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.

Overview

- Decision Trees: Extremely popular form for interpretable ML models since the 1980's.
- Existing algorithms use greedy splitting and pruning, providing no guarantee of 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 its leaves.
- A leaf, p_k, is the classification rule of the path from the root to leaf k.
- Let H be the number of leaves in a tree and $K <= H$ be the number of leaves that will not be split.
- 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:

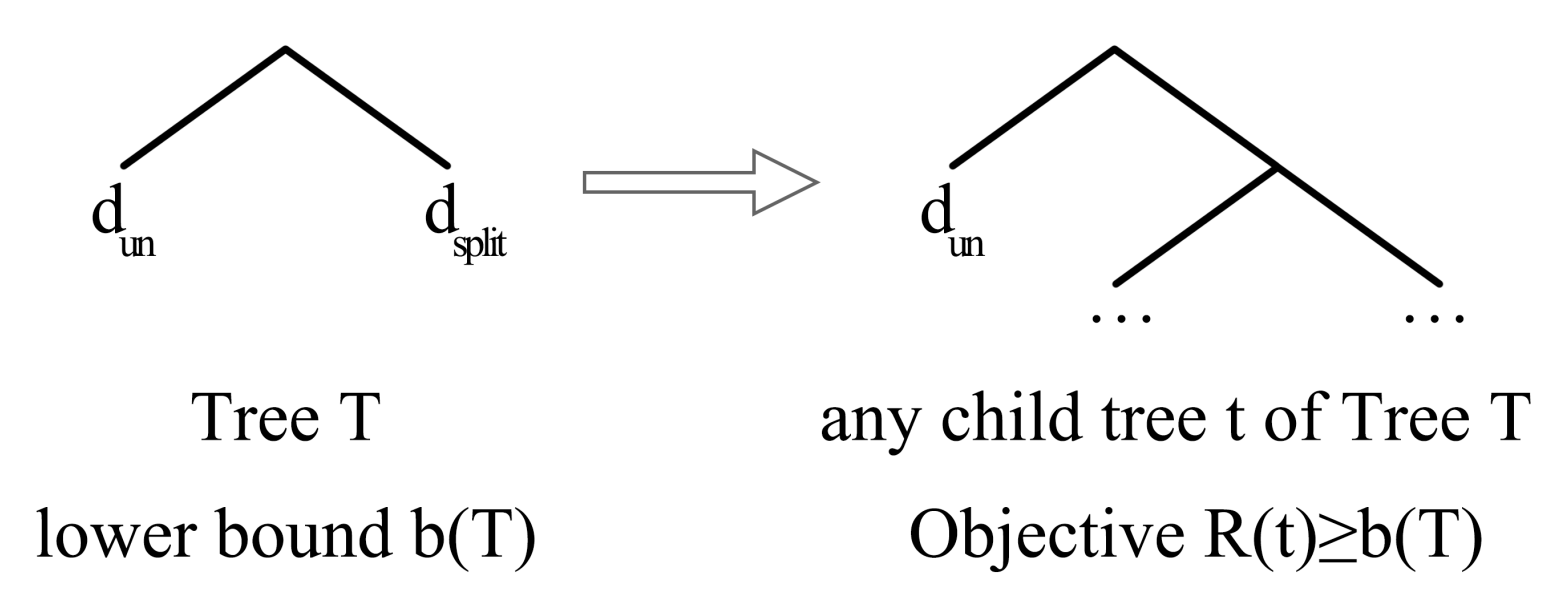
$$R(d, \mathbf{x}, \mathbf{y}) = \ell(d, \mathbf{x}, \mathbf{y}) + \lambda H(d). \qquad (1)$$

where $R(d, \mathbf{x}, \mathbf{y})$ is a regularized empirical risk, $H(d)$ is the number of leaves in the tree d, and 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

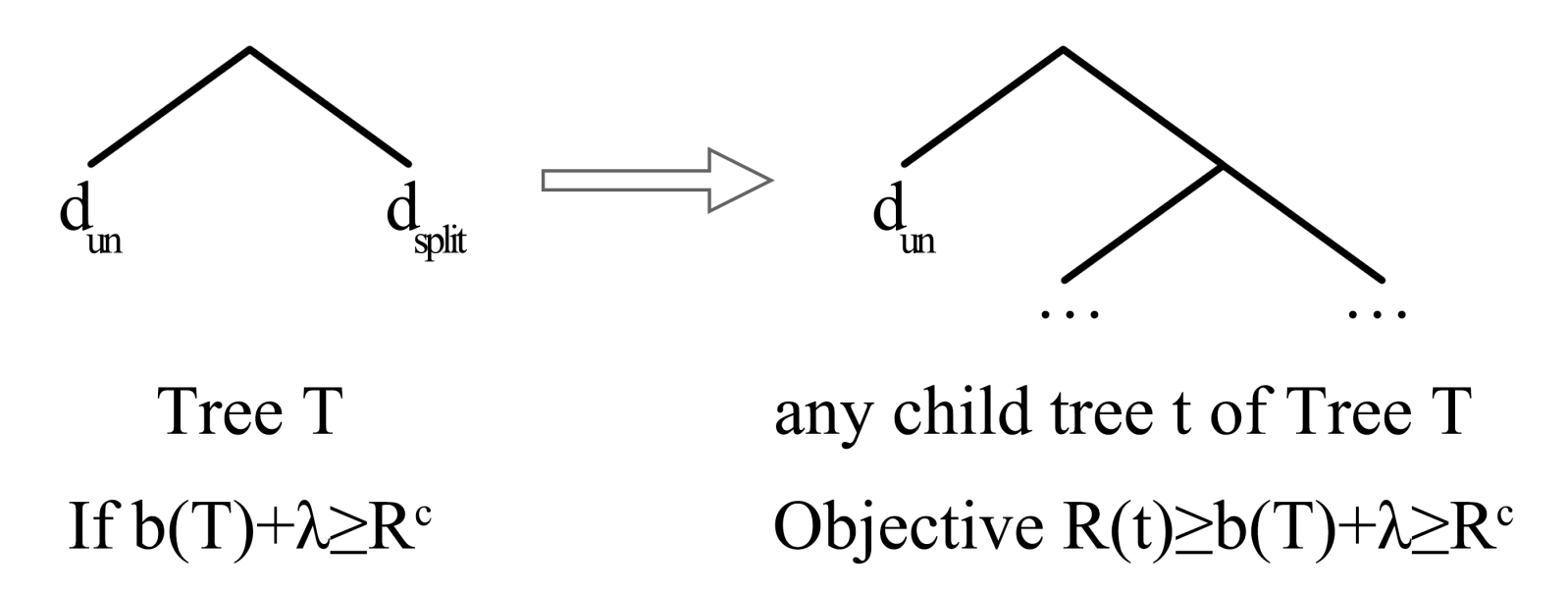
We minimize the objective function based on a branch-and-bound framework. We prove a series of useful bounds that work together to eliminate a large part of the search space.

Hierarchical objective lower bound

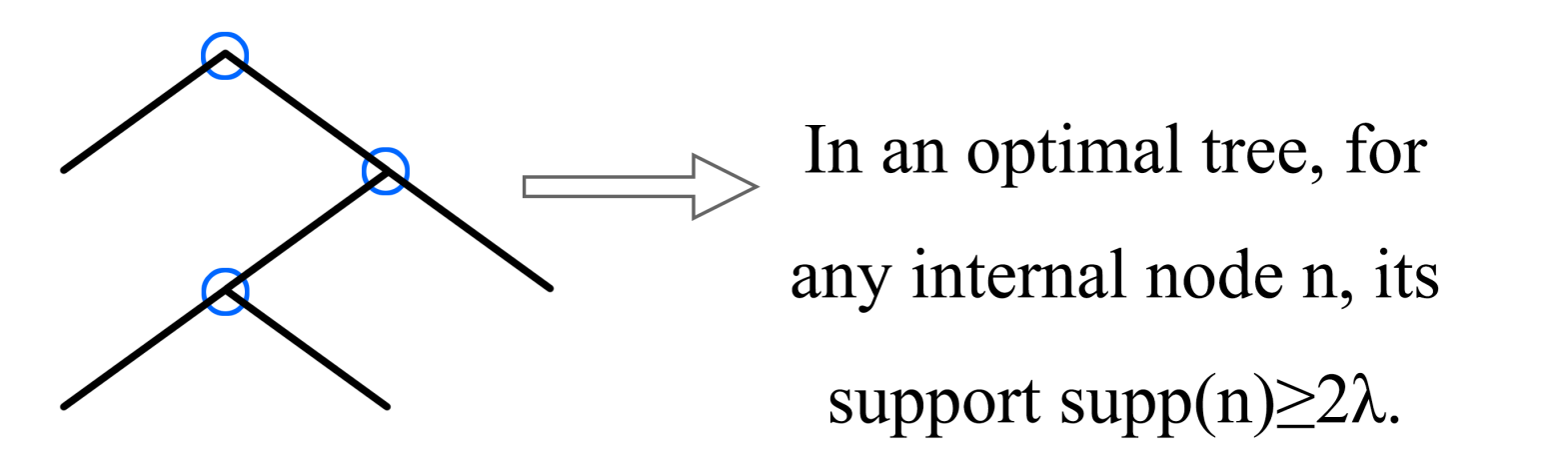


Optimization Framework Cont'd

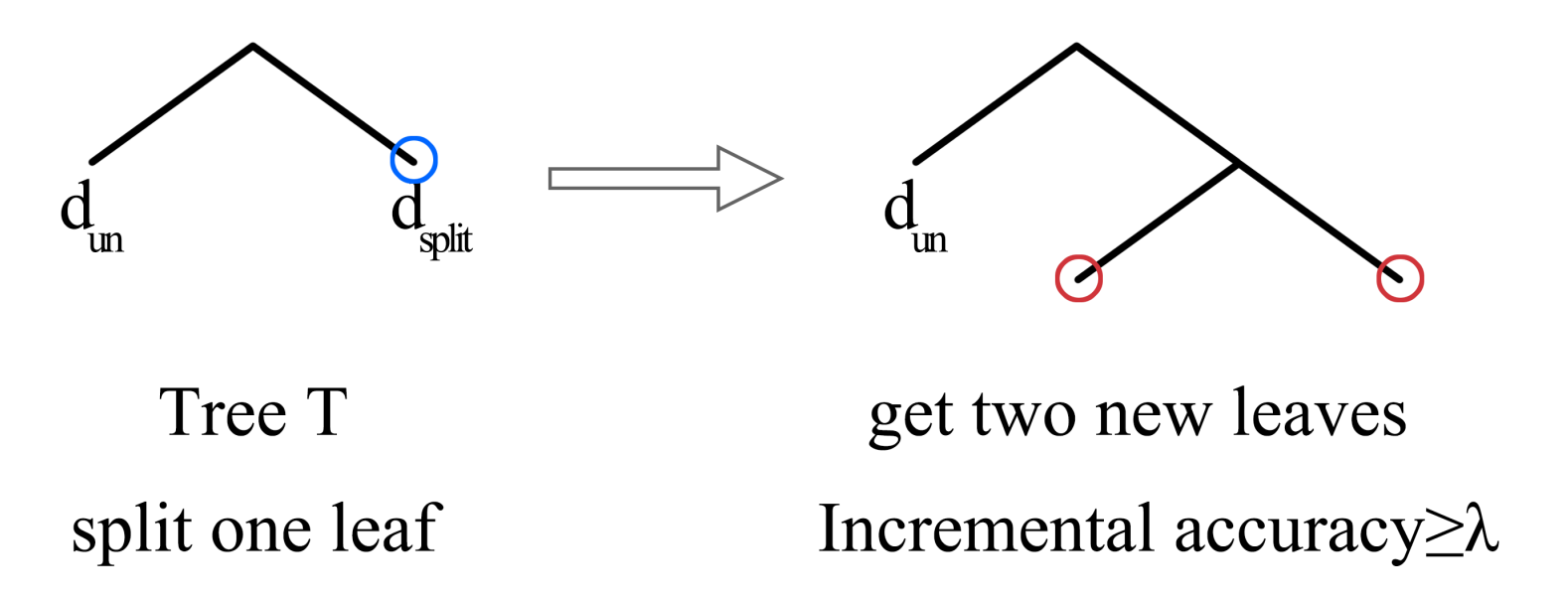
Objective lower bound with one-step lookahead



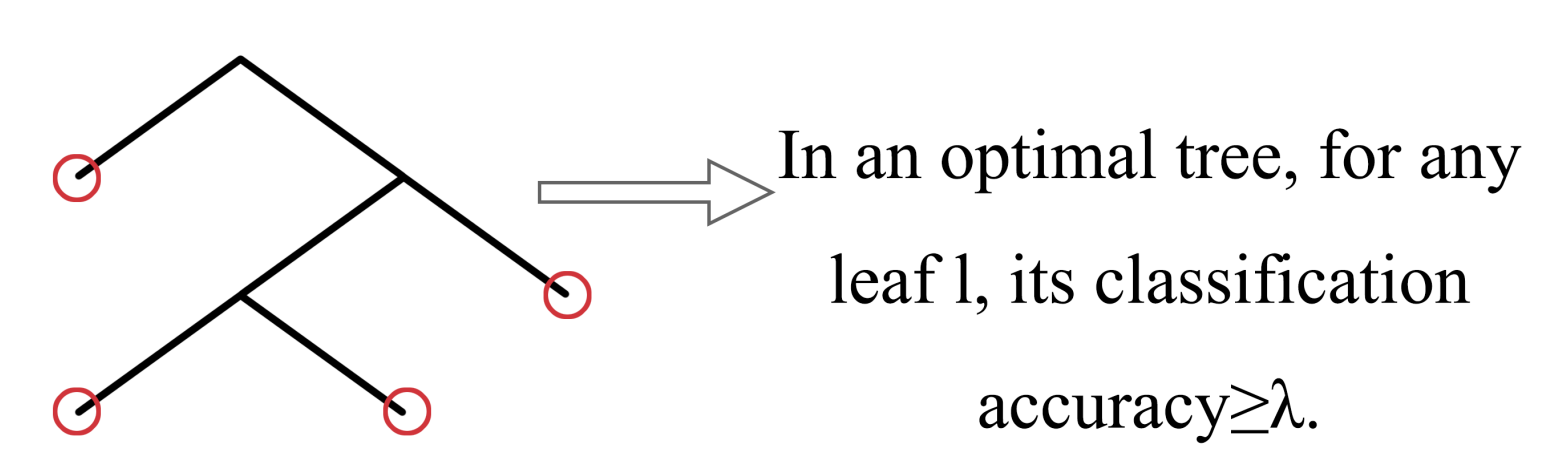
Lower bound on node support



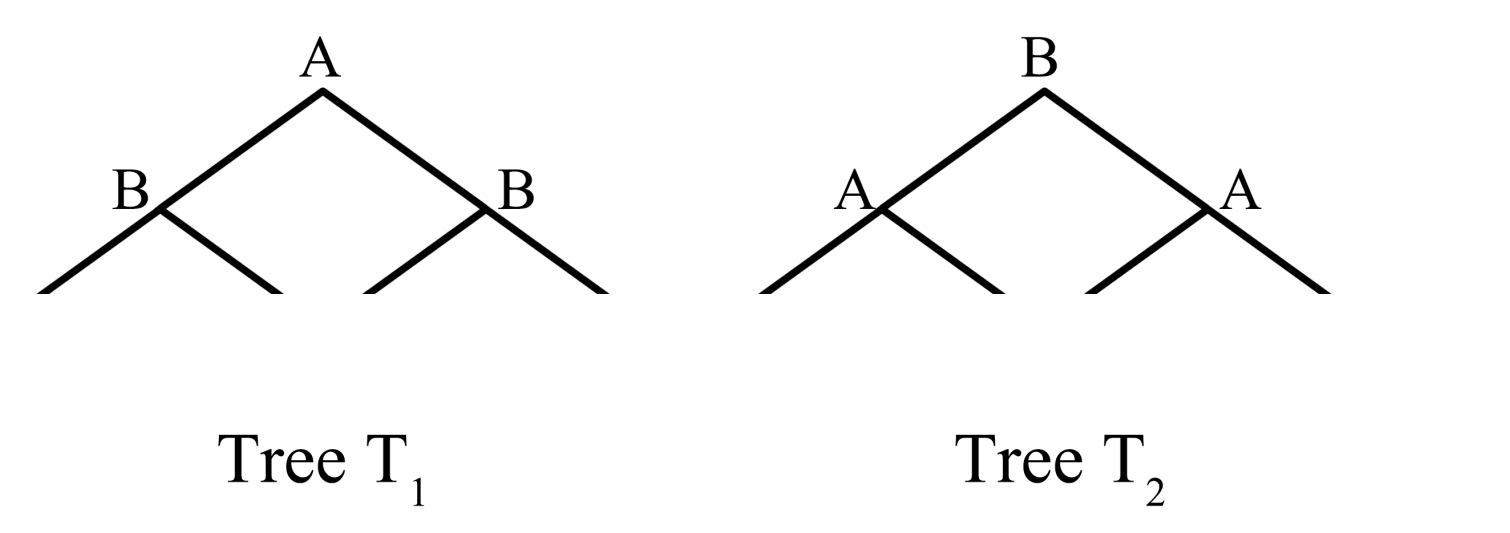
Lower bound on incremental classification accuracy



Leaf accurate support bound

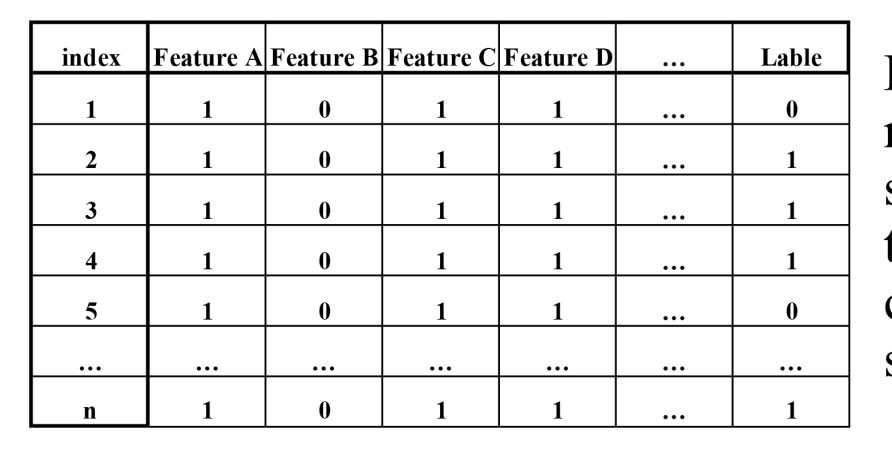


Leaf permutation bound



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:

- $\ell(d, \mathbf{x}, \mathbf{y}) \equiv \ell_p(d_{un}, \delta_{un}, \mathbf{x}, \mathbf{y}) + \ell_p(d_{\text{split}}, \delta_{\text{split}}, \mathbf{x}, \mathbf{y})$, 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 $\delta_{\text{split}} = (\hat{y}_{K+1}^{(\text{leaf})}, \ldots, \hat{y}_H^{(\text{leaf})})$;
- $\ell_p(d_{un}, \delta_{un}, \mathbf{x}, \mathbf{y}) = \frac{1}{N} \Sigma_{n=1}^{N} \Sigma_{k=1}^{K} \text{cap}(x_n, p_k) \wedge \mathbb{1}[\hat{y}_k^{(\text{leaf})} \neq y_n]$ is the proportion of data in the unchanged leaves that are misclassified;
- $\ell_p(d_{\text{split}}, \delta_{\text{split}}, \mathbf{x}, \mathbf{y}) = \frac{1}{N} \Sigma_{n=1}^{N} \Sigma_{k=K+1}^{H} \text{cap}(x_n, p_k) \wedge \mathbb{1}[\hat{y}_k^{(\text{leaf})} \neq y_n]$ 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,

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

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 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 $(\mathbf{x}, \mathbf{y}) = \{(x_n, y_n)\}_{n=1}^{N}$, initial best known tree d^0 with objective $R^0 = R(d^0, \mathbf{x}, \mathbf{y})$; d^0 could be obtained as output from another (approximate) algorithm, otherwise, $(d^0, R^0) = (\text{null}, 1)$ provides reasonable default values. The initial value of δ_{split} is the majority label of the whole dataset.
Output: Provably optimal decision tree d^* with minimum objective R^*

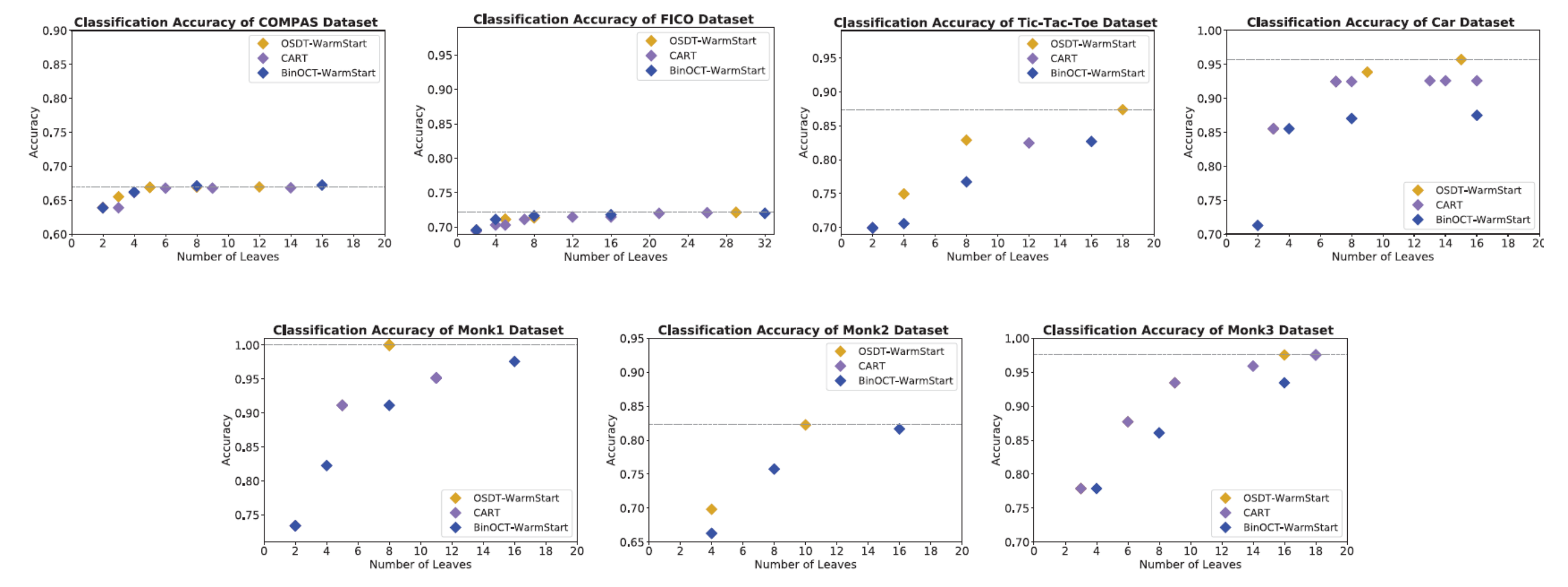
```
$(d^c, R^c) \leftarrow (d^0, R^0)$                                   ▷ Initialize best tree and objective
$Q \leftarrow$ queue( [ ((), (), (), $\delta_{\text{split}}$, 0, 0) ] )          ▷ Initialize queue with empty tree
while $Q$ not empty do                                          ▷ Stop when queue is empty
    $d = (d_{un}, \delta_{un}, d_{\text{split}}, \delta_{\text{split}}, K, H) \leftarrow Q$.pop( )      ▷ Remove tree $d$ from the queue
    if $b(d_{un}, \mathbf{x}, \mathbf{y}) < R^c$ then     ▷ Bound: Hierarchical objective lower bound
        $R \leftarrow R(d, \mathbf{x}, \mathbf{y})$                                 ▷ Compute objective of tree $d$
        if $R < R^c$ then                               ▷ Update best tree and objective
            $(d^c, R^c) \leftarrow (d, R)$
        end if
        for every possible combination of features to split $d_{\text{split}}$ do
                                                    ▷ Branch: Enqueue $d_{un}$'s children
            split $d_{\text{split}}$ and get new leaves $d_{\text{new}}$
            for each possible subset $d'_{\text{split}}$ of $d_{\text{new}}$ do
                $d'_{un} = d_{un} \cup (d_{\text{new}} \setminus d'_{\text{split}})$
                $Q$.push( $(d'_{un}, \delta'_{un}, d'_{\text{split}}, \delta'_{\text{split}}, K', H')$ )
            end for
        end for
    end if
end while
$(d^*, R^*) \leftarrow (d^c, R^c)$                              ▷ Identify provably optimal solution
```

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.

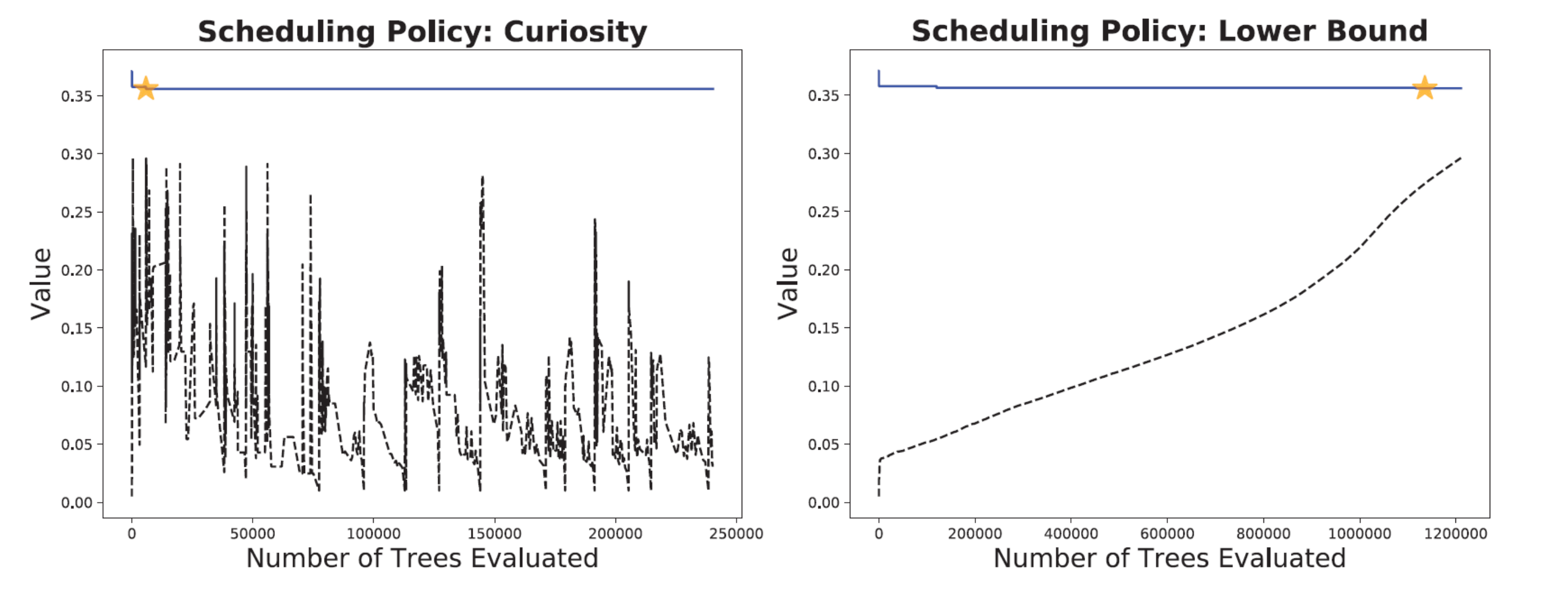
Experiments

- Accuracy and optimality



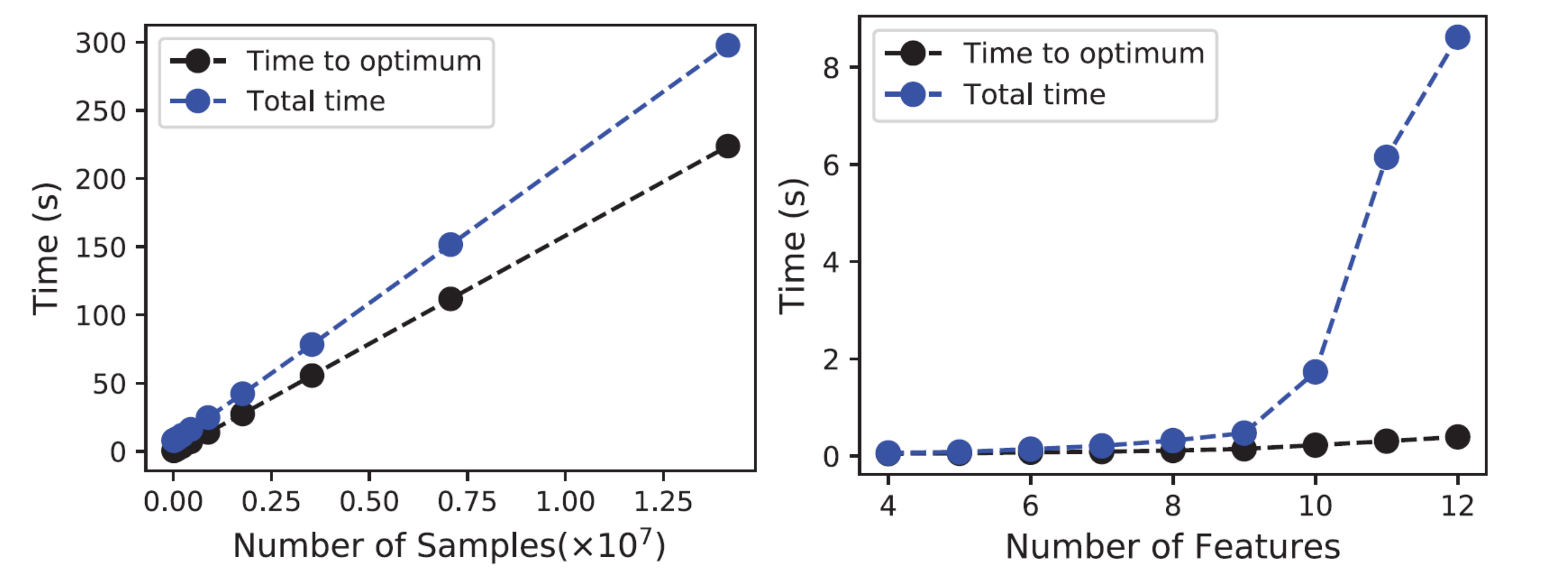
Training accuracy of OSDT, CART, BinOCT on different data (time limit: 30min). Horizontal lines indicate the accuracy of the best OSDT tree. On most datasets, all trees of BinOCT and CART are below this line.

- Convergence



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



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

Sample Trees

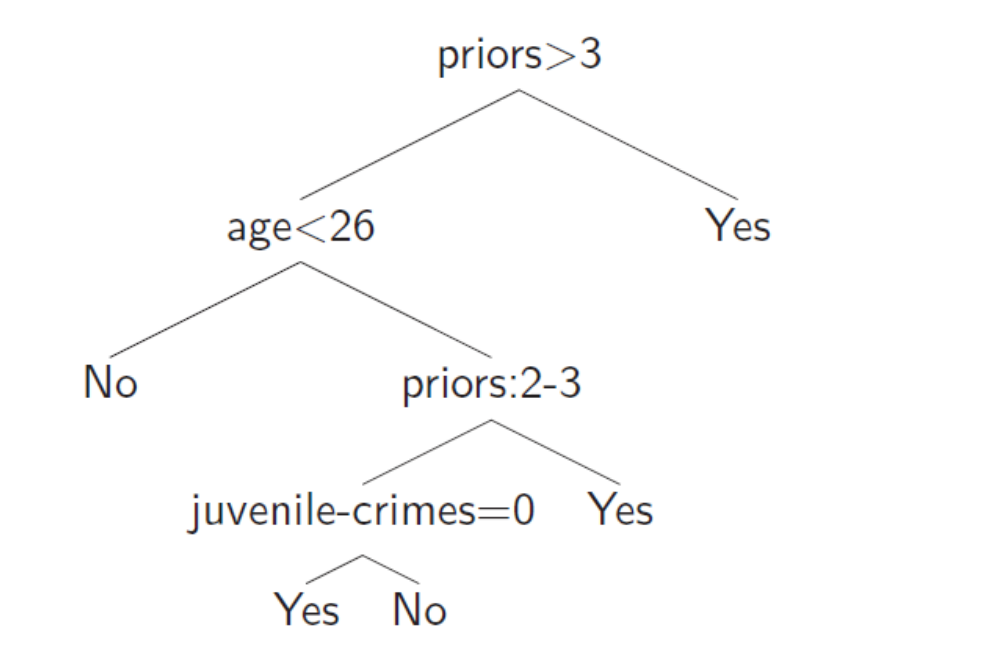
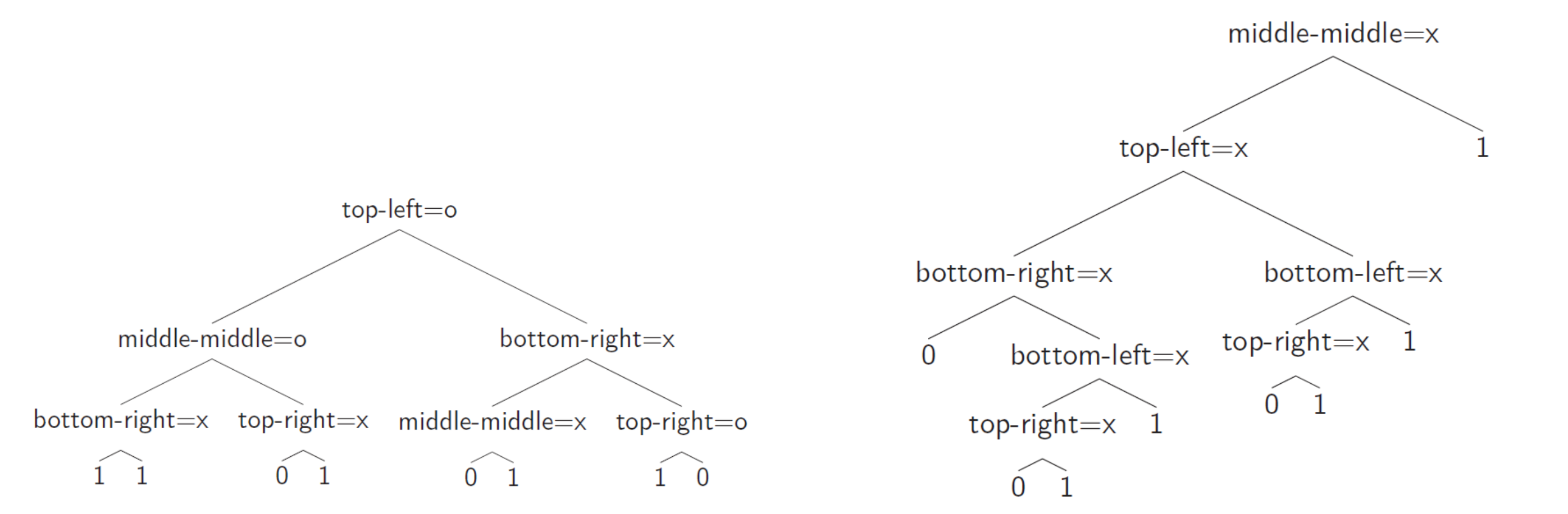
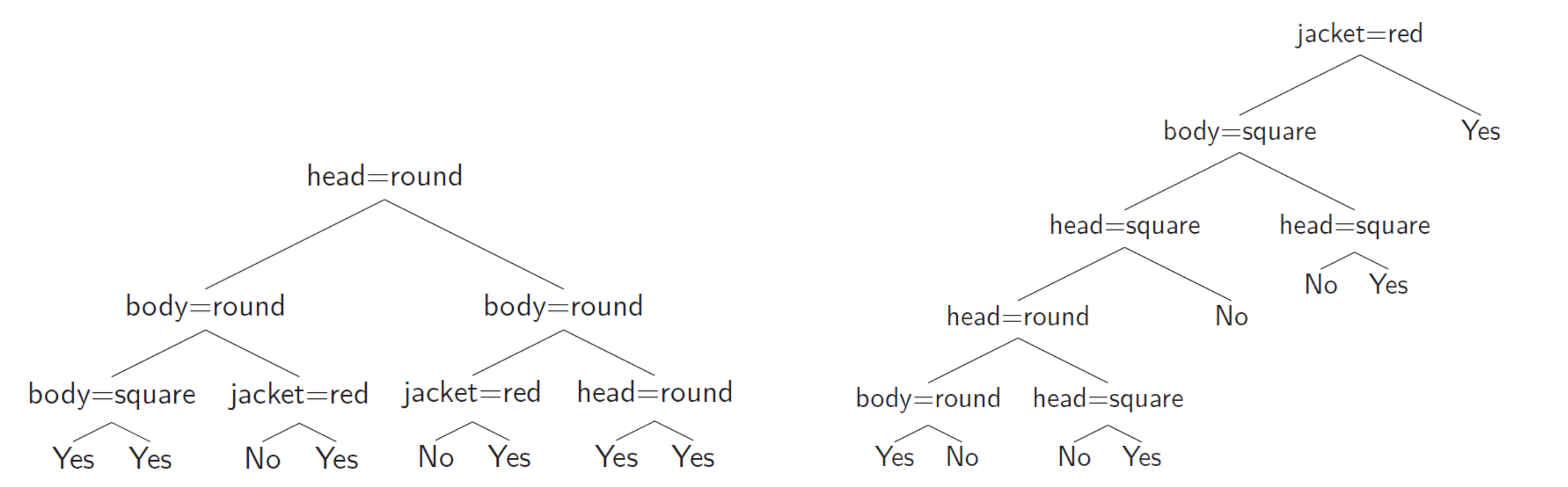


Figure: The optimal decision tree generated by OSDT on COMPAS dataset. ($\lambda = 0.005$)



(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