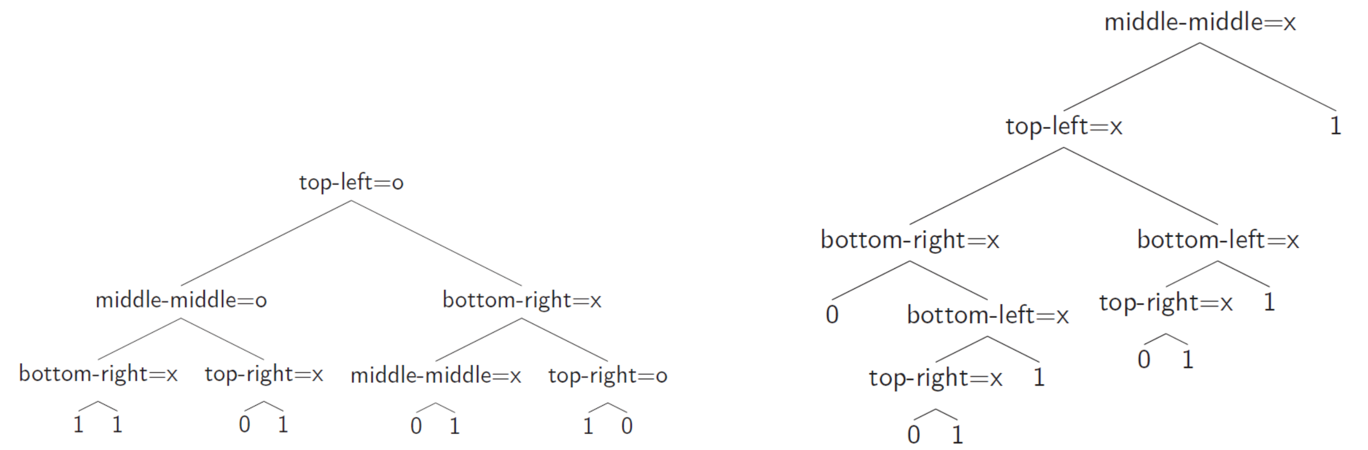
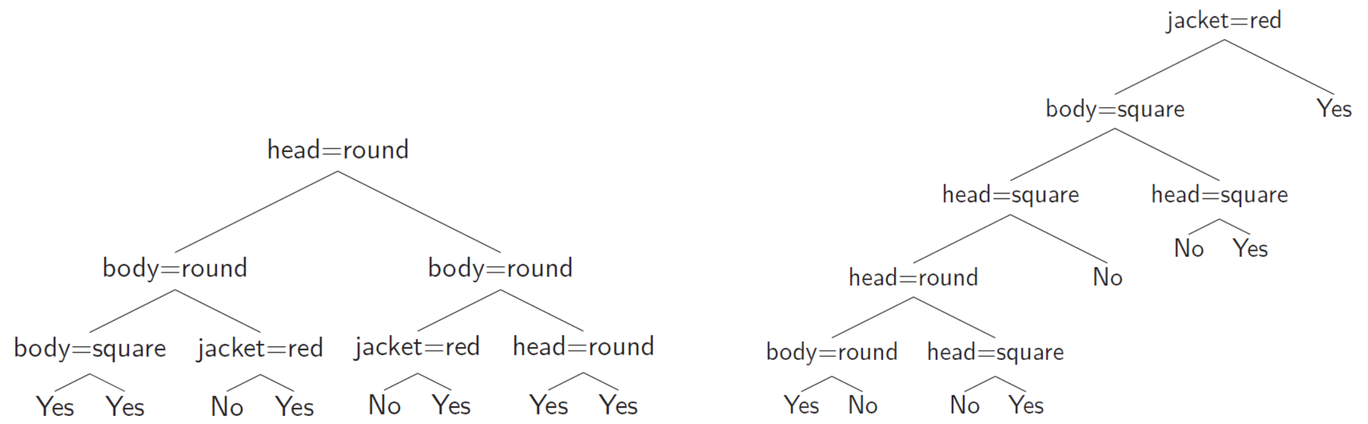


Figure: The optimal decision tree generated by OSDT on COMPAS dataset. ($\lambda = 0.005$)



(a) BinOCT (accuracy: 76.722%) | (b) OSDT (accuracy: 82.881%)

Figure: The decision tree generated by BinOCT and OSDT on the Tic-Tac-Toe data. Trees of BinOCT must be complete binary trees, while OSDT can generate trees of any shape.



(a) BinOCT (accuracy: 91.129%) | (b) OSDT (accuracy: 100%)

Figure: The decision tree generated by BinOCT and OSDT on Monk1 dataset. The tree generated by BinOCT includes useless splits, while OSDT can avoid this problem.

Paper and Code

- Paper: https://arxiv.org/abs/1904.12847
- Code: https://github.com/xiyanghu/OSDT