A. Comparison Between Decision Tree Methods

Attributes	GOSDT	DL8	DL8.5	BinOCT	DTC	CART
Built-in Preprocessor	Yes	No	No	Yes	No	No
Preprocessing Strategy	All values	Weka Discretization	Bucketization	Bucketization	None	None
Preprocessing Preserves Optimality?	Yes	No	No	No	N/A	N/A
Optimization Strategy	DPB	DP	DPB	ILP	Greedy Search	Greedy Search
Optimization Preserves Optimality?	Yes	Yes	Yes	Yes	No	No
Applies Hierarchical Upper bound to Reduce Search Space?	Yes	No	Yes	N/A	N/A	N/A
Uses Support Set Node Identifiers?	Yes	No	No	N/A	N/A	N/A
Can use Multiple Cores?	Yes	No	No	Yes	No	No
Can prune using updates from partial evaluation of subproblem?	Yes	No	No	Depends on (generic) solver	N/A	N/A
Strategy for preventing overfitting	Penalize Leaves	Structural Constraints	Structural Constraints	Structural Constraints	MDL Criteria	Structural Constraints
Can we modify this to use regularization?	N/A	Yes	Yes	Yes	N/A	No
Does it address class imbalance?	Yes	Maybe	Maybe	Maybe	Maybe	Maybe

Table 1. Comparison of GOSDT, DL8 (Nijssen & Fromont, 2007), DL8.5 (Nijssen et al., 2020), BinOCT (Verwer & Zhang, 2019), DTC (Garofalakis et al., 2003), and CART (Breiman et al., 1984). Green is a comparative advantage. Red is a comparative disadvantage. Blue highlights dynamic programming-based methods. White is neutral.

B. Objectives and Their Lower Bounds for Arbitrary Monotonic Losses

Before deriving the bounds for arbitrary monotonic losses, we first introduce some notation. As we know, a leaf set $d = (l_1, l_2, ..., l_{H_d})$ contains H_d distinct leaves, where l_i is the classification rule of the leaf i. If a leaf is labeled, then $y_i^{(\text{leaf})}$ is the label prediction for all data in leaf i. Therefore, a labeled partially-grown tree d with the leaf set $d = (l_1, l_2, ..., l_{H_d})$ could be rewritten as $d = (d_{\text{fix}}, \delta_{\text{fix}}, d_{\text{split}}, \delta_{\text{split}}, K, H_d)$, where $d_{\text{fix}} = (l_1, l_2, ..., l_K)$ is a set of K fixed leaves that are not permitted to be further split, $\delta_{\text{fix}} = (y_1^{(\text{leaf})}, y_2^{(\text{leaf})}, ..., y_K^{(\text{leaf})}) \in \{0,1\}^K$ are the predicted labels for leaves d_{fix}, $d_{\text{split}} = (l_{K+1}, l_{K+2}..., l_{H_d})$ is the set of $H_d - K$ leaves that can be further split, and $\delta_{\text{split}} = (y_{K+1}^{(\text{leaf})}, y_{K+2}^{(\text{leaf})}, ..., y_{H_d}^{(\text{leaf})}) \in \{0,1\}^K$ are the predicted labels for leaves d_{split}.

B.1. Hierarchical objective lower bound for arbitrary monotonic losses

Theorem B.1. *(Hierarchical objective lower bound for arbitrary monotonic losses) Let loss function $\ell(d, \mathbf{x}, \mathbf{y})$ be monotonically increasing in FP and FN. We now change notation of the loss to be only a function of these two quantities, written now as $\tilde{\ell}(FP, FN)$. Let $d = (d_{\text{fix}}, \delta_{\text{fix}}, d_{\text{split}}, \delta_{\text{split}}, K, H)$ be a labeled tree with fixed leaves d_{fix}, and let FP_{fix} and FN_{fix} be the false positives and false negatives of d_{fix}. Define the lower bound to the risk $R(d, \mathbf{x}, \mathbf{y})$ as follows (taking the lower bound of the split terms to be 0):*

$$R(d, \mathbf{x}, \mathbf{y}) \geq b(d_{\text{fix}}, \mathbf{x}, \mathbf{y}) = \ell(d_{\text{fix}}, \mathbf{x}, \mathbf{y}) + \lambda H = \tilde{\ell}(FP_{\text{fix}}, FN_{\text{fix}}) + \lambda H.$$

Let $d' = (d'_{\text{fix}}, \delta'_{\text{fix}}, d'_{\text{split}}, \delta'_{\text{split}}, K', H') \in \sigma(d)$ be any child tree of d such that its fixed leaves d'_{fix} contain d_{fix} and $K' > K$ and $H' > H$. Then, $b(d_{\text{fix}}, \mathbf{x}, \mathbf{y}) \leqslant R(d', \mathbf{x}, \mathbf{y})$.

The importance of this result is that the lower bound works for all (allowed) child trees of d. Thus, if d can be excluded via the lower bound, then all of its children can too.

Proof. Let FP_{fix} and FN_{fix} be the false positives and false negatives within leaves of d_{fix}, and let FP_{split} and FN_{split} be the false positives and false negatives within leaves of d_{split}. Similarly, denote $FP_{\text{fix}'}$ and $FN_{\text{fix}'}$ as the false positives and false negatives of d'_{fix} and let $FP_{\text{split}'}$ and $FN_{\text{split}'}$ be the false positives and false negatives of d'_{split}. Since the leaves are mutually exclusive, $FP_d = FP_{\text{fix}} + FP_{\text{split}}$ and $FN_d = FN_{\text{fix}} + FN_{\text{split}}$. Moreover, since $\tilde{\ell}(FP, FN)$ is monotonically increasing in FP and FN, we have:

$$R(d, \mathbf{x}, \mathbf{y}) = \tilde{\ell}(FP_d, FN_d) + \lambda H \geqslant \tilde{\ell}(FP_{\text{fix}}, FN_{\text{fix}}) + \lambda H = b(d_{\text{fix}}, \mathbf{x}, \mathbf{y}). \tag{11}$$

Similarly, $R(d', \mathbf{x}, \mathbf{y}) \geqslant b(d'_{\text{fix}}, \mathbf{x}, \mathbf{y})$. Since $d_{\text{fix}} \subseteq d_{\text{fix}'}$, $FP_{\text{fix}'} \geqslant FP_{\text{fix}}$ and $FN_{\text{fix}'} \geqslant FN_{\text{fix}}$, thus $\tilde{\ell}(FP_{\text{fix}'}, FN_{\text{fix}'}) \geqslant \tilde{\ell}(FP_{\text{fix}}, FN_{\text{fix}})$. Combined with $H' > H$, we have:

$$b(d_{\text{fix}}, \mathbf{x}, \mathbf{y}) = \tilde{\ell}(FP_{\text{fix}}, FN_{\text{fix}}) + \lambda H \leqslant \tilde{\ell}(FP_{\text{fix}'}, FN_{\text{fix}'}) + \lambda H' = b(d'_{\text{fix}}, \mathbf{x}, \mathbf{y}) \leqslant R(d', \mathbf{x}, \mathbf{y}). \tag{12}$$

□

Let us move to the next bound, which is the one-step lookahead. It relies on comparing the best current objective we have seen so far, denoted R^c, to the lower bound.

Theorem B.2. *(Objective lower bound with one-step lookahead) Let d be a H-leaf tree with K leaves fixed and let R^c be the current best objective. If $b(d_{\text{fix}}, \mathbf{x}, \mathbf{y}) + \lambda \geqslant R^c$, then for any child tree $d' \in \sigma(d)$, its fixed leaves d'_{fix} include d_{fix} and $H' > H$. It follows that $R(d', \mathbf{x}, \mathbf{y}) \geqslant R^c$.*

Proof. According to definition of the objective lower bound,

$$\begin{aligned} R(d', \mathbf{x}, \mathbf{y}) \geqslant b(d'_{\text{fix}}, \mathbf{x}, \mathbf{y}) &= \tilde{\ell}(FP_{\text{fix}'}, FN_{\text{fix}'}) + \lambda H' \\ &= \tilde{\ell}(FP_{\text{fix}'}, FN_{\text{fix}'}) + \lambda H + \lambda(H' - H) \\ &\geqslant b(d_{\text{fix}}, \mathbf{x}, \mathbf{y}) + \lambda \geqslant R^c, \end{aligned} \tag{13}$$

where on the last line we used that since H' and H are both integers, then $H' - H \geq 1$. □

According to this bound, even though we might have a tree d whose fixed leaves d_{fix} obeys lower bound $b(d_{\text{fix}}, \mathbf{x}, \mathbf{y}) \leqslant R^c$, its child trees may still all be guaranteed to be suboptimal: if $b(d_{\text{fix}}, \mathbf{x}, \mathbf{y}) + \lambda \geqslant R^c$, none of its child trees can ever be an optimal tree.

Theorem B.3. *(Hierarchical objective lower bound for sub-trees and additive losses) Let loss functions $\ell(d, \mathbf{x}, \mathbf{y})$ be monotonically increasing in FP and FN, and let the loss of a tree d be the sum of the losses of the leaves. Let R^c be the current best objective. Let d be a tree such that the root node is split by a feature, where two sub-trees d_{left} and d_{right} are generated with H_{left} leaves for d_{left} and H_{right} leaves for d_{right}. The data captured by the left tree is $(\mathbf{x}_{\text{left}}, \mathbf{y}_{\text{left}})$ and the data captured by the right tree is $(\mathbf{x}_{\text{right}}, \mathbf{y}_{\text{right}})$. Let $b(d_{\text{left}}, \mathbf{x}_{\text{left}}, \mathbf{y}_{\text{left}})$ and $b(d_{\text{right}}, \mathbf{x}_{\text{right}}, \mathbf{y}_{\text{right}})$ be the objective lower bound of the left sub-tree and right sub-tree respectively such that $b(d_{\text{left}}, \mathbf{x}_{\text{left}}, \mathbf{y}_{\text{left}}) \leqslant \ell(d_{\text{left}}, \mathbf{x}_{\text{left}}, \mathbf{y}_{\text{left}}) + \lambda H_{\text{left}}$ and $b(d_{\text{right}}, \mathbf{x}_{\text{right}}, \mathbf{y}_{\text{right}}) \leqslant \ell(d_{\text{right}}, \mathbf{x}_{\text{right}}, \mathbf{y}_{\text{right}}) + \lambda H_{\text{right}}$. If $b(d_{\text{left}}, \mathbf{x}_{\text{left}}, \mathbf{y}_{\text{left}}) > R^c$ or $b(d_{\text{right}}, \mathbf{x}_{\text{right}}, \mathbf{y}_{\text{right}}) > R^c$ or $b(d_{\text{left}}, \mathbf{x}_{\text{left}}, \mathbf{y}_{\text{left}}) + b(d_{\text{right}}, \mathbf{x}_{\text{right}}, \mathbf{y}_{\text{right}}) > R^c$, then the tree d is not the optimal tree.*

This bound is applicable to any tree d, even if part of it has not been constructed yet. That is, if a partially-constructed d's left lower bound or right lower bound, or the sum of left and right lower bounds, exceeds the current best risk R^c, then we do not need to construct d since we have already proven it to be suboptimal from its partial construction.

Proof. $R(d, \mathbf{x}, \mathbf{y}) = \ell(d, \mathbf{x}, \mathbf{y}) + \lambda H = \ell(d_{\text{left}}, \mathbf{x}_{\text{left}}, \mathbf{y}_{\text{left}}) + \ell(d_{\text{right}}, \mathbf{x}_{\text{right}}, \mathbf{y}_{\text{right}}) + \lambda H_{\text{left}} + \lambda H_{\text{right}} \geqslant b(d_{\text{left}}, \mathbf{x}_{\text{left}}, \mathbf{y}_{\text{left}}) + b(d_{\text{right}}, \mathbf{x}_{\text{right}}, \mathbf{y}_{\text{right}})$. If $b(d_{\text{left}}, \mathbf{x}_{\text{left}}, \mathbf{y}_{\text{left}}) > R^c$ or $b(d_{\text{right}}, \mathbf{x}_{\text{right}}, \mathbf{y}_{\text{right}}) > R^c$ or $b(d_{\text{left}}, \mathbf{x}_{\text{left}}, \mathbf{y}_{\text{left}}) + b(d_{\text{right}}, \mathbf{x}_{\text{right}}, \mathbf{y}_{\text{right}}) > R^c$, then $R(d, \mathbf{x}, \mathbf{y}) > R^c$. Therefore, the tree d is not the optimal tree. □

B.2. Upper bound on the number of leaves

Theorem B.4. *(Upper bound on the number of leaves) For a dataset with M features, consider a state space of all trees. Let H be the number of leaves of tree d and let R^c be the current best objective. For any optimal tree $d^* \in \text{argmin}_d R(d, \mathbf{x}, \mathbf{y})$, its number of leaves obeys:*

$$H^* \leqslant \min([R^c/\lambda], 2^M) \tag{14}$$

where λ is the regularization parameter.

Proof. This bound adapts directly from OSDT (Hu et al., 2019), where the proof can be found. □

Theorem B.5. *(Parent-specific upper bound on the number of leaves) Let* $d = (d_{\text{fix}}, \delta_{\text{fix}}, d_{\text{split}}, \delta_{\text{split}}, K, H)$ *be a tree,* $d' = (d'_{\text{fix}}, \delta'_{\text{fix}}, d'_{\text{split}}, \delta'_{\text{split}}, K', H') \in \sigma(d)$ *be any child tree such that* $d_{\text{fix}} \subseteq d'_{\text{fix}}$*, and* R^c *be the current best objective. If* d'_{fix} *has lower bound* $b(d'_{\text{fix}}, \mathbf{x}, \mathbf{y}) < R^c$*, then*

$$H' < \min\left(H + \left[\frac{R^c - b(d_{\text{fix}}, \mathbf{x}, \mathbf{y})}{\lambda}\right], 2^M\right). \tag{15}$$

where λ *is the regularization parameter.*

Proof. This bound adapts directly from OSDT (Hu et al., 2019), where the proof can be found. □

B.3. Incremental Progress Bound to Determine Splitting and Lower Bound on Incremental Progress

In the implementation, Theorem B.6 below is used to check if a leaf node within d_{split} is worth splitting. If the bound is satisfied and the leaf can be further split, then we generate new leaves and Theorem B.7 is applied to check if this split yields new nodes or leaves that are good enough to consider in the future. Let us give an example to show how Theorem B.6 is easier to compute than Theorem B.7. If we are evaluating a potential split on leaf j, Theorem B.6 requires FP_j and FN_j which are the false positives and false negatives for leaf j, but no extra information about the split we are going to make, whereas Theorem B.7 requires that additional information. Let us work with balanced accuracy as the loss function: for Theorem B.6 below, we would need to compute $\tau = \frac{1}{2}(\frac{FN_j}{N^+} + \frac{FP_j}{N^-})$ but for Theorem B.7 below we would need to calculate quantities for the new leaves we would form by splitting j into child leaves i and $i+1$. Namely, we would need FN_i, FN_{i+1}, FP_i, and FP_{i+1} as well.

Theorem B.6. *(Incremental progress bound to determine splitting) Let* $d^* = (d_{\text{fix}}, \delta_{\text{fix}}, d_{\text{split}}, \delta_{\text{split}}, K, H)$ *be any optimal tree with objective* R^**, i.e.,* $d^* \in \text{argmin}_d R(d, \mathbf{x}, \mathbf{y})$*. Consider tree* d' *derived from* d^* *by deleting a pair of leaves* l_i *and* l_{i+1} *and adding their parent leaf* l_j*,* $d' = (l_1, ..., l_{i-1}, l_{i+2}, ..., l_H, l_j)$*. Let* $\tau := \tilde{\ell}(FP_{d'}, FN_{d'}) - \tilde{\ell}(FP_{d'} - FP_{l_j}, FN_{d'} - FN_{l_j})$*. Then,* τ *must be at least* λ*.*

Proof. $\ell(d', \mathbf{x}, \mathbf{y}) = \tilde{\ell}(FP_{d'}, FN_{d'})$ and $\ell(d^*, \mathbf{x}, \mathbf{y}) = \tilde{\ell}(FP_{d'} + FP_{l_i} + FP_{l_{i+1}} - FP_{l_j}, FN_{d'} + FN_{l_i} + FN_{l_{i+1}} - FN_{l_j})$. The difference between $\ell(d^*, \mathbf{x}, \mathbf{y})$ and $\ell(d', \mathbf{x}, \mathbf{y})$ is maximized when l_i and l_{i+1} correctly classify all the captured data. Therefore, τ is the maximal difference between $\ell(d', \mathbf{x}, \mathbf{y})$ and $\ell(d^*, \mathbf{x}, \mathbf{y})$. Since $l(d', \mathbf{x}, \mathbf{y}) - \ell(d^*, \mathbf{x}, \mathbf{y}) \leqslant \tau$, we can get $\ell(d', \mathbf{x}, \mathbf{y}) + \lambda(H-1) \leqslant \ell(d^*, \mathbf{x}, \mathbf{y}) + \lambda(H-1) + \tau$, that is (and remember that d^* is of size H whereas d' is of size $H-1$), $R(d', \mathbf{x}, \mathbf{y}) \leqslant R(d^*, \mathbf{x}, \mathbf{y}) - \lambda + \tau$. Since d^* is optimal with respect to R, $0 \leqslant R(d', \mathbf{x}, \mathbf{y}) - R(d^*, \mathbf{x}, \mathbf{y}) \leqslant -\lambda + \tau$, thus, $\tau \geqslant \lambda$. □

Hence, for a tree d, if any of its internal nodes contributes less than λ in loss, even though $b(d_{\text{fix}}, \mathbf{x}, \mathbf{y}) \leqslant R^*$, it cannot be the optimal tree and none of its child tree could be the optimal tree. Thus, after evaluating tree d, we can prune it.

Theorem B.7. *(Lower bound on incremental progress) Let* $d^* = (d_{\text{fix}}, \delta_{\text{fix}}, d_{\text{split}}, \delta_{\text{split}}, K, H)$ *be any optimal tree with objective* R^**, i.e.,* $d^* \in \text{argmin}_d R(d, \mathbf{x}, \mathbf{y})$*. Let* d^* *have leaves* $d_{\text{fix}} = (l_1, ..., l_H)$ *and* $\delta_{\text{fix}} = (y_1^{(leaf)}, y_2^{(leaf)}, ..., y_H^{(leaf)})$*. Consider tree* d' *derived from* d^* *by deleting a pair of leaves* l_i *and* l_{i+1} *with corresponding labels* y_i^{leaf} *and* y_{i+1}^{leaf} *and adding their parent leaf* l_j *and its label* y_j^{leaf}*. Define* a_i *as the incremental objective of splitting* l_j *to get* l_i, l_{i+1}*:* $a_i := \ell(d', \mathbf{x}, \mathbf{y}) - \ell(d^*, \mathbf{x}, \mathbf{y})$*. In this case,* λ *provides a lower bound s.t.* $a_i \geqslant \lambda$*.*

Proof. Let $d' = (d'_{\text{fix}}, \delta'_{\text{fix}}, d'_{\text{split}}, \delta'_{\text{split}}, K', H')$ be the tree derived from d^* by deleting a pair of leaves l_i and l_{i+1}, and adding their parent leaf l_j. Then,

$$\begin{aligned} R(d', \mathbf{x}, \mathbf{y}) &= \ell(d', \mathbf{x}, \mathbf{y}) + \lambda(H-1) = a_i + l(d^*, \mathbf{x}, \mathbf{y}) + \lambda(H-1) \\ &= a_i + R(d^*, \mathbf{x}, \mathbf{y}) - \lambda. \end{aligned} \tag{16}$$

Since $0 \leqslant R(d', \mathbf{x}, \mathbf{y}) - R(d^*, \mathbf{x}, \mathbf{y})$, then $a_i \geqslant \lambda$. □

In the implementation, we apply both Theorem B.6 and Theorem B.7. If Theorem B.6 is not satisfied, even though $b(d_{\text{fix}}, \mathbf{x}, \mathbf{y}) \leqslant R^*$, it cannot be an optimal tree and none of its child trees could be an optimal tree. In this case, d can be pruned, as we showed before. However, if Theorem B.6 is satisfied, we check Theorem B.7. If Theorem B.7 is not satisfied, then we would need to further split at least one of the two child leaves–either of the new leaves i or $i+1$–in order to obtain a potentially optimal tree.

B.4. Permutation Bound

Theorem B.8. *(Leaf Permutation bound) Let π be any permutation of $\{1, ..., H\}$. Let $d = (d_{\text{fix}}, d_{\text{split}}, K, H)$ and $D = (D_{\text{fix}}, D_{\text{split}}, K, H)$ be trees with leaves $(l_1, ..., l_H)$ and $(l_{\pi(1)}, ..., l_{\pi(H)})$ respectively, i.e., the leaves in D correspond to a permutation of the leaves in d. Then the objective lower bounds of d and D are the same and their child trees correspond to permutations of each other.*

Proof. This bound adapts directly from OSDT (Hu et al., 2019), where the proof can be found. □

Therefore, if two trees have the same leaves, up to a permutation, according to Theorem B.8, one of them can be pruned. This bound is capable of reducing the search space by all future symmetries of trees we have already seen.

B.5. Equivalent Points Bound

As we know, for a tree $d = (d_{\text{fix}}, \delta_{\text{fix}}, d_{\text{split}}, \delta_{\text{split}}, K, H)$, the objective of this tree (and that of its children) is minimized when there are no errors in the split leaves: $FP_{\text{split}} = 0$ and $FN_{\text{split}} = 0$. In that case, the risk is equal to $b(d_{\text{fix}}, \mathbf{x}, \mathbf{y})$. However, if multiple observations captured by a leaf in d_{split} have the same features but different labels, then no tree, including those that extend d_{split}, can correctly classify all of these observations, that is FP_{split} and FN_{split} cannot be zero. In this case, we can apply the equivalent points bound to give a tighter lower bound on the objective.

Let Ω be a set of leaves. *Capture* is an indicator function that equals 1 if x_i falls into one of the leaves in Ω, and 0 otherwise, in which case we say that $\text{cap}(x_i, \Omega) = 1$. We define a set of samples to be equivalent if they have exactly the same feature values. Let e_u be a set of equivalent points and let q_u be the minority class label that minimizes the loss among points in e_u. Note that a dataset consists of multiple sets of equivalent points. Let $\{e_u\}_{u=1}^{U}$ enumerate these sets.

Theorem B.9. *(Equivalent points bound) Let $d = (d_{\text{fix}}, \delta_{\text{fix}}, d_{\text{split}}, \delta_{\text{split}}, K, H)$ be a tree such that $l_k \in d_{\text{fix}}$ for $k \in \{1, ..., K\}$ and $l_k \in d_{\text{split}}$ for $k \in \{K+1, ..., H\}$. For any tree $d' \in \sigma(d)$,*

$$\ell(d', \mathbf{x}, \mathbf{y}) \geqslant \tilde{\ell}(FP_{\text{fix}} + FP_e, FN_{\text{fix}} + FN_e), \ where \tag{17}$$

$$\begin{aligned} FP_e &= \sum_{i=1}^{N} \sum_{u=1}^{U} \sum_{k=K+1}^{H} \text{cap}(x_i, l_k) \wedge \mathbb{1}[y_i = 0] \wedge \mathbb{1}[x_i \in e_u] \mathbb{1}[y_i = q_u] \\ FN_e &= \sum_{i=1}^{N} \sum_{u=1}^{U} \sum_{k=K+1}^{H} \text{cap}(x_i, l_k) \wedge \mathbb{1}[y_i = 1] \wedge \mathbb{1}[x_i \in e_u] \mathbb{1}[y_i = q_u]. \end{aligned} \tag{18}$$

Proof. Since $d' \in \sigma(d)$, $d' = (l'_1, ..., l'_K, l'_{K+1}, ..., l'_{K'}, ..., l_{H'})$ where we have both $l'_k \in d'_{\text{fix}}$ for $k \in \{1, ..., K'\}$, which are the fixed leaves and also $l'_k \in d'_{\text{split}}$ for $k \in \{K'+1, ..., H'\}$. Note that for $k \in \{1, ..., K\}$, $l'_k = l_k$.

Let $\Delta = d'_{\text{fix}} \backslash d_{\text{fix}}$ which are the leaves in d'_{fix} that are not in d_{fix}. Then $\ell(d', \mathbf{x}, \mathbf{y}) = \tilde{\ell}(FP_{d'}, FN_{d'}) = \tilde{\ell}(FP_{\text{fix}} + FP_{\Delta} + FP_{\text{split}'}, FN_{\text{fix}} + FN_{\Delta} + FN_{\text{split}'})$, where FP_{Δ} and FN_{Δ} are false positives and false negatives in d'_{fix} but not d_{fix} and $FP_{\text{split}'}$ and $FN_{\text{split}'}$ are false positives and false negatives in d'_{split}. For tree d', its leaves in Δ are those indexed from K to

K'. Thus, the sum over leaves of d' from K to H' includes leaves from Δ and leaves from d'_{split}.

$$\begin{aligned} FP_\Delta + FP_{\text{split}'} &= \sum_{i=1}^{N}\sum_{u=1}^{U}\sum_{k=K+1}^{H'} \text{cap}(x_i, l'_k) \wedge \mathbb{1}[y_i \neq \hat{y}_k^{(\text{leaf})}] \wedge \mathbb{1}[y_i = 0] \wedge \mathbb{1}[x_i \in e_u] \\ &\geqslant \sum_{i=1}^{N}\sum_{u=1}^{U}\sum_{k=K+1}^{H'} \text{cap}(x_i, l'_k) \wedge \mathbb{1}[y_i = 0] \wedge \mathbb{1}[x_i \in e_u]\mathbb{1}[y_i = q_u] \\ FN_\Delta + FN_{\text{split}'} &= \sum_{i=1}^{N}\sum_{u=1}^{U}\sum_{k=K+1}^{H'} \text{cap}(x_i, l'_k) \wedge \mathbb{1}[y_i \neq \hat{y}_k^{(\text{leaf})}] \wedge \mathbb{1}[y_i = 1] \wedge \mathbb{1}[x_i \in e_u] \\ &\geqslant \sum_{i=1}^{N}\sum_{u=1}^{U}\sum_{k=K+1}^{H'} \text{cap}(x_i, l'_k) \wedge \mathbb{1}[y_i = 1] \wedge \mathbb{1}[x_i \in e_u]\mathbb{1}[y_i = q_u]. \end{aligned} \tag{19}$$

For $i \in \{1, ..., N\}$, the samples in d_{split} are the same ones captured by either Δ or d'_{split}, that is $\sum_{k=K+1}^{H} \text{cap}(x_i, l_k) = \sum_{k=K+1}^{H'} \text{cap}(x_i, l'_k)$. Then

$$FP_\Delta + FP_{\text{split}'} \geqslant \sum_{i=1}^{N}\sum_{u=1}^{U}\sum_{k=K+1}^{H} \text{cap}(x_i, l_k) \wedge \mathbb{1}[y_i = 0] \wedge \mathbb{1}[x_i \in e_u]\mathbb{1}[y_i = q_u] = FP_e. \tag{20}$$

Similarly, $FN_\Delta + FN_{\text{split}'} \geqslant FN_e$. Therefore,

$$\ell(d', \mathbf{x}, \mathbf{y}) = \tilde{\ell}(FP_{\text{fix}} + FP_\Delta + FP_{\text{split}'}, FN_{\text{fix}} + FN_\Delta + FN_{\text{split}'}) \geqslant \tilde{\ell}(FP_{\text{fix}} + FP_e, FN_{\text{fix}} + FN_e). \tag{21}$$

□

B.6. Similar Support Bound

Given two trees that are exactly the same except for one internal node split by different features f_1 and f_2, we can use the similar support bound for pruning.

Theorem B.10. *(Similar support bound) Define $d = (d_{\text{fix}}, \delta_{\text{fix}}, d_{\text{split}}, \delta_{\text{split}}, K, H)$ and $D = (D_{\text{fix}}, \Delta_{\text{fix}}, D_{\text{split}}, \Delta_{\text{split}}, K, H)$ to be two trees that are exactly the same except for one internal node split by different features. Let f_1 and f_2 be the features used to split that node in d and D respectively. Let t_1, t_2 be the left and right sub-trees under the node f_1 in d and let T_1, T_2 be the left and right sub-trees under the node f_2 in D. Let ω be the observations captured by only one of t_1 or T_1, i.e.,*

$$\omega \coloneqq \{i : [\text{cap}(x_i, t_1) \wedge \neg\text{cap}(x_i, T_1) + \neg\text{cap}(x_i, t_1) \wedge \text{cap}(x_i, T_1)]\}. \tag{22}$$

Let $FP_{-\omega}$ and $FN_{-\omega}$ be the false positives and false negatives of samples except ω. The difference between the two trees' objectives is bounded as follows:

$$|R(d, \mathbf{x}, \mathbf{y}) - R(D, \mathbf{x}, \mathbf{y})| \leqslant \gamma, \ \textit{where} \tag{23}$$

$$\gamma := \max_{a \in \{0, \ldots, |\omega|\}} [\tilde{\ell}(FP_{-\omega} + a, FN_{-\omega} + |\omega| - a)] - \tilde{\ell}(FP_{-\omega}, FN_{-\omega}). \tag{24}$$

Then we have

$$\left| \min_{d^+ \in \sigma(d)} R(d^+, \mathbf{x}, \mathbf{y}) - \min_{D^+ \in \sigma(D)} R(D^+, \mathbf{x}, \mathbf{y}) \right| \leqslant \gamma. \tag{25}$$

Proof. The difference between the objectives of d and D is largest when one of them correctly classifies all the data in ω but the other misclassifies all of them. If d classifies all the data corresponding to ω correctly while D misclassifies them,

$$R(d, \mathbf{x}, \mathbf{y}) - R(D, \mathbf{x}, \mathbf{y}) \geqslant \tilde{\ell}(FP_{-\omega}, FN_{-\omega}) - \max_{a \in \{0, \ldots, |\omega|\}} [\tilde{\ell}(FP_{-\omega} + a, FN_{-\omega} + |\omega| - a)] = -\gamma. \tag{26}$$

We can get $R(d, \mathbf{x}, \mathbf{y}) - R(D, \mathbf{x}, \mathbf{y}) \leqslant \gamma$ in the same way. Therefore, $\gamma \geqslant R(d, \mathbf{x}, \mathbf{y}) - R(D, \mathbf{x}, \mathbf{y}) \geqslant -\gamma$.

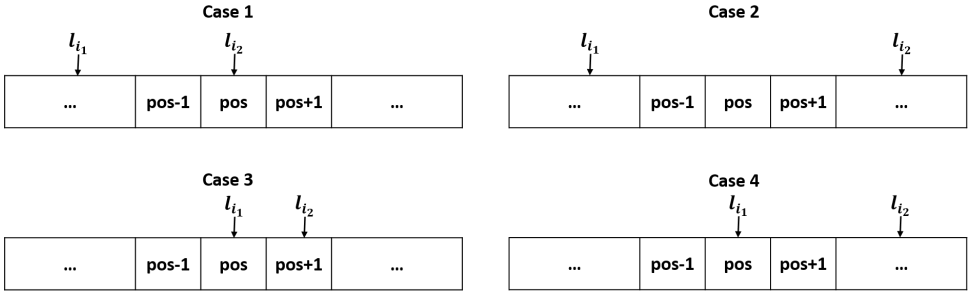


Figure 7. Four cases of positions of l_{i_1} and l_{i_2}

Let d^* be the best child tree of d, i.e., $R(d^*, \mathbf{x}, \mathbf{y}) = \min_{d^+ \in \sigma(d)} R(d^+, \mathbf{x}, \mathbf{y})$. Let $D' \in \sigma(D)$ be its counterpart which is exactly the same except for one internal node split by a different feature. Then, using Equation 26,

$$\min_{d^+ \in \sigma(d)} R(d^+, \mathbf{x}, \mathbf{y}) = R(d^*, \mathbf{x}, \mathbf{y}) \geqslant R(D', \mathbf{x}, \mathbf{y}) - \gamma \geqslant \min_{D^+ \in \sigma(D)} R(D^+, \mathbf{x}, \mathbf{y}) - \gamma. \tag{27}$$

Similarly, using the symmetric counterpart to Equation 26 and the same logic, $\min_{D^+ \in \sigma(D)} R(D^+, \mathbf{x}, \mathbf{y}) + \gamma \geqslant \min_{d^+ \in \sigma(d)} R(d^+, \mathbf{x}, \mathbf{y})$. □

C. Objectives and Their Lower Bounds for Rank Statistics

In this appendix, we provide proofs for Theorem 2.1 and Theorem 2.2, and adapt the Incremental Progress Bound to Determine Splitting and the Equivalent Points Bound for the objective AUC_{ch}. The Upper Bound on the Number of Leaves, Parent-Specific Upper Bound on the Number of Leaves, Lower Bound of Incremental Progress, and Permutation Bound are the same as the bounds in Appendix B. We omit these duplicated proofs here. At the end of this appendix, we define the objective pAUC_{ch} and how we implement the derived bounds for this objective. As a reminder, we use notation $d = (d_{\text{fix}}, r_{\text{fix}}, d_{\text{split}}, r_{\text{split}}, K, H_d)$ to represent tree d.

Lemma C.1. *Let $d = (d_{\text{fix}}, r_{\text{fix}}, d_{\text{split}}, r_{\text{split}}, K, H_d)$ be a tree. The AUC convex hull does not decrease when an impure leaf is split.*

Proof. Let l_i be the impure leaf that we intend to split, where $i \in \{1, ..., H_d\}$. Let n_i^+ be the positive samples in l_i and n_i^- negative samples. Suppose l_i is ranked in position "pos." If the leaf is split once, it will generate two leaves l_{i_1} and l_{i_2} such that $r_{i_1} \geq r_i \geq r_{i_2}$ without loss of generality. Let d' be the tree that consists of the leaf set $(l_1, ..., l_{i-1}, l_{i+1}, ..., l_{H_d}, l_{i_1}, l_{i_2})$. If $r_{i_1} = r_i = r_{i_2}$, then the rank order of leaves (according to the r_i's) will not change, so AUC_{ch} will be unchanged after the split. Otherwise (if the rank order of leaves changes when introducing a child) we can reorder the $H+1$ leaves, leading to the following four cases. For the new leaf set $(l_1, ..., l_{i-1}, l_{i+1}, ..., l_{H_d}, l_{i_1}, l_{i_2})$, either:

1. The rank of l_{i_1} is smaller than pos and the rank of l_{i_2} equals $\text{pos} + 1$ (requires $r_{i_1} > r_i$ and $r_i \geq r_{i_2}$);
2. The rank of l_{i_1} is smaller than pos and the rank of l_{i_2} is larger than $\text{pos} + 1$ (requires $r_{i_1} > r_i$ and $r_i > r_{i_2}$);
3. The rank of l_{i_1} is equal to pos and the rank of l_{i_2} is equal to $\text{pos} + 1$ (requires $r_{i_1} \geq r_i$ and $r_i \geq r_{i_2}$);
4. The rank of l_{i_1} is pos and the rank of l_{i_2} is larger than $\text{pos} + 1$ (requires $r_{i_1} \geq r_i$ and $r_i > r_{i_2}$).

Figure 7 shows four cases of the positions of l_{i_1} and l_{i_2}.

Let us go through these cases in more detail.

For the new leaf set after splitting l_i, namely $(l_1, ..., l_{i-1}, l_{i+1}, ..., l_{H_d}, l_{i_1}, l_{i_2})$, we have that:

1. l_{i_1} has rank smaller than pos (which requires $r_{i_1} > r_i$) and l_{i_2} has rank $\text{pos} + 1$ (which requires $r_i \geq r_{i_2}$).

   Let $A = \{l_{a_1}, l_{a_2}, ...l_{a_U}\}$ be a collection of leaves ranked before l_{i_1} and let $B = \{l_{b_1}, l_{b_2}, ...l_{b_V}\}$ be a collection of leaves ranked after l_{i_1} but before $\text{pos} + 1$. In this case, recalling Equation (2), a change in the AUC_{ch} after splitting on leaf i is due only to a subset of leaves, namely $l_{i_1}, l_{b_1}, ..., l_{b_V}, l_{i_2}$. Then we can compute the change in the AUC_{ch} as

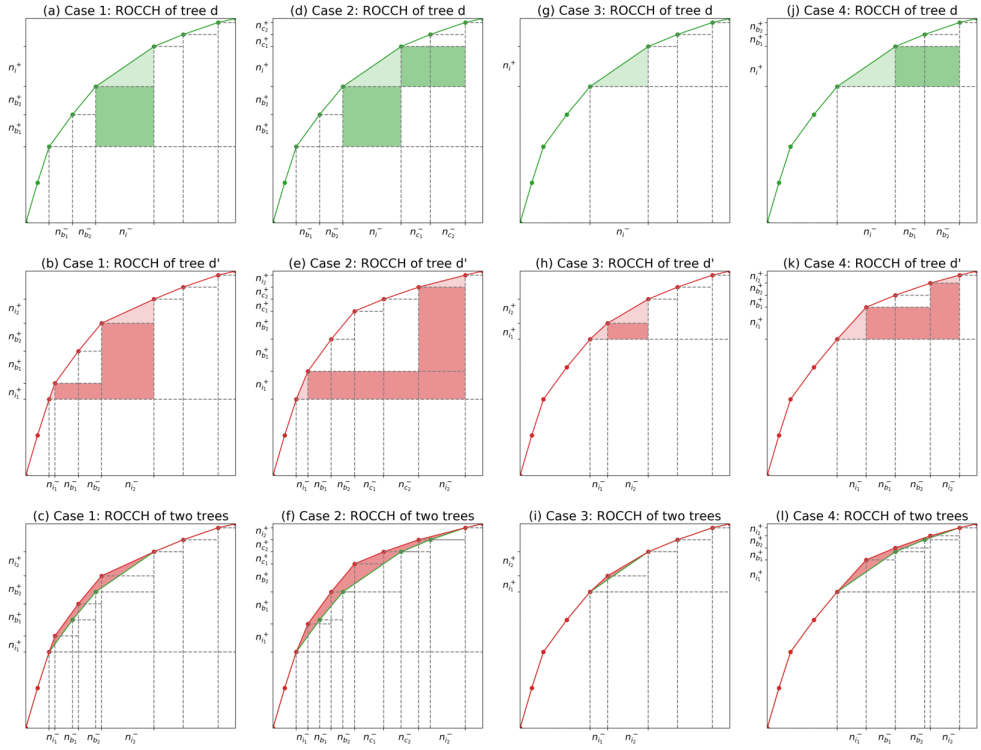


Figure 8. Four different cases of changing the rank orders after introducing a child. Each column represents a case. Subplots in the first row show the ROCCH of tree d and subplots in the second row indicate the ROCCH of tree d' corresponding to the different positions of two child leaves. Subplots in the third row present the change of AUC_{ch} after introducing the child leaves.

follows:

$$\Delta_{\text{AUC}_{\text{ch}}} = \frac{1}{N^+N^-}\left(\frac{n^-_{i_1}n^+_{i_1}}{2} + \left(\sum_{v=1}^{V} n^-_{b_v}\right)n^+_{i_1} + n^-_{i_2}\left[n^+_{i_1} + \left(\sum_{v=1}^{V} n^+_{b_v}\right) + \frac{n^+_{i_2}}{2}\right] - n^-_i\left[\left(\sum_{v=1}^{V} n^+_{b_v}\right) + \frac{n^+_i}{2}\right]\right) \quad (28)$$

To derive the expression for $\Delta_{\text{AUC}_{\text{ch}}}$, we first sum shaded areas of rectangles and triangles under the ROC curves' convex hull for both tree d and its child tree d', and then calculate the difference between the two shaded areas, as indicated in Figure 8 (a-c). This figure shows where each of the terms arises within the $\Delta_{\text{AUC}_{\text{ch}}}$: terms $n^-_{b_v}n^+_{i_1}$ and $n^-_{i_2}[n^+_{i_1} + (\sum_{v=1}^{V} n^+_{b_v})]$ come from the area of rectangles colored in dark pink in Figure 8 (b). Terms $\frac{n^-_{i_1}n^+_{i_1}}{2}$ and $\frac{n^-_{i_2}n^+_{i_2}}{2}$ handle the top triangles colored in light pink. Term $n^-_i(\sum_{v=1}^{V} n^+_{b_v})$ represents the rectangles colored in dark green in Figure 8 (a) and term $\frac{n^-_i n^+_i}{2}$ deals with the triangles colored in light green. Subtracting green shaded areas from red shaded areas, we get $\Delta_{\text{AUC}_{\text{ch}}}$, which is represented by the (remaining) pink area in Figure 8 (c).

Simplifying Equation (28), we get

$$\Delta_{\text{AUC}_{\text{ch}}} = \frac{1}{N^+N^-}\left(n^+_{i_1}\left(\sum_{v=1}^{V} n^-_{b_v}\right) + n^-_{i_2}\left(\sum_{v=1}^{V} n^+_{b_v}\right) - n^-_i\left(\sum_{v=1}^{V} n^+_{b_v}\right) + \frac{n^-_{i_1}n^+_{i_1} + 2n^-_{i_2}n^+_{i_1} + n^-_{i_2}n^+_{i_2} - n^-_i n^+_i}{2}\right). \quad (29)$$

Recall that $n^-_i = n^-_{i_1} + n^-_{i_2}$ and $n^+_i = n^+_{i_1} + n^+_{i_2}$. Then, simplifying,

$$\Delta_{\text{AUC}_{\text{ch}}} = \frac{1}{N^+N^-}\left(n^+_{i_1}\left(\sum_{v=1}^{V} n^-_{b_v}\right) - n^-_{i_1}\left(\sum_{v=1}^{V} n^+_{b_v}\right) + \frac{n^+_{i_1}n^-_{i_2} - n^-_{i_1}n^+_{i_2}}{2}\right). \quad (30)$$

Since $r_{i_1} > r_{b_1} > r_{b_2} > ... > r_{b_V}, \forall v \in \{1, 2, ..., V\}$, $\frac{n^+_{i_1}}{n^+_{i_1}+n^-_{i_1}} > \frac{n^+_{b_v}}{n^+_{b_v}+n^-_{b_v}}$. Then we can get $n^+_{i_1}n^-_{b_v} > n^+_{b_v}n^-_{i_1}$. Hence

$n_{i_1}^+(\sum_{v=1}^V n_{b_v}^-) - n_{i_1}^-(\sum_{v=1}^V n_{b_v}^+) > 0$. Similarly, because $r_{i_1} > r_{i_2}$, $\frac{n_{i_1}^+}{n_{i_1}^+ + n_{i_1}^-} > \frac{n_{i_2}^+}{n_{i_2}^+ + n_{i_2}^-}$. Then $n_{i_1}^+ n_{i_2}^- > n_{i_1}^- n_{i_2}^+$. Therefore, $\Delta_{\text{AUC}_{\text{ch}}} > 0$.

2. l_{i_1} has a ranking smaller than pos (which requires $r_{i_1} > r_i$) and l_{i_2} has a ranking larger than $\text{pos}+1$ (which requires $r_i > r_{i_2}$).

   Let $A = \{l_{a_1}, ...l_{a_U}\}$ be a collection of leaves that ranked before l_{i_1} and $B = \{l_{b_1}, ...l_{b_V}\}$ be a collection of leaves that ranked after l_{i_1} but before $\text{pos}+1$, and $C = \{l_{c_1}, ..., l_{c_W}\}$ be a collection of leaves that ranked after $\text{pos}+1$ but before the rank of l_{i_2}. In this case, the change is caused by $l_{i_1}, l_{b_1}, ..., l_{b_V}, l_{c_1}, ..., l_{c_W}, l_{i_2}$. Then we can compute the change in the AUC_{ch} as follows:

$$\begin{aligned}\Delta_{\text{AUC}_{\text{ch}}} = \frac{1}{N^+N^-}\Bigg(&\frac{n_{i_1}^- n_{i_1}^+}{2} + \Big(\sum_{v=1}^V n_{b_v}^-\Big)n_{i_1}^+ + \Big(\sum_{w=1}^W n_{c_w}^-\Big)n_{i_1}^+ + n_{i_2}^-\Big[n_{i_1}^+ + \Big(\sum_{v=1}^V n_{b_v}^+\Big) + \Big(\sum_{w=1}^W n_{c_w}^+\Big) + \frac{n_{i_2}^+}{2}\Big] \\ &- n_i^-\Big[\Big(\sum_{v=1}^V n_{b_v}^+\Big) + \frac{n_i^+}{2}\Big] - \Big(\sum_{w=1}^W n_{c_w}^-\Big)n_i^+\Bigg).\end{aligned} \tag{31}$$

   Similar to the derivation proposed in case 1, we first sum shaded areas of rectangles and triangles under the ROC curves' convex hull for both tree d and its child tree d' and then calculate the difference between two shaded areas as indicated in Figure 8 (d-f). These three subfigures show where each of the terms arises within the $\Delta_{\text{AUC}_{\text{ch}}}$: terms $n_{b_v}^- n_{i_1}^+$, $n_{c_w}^- n_{i_1}^+$, and $n_{i_2}^-[n_{i_1}^+ + (\sum_{v=1}^V n_{b_v}^+) + (\sum_{w=1}^W n_{c_w}^+)]$ come from the area of rectangles colored in dark pink in Figure 8 (e). Terms $\frac{n_{i_1}^- n_{i_1}^+}{2}$ and $\frac{n_{i_2}^- n_{i_2}^+}{2}$ handle the top triangles colored in light pink. Terms $n_i^- n_{b_v}^+$ and $n_i^+ n_{c_w}^-$ represent the rectangles colored in dark green in Figure 8 (d) and term $\frac{n_i^- n_i^+}{2}$ deals with the triangles colored in light green. Subtracting green shaded areas from red shaded areas, we can get $\Delta_{\text{AUC}_{\text{ch}}}$, which is represented by the light red area in Figure 8 (f).

   Recall that $n_i^- = n_{i_1}^- + n_{i_2}^-$ and $n_i^+ = n_{i_1}^+ + n_{i_2}^+$. Then, simplifying Equation (31), we get

$$\begin{aligned}\Delta_{\text{AUC}_{\text{ch}}} &= \frac{1}{N^+N^-}\Bigg(\Big(\sum_{v=1}^V n_{b_v}^-\Big)n_{i_1}^+ + \Big(\sum_{w=1}^W n_{c_w}^-\Big)n_{i_1}^+ + \Big(\sum_{v=1}^V n_{b_v}^+\Big)n_{i_2}^- + \Big(\sum_{w=1}^W n_{c_w}^+\Big)n_{i_2}^- - \Big(\sum_{v=1}^V n_{b_v}^+\Big)n_i^- \\ &\quad - \Big(\sum_{w=1}^W n_{c_w}^-\Big)n_i^+ + \frac{n_{i_1}^- n_{i_1}^+ + 2n_{i_2}^- n_{i_1}^+ + n_{i_2}^- n_{i_2}^+ - n_i^- n_i^+}{2}\Bigg) \\ &= \frac{1}{N^+N^-}\Bigg(\Big(\sum_{v=1}^V n_{b_v}^-\Big)n_{i_1}^+ - \Big(\sum_{v=1}^V n_{b_v}^+\Big)n_{i_1}^- + \Big(\sum_{w=1}^W n_{c_w}^+\Big)n_{i_2}^- - \Big(\sum_{w=1}^W n_{c_w}^-\Big)n_{i_2}^+ + \frac{n_{i_2}^- n_{i_1}^+ - n_{i_1}^- n_{i_2}^+}{2}\Bigg).\end{aligned} \tag{32}$$

   Since $r_{i_1} > r_{b_1} > ... > r_{b_V}$, $\forall v \in \{1, ..., V\}$, $\frac{n_{i_1}^+}{n_{i_1}^+ + n_{i_1}^-} > \frac{n_{b_v}^+}{n_{b_v}^+ n_{b_v}^-}$. Then we get $n_{i_1}^+ n_{b_v}^- > n_{b_v}^+ n_{i_1}^-$. Thus, $(\sum_{v=1}^V n_{b_v}^-)n_{i_1}^+ - (\sum_{v=1}^V n_{b_v}^+)n_{i_1}^- > 0$. Similarly, since $r_{c_1} > ... > r_{c_W} > r_{i_2}$, $\forall w \in \{1, ..., W\}, n_{c_w}^+ n_{i_2}^- > n_{i_2}^+ n_{c_w}^-$. Thus, $(\sum_{w=1}^W n_{c_w}^+)n_{i_2}^- - (\sum_{w=1}^W n_{c_w}^-)n_{i_2}^+ > 0$. Moreover, because $r_{i_1} > r_{i_2}$, $n_{i_2}^- n_{i_1}^+ > n_{i_1}^- n_{i_2}^+$. Hence, $\Delta_{\text{AUC}_{\text{ch}}} > 0$.

3. l_{i_1} has a ranking same as pos (which requires $r_{i_1} \leq r_i$) and l_{i_2} has a ranking $\text{pos}+1$ (which requires $r_i \leq r_{i_2}$).

   In this case, the change of AUC_{ch} is caused by l_{i_1} and l_{i_2}. Then we compute the change in the AUC_{ch} as follows:

$$\Delta_{\text{AUC}_{\text{ch}}} = \frac{1}{N^+N^-}\Bigg(\frac{n_{i_1}^- n_{i_1}^+}{2} + n_{i_2}^- n_{i_1}^+ + \frac{n_{i_2}^- n_{i_2}^+}{2} - \frac{n_i^- n_i^+}{2}\Bigg). \tag{33}$$

   We derive the expression in the similar way as case 1 and case 2. Term $n_{i_2}^- n_{i_1}^+$ comes from the area of rectangle colored in dark pink in Figure 8 (h) and terms $\frac{n_{i_1}^- n_{i_1}^+}{2}$ and $\frac{n_{i_2}^- n_{i_2}^+}{2}$ handle the top triangles colored in light pink. Term $\frac{n_i^- n_i^+}{2}$ deals with the top triangle colored in light green in Figure 8 (g). Subtracting green shaded areas from pink shaded areas, we get $\Delta_{\text{AUC}_{\text{ch}}}$, which is represented by the (remaining) pink area in Figure 8 (i).

   Recall $n_i^- = n_{i_1}^- + n_{i_2}^-$ and $n_i^+ = n_{i_1}^+ + n_{i_2}^+$. Simplifying Equation (33), we get

$$\Delta_{\text{AUC}_{\text{ch}}} = \frac{1}{N^+N^-}\Bigg(\frac{n_{i_1}^+ n_{i_2}^- - n_{i_2}^+ n_{i_1}^-}{2}\Bigg) \tag{34}$$

Since $r_{i_1} > r_{i_2}$, $n^+_{i_1} n^-_{i_2} - n^+_{i_2} n^-_{i_1}$. Therefore, $\Delta_{\text{AUC}_{\text{ch}}} > 0$.

4. l_{i_1} has a ranking same as pos (which requires $r_{i_1} \leq r_i$) and l_{i_2} has a ranking larger than pos + 1 (which requires $r_i > r_{i_2}$).

   Let $A = \{l_{a_1}, ..., l_{a_U}\}$ be a collection of leaves that ranked before l_{i_1} and $B = \{l_{b_1}, ..., l_{b_V}\}$ be a collection of leaves that ranked after l_{i_1} but before l_{i_2}. In this case the change of AUC_{ch} is caused by $l_{i_1}, l_{b_1}, ..., l_{b_V}, l_{i_2}$. Then we can compute the change as follows:

$$\Delta_{\text{AUC}_{\text{ch}}} = \frac{1}{N^+N^-}\left(\frac{n^-_{i_1} n^+_{i_1}}{2} + \left(\sum_{v=1}^{V} n^-_{b_v}\right) n^+_{i_1} + n^-_{i_2}\left[n^+_{i_1} + \left(\sum_{v=1}^{V} n^+_{b_v}\right) + \frac{n^+_{i_2}}{2}\right] - \frac{n^-_i n^+_i}{2} - \left(\sum_{v=1}^{V} n^-_{b_v}\right) n^+_i\right) \quad (35)$$

   The Figure 8 (j-l) show where each of the terms arises within the $\Delta_{\text{AUC}_{\text{ch}}}$: terms $n^-_{b_v} n^+_{i_1}$ and $n^-_{i_2}[n^+_{i_1} + (\sum_{v=1}^{V} n^+_{b_v})]$ come from the area of rectangles colored in dark pink in Figure 8 (k) and terms $\frac{n^-_{i_1} n^+_{i_1}}{2}$ and $\frac{n^-_{i_2} n^+_{i_2}}{2}$ handle triangles colored in light pink. Term $n^-_{b_v} n^+_i$ represents the rectangle colored in dark green in Figure 8 (j) and term $\frac{n^-_i n^+_i}{2}$ deals with the triangle colored in light green. Subtracting green shaded areas from pink shaded areas, we get $\Delta_{\text{AUC}_{\text{ch}}}$, which is represented by the (remaining) pink area in Figure 8 (l).

   Recall $n^-_i = n^-_{i_1} + n^-_{i_2}$ and $n^+_i = n^+_{i_1} + n^+_{i_2}$. Simplifying Equation (35), we get

$$\begin{aligned}\Delta_{\text{AUC}_{\text{ch}}} &= \frac{1}{N^+N^-}\left(\left(\sum_{v=1}^{V} n^-_{b_v}\right) n^+_{i_1} + \left(\sum_{v=1}^{V} n^+_{b_v}\right) n^-_{i_2} - \left(\sum_{v=1}^{V} n^-_{b_v}\right) n^+_i + \frac{n^-_{i_1} n^+_{i_1} + 2n^-_{i_2} n^+_{i_1} + n^-_{i_2} n^+_{i_2} - n^-_i n^+_i}{2}\right) \\ &= \frac{1}{N^+N^-}\left(\left(\sum_{v=1}^{V} n^+_{b_v}\right) n^-_{i_2} - \left(\sum_{v=1}^{V} n^-_{b_v}\right) n^+_{i_2} + \frac{n^-_{i_2} n^+_{i_1} - n^-_{i_1} n^+_{i_2}}{2}\right)\end{aligned} \quad (36)$$

   Since $r_{b_1} > ... > r_{b_v} > r_{i_2}, \forall v \in \{1, ..., V\}, n^+_{b_v} n^-_{i_2} > n^+_{i_2} n^-_{b_v}$. Thus, $(\sum_{v=1}^{V} n^+_{b_v}) n^-_{i_2} > (\sum_{v=1}^{V} n^-_{b_v}) n^+_{i_2}$. Since $r_{i_1} > r_{i_2}$, $n^+_{i_1} n^-_{i_2} - n^+_{i_2} n^-_{i_1}$. Therefore, $\Delta_{\text{AUC}_{\text{ch}}} > 0$.

Therefore, once an impure leaf is split, the AUC_{ch} doesn't decrease. If the split is leading to the change of the rank order of leaves, then AUC_{ch} increases. □

C.1. Proof of Theorem 2.1

Proof. Let tree $d = (d_{\text{fix}}, r_{\text{fix}}, d_{\text{split}}, r_{\text{split}}, K, H_d)$ and leaves of tree d are mutually exclusive. According to Lemma C.1, for leaves that can be further split, we can do splits to increase the AUC_{ch}. In the best possible hypothetical case, all new generated leaves are pure (contain only positive or negative samples). In this case, AUC_{ch} is maximized for d_{split}. In this case, we will show that $b(d_{\text{fix}}, \mathbf{x}, \mathbf{y}) = 1 - \frac{1}{N^+N^-}\left(\sum_{i=1}^{K} n^-_i \left[N^+_{\text{split}} + \left(\sum_{j=1}^{i-1} n^+_j\right) + \frac{1}{2} n^+_i\right] + N^+ N^-_{\text{split}}\right) + \lambda H_d$ as defined by Equation (4). Then $b(d_{\text{fix}}, \mathbf{x}, \mathbf{y}) \leq R(d, \mathbf{x}, \mathbf{y})$.

To derive the expression for $b(d_{\text{fix}}, \mathbf{x}, \mathbf{y})$, we sum areas of rectangles and triangles under the ROC curve's convex hull. Figure 9 shows where each of the terms arises within this sum: the first term in the sum, which is $n^-_i N^+_{\text{split}}$, comes from the area of the lower rectangle of the ROC curve's convex hull, colored in green. This rectangle arises from the block of N^+_{split} positives at the top of the ranked list (within the split leaves, whose hypothetical predictions are 1). The $n^-_i \left(\sum_{j=1}^{i-1} n^+_j\right)$ term handles the areas of the growing rectangles, colored in blue in Figure 9. The areas of the triangles comprising the top of the ROCCH account for the third term $\frac{1}{2} n^-_i n^+_i$. The final term $N^+ N^-_{\text{split}}$ comes from the rectangle on the right, colored red, stemming from the split observations within a hypothetical purely negative leaf.

□

C.2. Proof of Theorem 2.2

Proof. According to Theorem 2.1, $R(d', \mathbf{x}, \mathbf{y}) \geqslant b(d'_{\text{fix}}, \mathbf{x}, \mathbf{y})$ and $R(d, \mathbf{x}, \mathbf{y}) \geqslant b(d_{\text{fix}}, \mathbf{x}, \mathbf{y})$. Since $d_{\text{fix}} \subseteq d'_{\text{fix}}$, leaves in d_{split} but not in d'_{split} can be further split to generate pure leaves for tree d but fixed for tree d'. Based on Lemma C.1 and

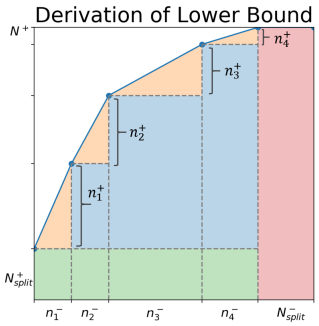


Figure 9. Example derivation of the hierarchical objective lower bound for rank statistics. N^+ at top left is the number of positive samples in the dataset. N^+_{split} at bottom left is the number of positive samples (N^-_{split} is the number of negative samples) captured by leaves that can be further split.

$H_d \leqslant H_{d'}$, $b(d_{\text{fix}}, \mathbf{x}, \mathbf{y}) \leqslant b(d'_{\text{fix}}, \mathbf{x}, \mathbf{y})$. Therefore, $b(d_{\text{fix}}, \mathbf{x}, \mathbf{y}) \leqslant R(d', \mathbf{x}, \mathbf{y})$. □

C.3. Incremental Progress Bound to Determine Splitting for Rank Statistics

Theorem C.2. *(Incremental progress bound to determine splitting for rank statistics) Let $d^* = (d_{\text{fix}}, r_{\text{fix}}, d_{\text{split}}, r_{\text{split}}, K, H_d)$ be any optimal tree with objective R^*, i.e., $d^* \in \text{argmin}_d R(d, \mathbf{x}, \mathbf{y})$. Consider tree d' derived from d^* by deleting a pair of leaves l_i and l_{i+1} and adding their parent leaf l_j, $d' = (l_1, ..., l_{i-1}, l_{i+2}, ..., l_{H_d}, l_j)$. Let n_j^+ be the number of positive samples (n_j^- be the number of negative samples) in leaf j. Calculate $\ell(d', \mathbf{x}, \mathbf{y})$ as in Equation (2). Define d'_{-j} as the tree d' after dropping leaf j, adding two hypothetical pure leaves (i.e., one has all positives and the other has all negatives), and reordering the remaining $H_d - 2$ leaves based on the fraction of positives. Then, we can calculate the loss of the tree d'_{-j} as*

$$\ell(d'_{-j}, \mathbf{x}, \mathbf{y}) = 1 - \frac{1}{N^+ N^-}\left(\sum_{i=1}^{H_d-2} n_i^- \left[n_j^+ + \left(\sum_{v=1}^{i-1} n_v^+\right) + \frac{1}{2}n_i^+\right] + N^+ n_j^-\right). \tag{37}$$

Let $\tau := \ell(d', \mathbf{x}, \mathbf{y}) - \ell(d'_{-j}, \mathbf{x}, \mathbf{y})$. Then τ must be at least λ.

Proof. By the way $\ell(d'_{-j}, \mathbf{x}, \mathbf{y})$ is defined (using the same counting argument we used in the proof of Theorem 2.1), it is a lower bound for $\ell(d^*, \mathbf{x}, \mathbf{y})$. These two quantities are equal when the split leaves are all pure. So, we have $\ell(d'_{-j}, \mathbf{x}, \mathbf{y}) \leq \ell(d^*, \mathbf{x}, \mathbf{y})$. Since $\ell(d', \mathbf{x}, \mathbf{y}) - \ell(d^*, \mathbf{x}, \mathbf{y}) \leqslant \ell(d', \mathbf{x}, \mathbf{y}) - \ell(d'_{-j}, \mathbf{x}, \mathbf{y}) = \tau$, we can get $\ell(d', \mathbf{x}, \mathbf{y}) + \lambda(H_d - 1) \leqslant \ell(d^*, \mathbf{x}, \mathbf{y}) + \lambda(H_d - 1) + \tau$, that is (and remember that d^* is of size H_d whereas d' is of size $H_d - 1$), $R(d', \mathbf{x}, \mathbf{y}) \leq R(d^*, \mathbf{x}, \mathbf{y}) - \lambda + \tau$. Since d^* is optimal with respect to R, then $0 \leqslant R(d', \mathbf{x}, \mathbf{y}) - R(d^*, \mathbf{x}, \mathbf{y}) \leqslant -\lambda + \tau$, thus $\tau \geqslant \lambda$. □

Similar to the Incremental Progress Bound to Determine Splitting for arbitrary monotonic losses, for a tree d, if any of its internal node contributes less than λ in loss, it is not the optimal tree. Thus, after evaluating tree d, we can prune it.

C.4. Equivalent points bound for rank statistics

Similar to the equivalent points bound for arbitrary monotonic losses, for a tree $d = (d_{\text{fix}}, r_{\text{fix}}, d_{\text{split}}, r_{\text{split}}, K, H_d)$, the objective of this tree and its children is minimized when leaves that can be further split to generate pure leaves. In the case when it is possible to split the data into pure leaves, the risk could be equal to $b(d_{\text{fix}}, \mathbf{x}, \mathbf{y})$. However, if multiple observations captured by a leaf in d_{split} have the same features but different labels, then no tree, including those that extend d_{split}, can correctly classify all of these observations; that is, leaves in d_{split} can not generate pure leaves. In this case, we can leverage these equivalent points to give a tighter lower bound for the objective. We use the same notation for capture and set of equivalent points as in Appendix B, and minority class label is simply the label with fewer samples.

Let $d = (d_{\text{fix}}, r_{\text{fix}}, d_{\text{split}}, r_{\text{split}}, K, H_d)$ be a tree. Data in leaves from d_{split} can be separated into U equivalence classes. For $u = 1, ..., U$, let $N_u = \sum_{i=1}^N \text{cap}(x_i, d_{\text{split}}) \wedge \mathbb{1}[x_i \in e_u]$, that is, the number of samples captured by d_{split} belonging to equivalence set u, and let $\delta_u = \sum_{i=1}^N \text{cap}(x_i, d_{\text{split}}) \wedge \mathbb{1}[x_i \in e_u]\mathbb{1}[y_i = q_u]$ be the number of minority-labeled samples

captured by d_{split} in equivalence point set u. Then we define $\tilde{r}_u$ as the classification rule we would make on each equivalence class separately (if we were permitted):

$$\tilde{r}_u = \begin{cases} \frac{\delta_u}{N_u} & \text{if} \quad \delta_u \geq 0 \text{ and } q_u = 1 \\ \frac{N_u - \delta_u}{N_u} & \text{if} \quad \delta_u \geq 0 \text{ and } q_u = 0. \end{cases}$$

Let us combine this with d_{fix} to get a bound. We order the combination of leaves in d_{fix} and equivalence classes $u \in \{1, ..., U\}$ by the fraction of positives in each, $\text{Sort}(r_1, ..., r_i, ..., r_K, \tilde{r}_1, ..., \tilde{r}_u, ..., \tilde{r}_U)$ from highest to lowest. Let us re-index these sorted $K+U$ elements by index $\tilde{i}$. Denote $n_{\tilde{i}}^+$ as the number of positive samples in either the leaf or equivalence class corresponding to $\tilde{i}$. (Define n_i^- to be the number of negative samples analogously). We define our new, tighter, lower bound as:

$$b(d_{\text{equiv}}, \mathbf{x}, \mathbf{y}) = 1 - \frac{1}{N^+ N^-} \sum_{\tilde{i}=1}^{K+U} n_{\tilde{i}}^- \left[\left(\sum_{j=1}^{\tilde{i}-1} n_j^+ \right) + \frac{1}{2} n_{\tilde{i}}^+ \right] + \lambda H_d. \tag{38}$$

Theorem C.3. *(Equivalent points bound for rank statistics) For a tree $d = (d_{\text{fix}}, r_{\text{fix}}, d_{\text{split}}, r_{\text{split}}, K, H_d)$, Let tree $d' = (d'_{\text{fix}}, r'_{\text{fix}}, d'_{\text{split}}, r'_{\text{split}}, K', H_{d'}) \in \sigma(d)$ be any child tree such that its fixed leaves d'_{fix} contain d_{fix} and $H_{d'} \geqslant H_d$. Then $b(d_{\text{equiv}}, \mathbf{x}, \mathbf{y}) \leqslant R(d', \mathbf{x}, \mathbf{y})$, where $b(d_{\text{equiv}}, \mathbf{x}, \mathbf{y})$ is defined in Equation (38).*

Proof. The proof is similar to the proof of Theorem 2.1 and Theorem 2.2. □

C.5. Partial area under the ROCCH

We discuss the partial area under the ROC convex hull in this section. The ROCCH for a decision tree is defined in Section 2.1, where leaves are rank-ordered by the fraction of positives in the leaves. Given a threshold θ, the partial area under the ROCCH (pAUC_{ch}) looks at only the leftmost part of the ROCCH, that is focusing on the top ranked leaves. This measure is important for applications such as information retrieval and maintenance (see, e.g., Rudin & Wang, 2018). In our implementation, all bounds derived for the objective AUC_{ch} can be adapted directly for pAUC_{ch}, where all terms are calculated only for false positive rates smaller or equal to θ.

In our code, we implement all of the rank statistics bounds, with one exception for the partial AUCCH – the equivalent points bound. We do not implement the equivalent points bound for partial AUCCH since the pAUC_{ch} statistic is heavily impacted by the leaves with high fraction of positives, which means that the leaves being repeatedly calculated for the objective tend not to be impure, and thus the equivalence points bound is less effective.

D. Optimizing F-score with Decision Trees

For a labeled tree $d = (d_{\text{fix}}, \delta_{\text{fix}}, d_{\text{split}}, \delta_{\text{split}}, K, H)$, the F-score loss is defined as

$$\ell(d, \mathbf{x}, \mathbf{y}) = \frac{FP + FN}{2N^+ + FP - FN}. \tag{39}$$

For objectives like accuracy, balanced accuracy and weighted accuracy, the loss of a tree is the sum of loss in each leaf. For F-score loss, however, FP and FN appear in both numerator and denominator, thus the loss no longer can be calculated using a sum over the leaves.

Lemma D.1. *The label of a single leaf depends on the labels of other leaves when optimizing F-score loss.*

Proof. Let $l_1, ..., l_{H_d}$ be the leaves of tree d. Suppose $H_d - 1$ leaves are labeled. Let FP_{H_d-1} and FN_{H_d-1} be the number of false positives and false negatives of these $H_d - 1$ leaves, respectively. Let $A = FP_{H_d-1} + FN_{H_d-1}$ and $B = 2N^+ + FP_{H_d-1} - FN_{H_d-1}$, and by these definitions, we will always have $A \leq B$. Let $n_{H_d}^+$ be the number of positive samples in leaf H_d and $n_{H_d}^-$ be the number of negative samples. The leaf's predicted label can be either positive or negative. The loss of the tree depends on this predicted label as follows:

$$\text{If } \hat{y}_{H_d}^{(\text{leaf})} = 1, \text{there can be only false positives, thus } \ell(d, \mathbf{x}, \mathbf{y}) = \frac{A + n_{H_d}^-}{B + n_{H_d}^-}. \tag{40}$$

$$\text{If } \hat{y}_{H_d}^{(\text{leaf})} = 0, \text{there can be only false negatives, thus } \ell(d, \mathbf{x}, \mathbf{y}) = \frac{A + n_{H_d}^+}{B - n_{H_d}^+}. \tag{41}$$

Calculating loss (41) minus loss (40):

$$\frac{A+n^+_{H_d}}{B-n^+_{H_d}} - \frac{A+n^-_{H_d}}{B+n^-_{H_d}} = \frac{(A+n^+_{H_d})\times(B+n^-_{H_d}) - (A+n^-_{H_d})\times(B-n^+_{H_d})}{(B-n^+_{H_d})(B+n^-_{H_d})}. \tag{42}$$

Denote Δ as the numerator of (42), that is

$$(A+n^+_{H_d})\times(B+n^-_{H_d}) - (A+n^-_{H_d})\times(B-n^+_{H_d}).$$

Then we can get

$$\begin{aligned}\Delta &= AB + An^-_{H_d} + Bn^+_{H_d} + n^+_{H_d}n^-_{H_d} - AB + An^+_{H_d} - Bn^-_{H_d} + n^-_{H_d}n^+_{H_d} &(43)\\ &= 2n^+_{H_d}n^-_{H_d} + An^+_{H_d} + Bn^+_{H_d} + An^-_{H_d} - Bn^-_{H_d}. &(44)\end{aligned}$$

The value of Δ depends on A, B, $n^+_{H_d}$ and $n^-_{H_d}$. Hence, in order to minimize the loss, the predicted label of leaf H_d is 0 if $\Delta \leq 0$ and 1 otherwise. Therefore, the predicted label of a single leaf depends on A and B, which depend on the labels of the other samples, as well as the positive and negative samples captured by that leaf. □

Theorem D.2. *(Optimizing F-score Poses a Unusual Challenge) Let $l_1, ..., l_{H_d}$ be the leaves of tree d and let N^+ be the number of positive samples in the dataset. Let Γ_1 and Γ_2 be two predicted labelings for the first $H_d - 1$ leaves. Leaf H_d has a fixed predicted label. Suppose the loss for the F-score (Equation (39)) of the first $H_d - 1$ leaves based on labeling method Γ_1 is smaller than the loss based on labeling method Γ_2 (where in both cases, leaf H_d has the same predicted label). It is not guaranteed that the F_1 loss of the tree d based on the first labeling Γ_1 is always smaller than the loss based on the second labeling Γ_2.*

Proof. Let $FP^{(1)}_{H_d-1}$ and $FN^{(1)}_{H_d-1}$ be the number of false positives and number of false negatives for the first $H_d - 1$ leaves from the labeling method Γ_1 and similarly define $FP^{(2)}_{H_d-1}$ and $FN^{(2)}_{H_d-1}$ for labeling method Γ_2. Denote $A_1 = FP^{(1)}_{H_d-1} + FN^{(1)}_{H_d-1}$ and $B_1 = 2N^+ + FP^{(1)}_{H_d-1} - FN^{(1)}_{H_d-1}$. Similarly, denote $A_2 = FP^{(2)}_{H_d-1} + FN^{(2)}_{H_d-1}$ and $B_2 = 2N^+ + FP^{(2)}_{H_d-1} - FN^{(2)}_{H_d-1}$. As we know from the assumptions of the theorem, $\frac{A_1}{B_1} \leq \frac{A_2}{B_2}$.

Denote FP_{H_d} and FN_{H_d} be the number of false positives and number of false negatives of the last leaf l_{H_d}.

Let $\ell^{(1)}(d, \mathbf{x}, \mathbf{y})$ and $\ell^{(2)}(d, \mathbf{x}, \mathbf{y})$ be the loss of the tree d based on two different predicted labelings of the leaves.

Suppose the predicted label of leaf l_{H_d} is 1. (An analogous result holds when the predicted label of leaf l_{H_d} is 0.) Then $\ell^{(1)}(d, \mathbf{x}, \mathbf{y}) = \frac{A_1+FP_{H_d}}{B_1+FP_{H_d}}$ and $\ell^{(2)}(d, \mathbf{x}, \mathbf{y}) = \frac{A_2+FP_{H_d}}{B_2+FP_{H_d}}$. Let Δ be the numerator of $\ell^{(2)}(d, \mathbf{x}, \mathbf{y}) - \ell^{(1)}(d, \mathbf{x}, \mathbf{y})$.

$$\begin{aligned}\Delta &= (A_2 + FP_{H_d})(B_1 + FP_{H_d}) - (A_1 + FP_{H_d})(B_2 + FP_{H_d})\\ &= A_2B_1 - A_1B_2 + (A_2 - B_2 + B_1 - A_1)FP_{H_d}. &(45)\end{aligned}$$

Since $\frac{A_1}{B_1} \leq \frac{A_2}{B_2}$, $A_2B_1 \geq A_1B_2$, that is, the first two terms together are nonnegative. Meanwhile, $A_1 \leq B_1$ and $A_2 \leq B_2$. Thus, Δ could be negative or positive. Therefore, even though the labeling method Γ_1 leads to smaller loss for the first $H_d - 1$ leaves and the label of the last leaf depends on the label of previous $H_d - 1$ leaves, it is not guaranteed that the loss of the tree is smaller than that based on Γ_2. It is easy to construct examples of A_1, A_2, B_1, B_2, and FP_{H_d} where the result is either positive or negative, as desired. □

Lemma D.1 and Theorem D.2 indicate that optimizing F-score loss is much harder than other arbitrary monotonic losses such as balanced accuracy and weighted accuracy. Thus, we simplify the labeling step by incorporating a parameter ω at each leaf node s.t. l_i is labeled as 0 if $\omega n^+_i \leqslant n^-_i$ and 1 otherwise $\forall i \in \{1, ..., H_d\}$.

E. Dynamic Programming Formulation

Note that this section describes standard dynamic programming, where possible splits describe subproblems. The more interesting aspects are the bounds and how they interact with the dynamic programming.

We will work only with the weighted misclassification loss for the following theorem, so that the loss is additive over the data:

$$\ell(d, \mathbf{x}, \mathbf{y}) = \sum_i \text{weight}_i \text{loss}(x_i, y_i).$$

We denote $(\mathbf{x}, \mathbf{y})$ as a data set of features $\mathbf{x}$ and binary labels $\mathbf{y}$ containing a total of N samples and M features.

Initial Problem: We define a tree optimization problem as a minimization of the regularized risk $R(d, \mathbf{x}, \mathbf{y})$ over the domain $\sigma(D)$, where D is a tree consisting of a single split leaf

$$D = (D_{\text{fix}}, r_{\text{fix}}, D_{\text{split}}, r_{\text{split}}, K, H) = (\emptyset, \emptyset, D_{\text{split}}, r_{\text{split}}, 0, 1)$$

$$d^* \in \text{argmin}_{d \in \sigma(D)} R(d, \mathbf{x}, \mathbf{y}). \tag{46}$$

Since all trees are descendants of a tree that is a single split leaf, this setup applies to tree optimization of any arbitrary data set $\mathbf{x}$, $\mathbf{y}$. We can rewrite the optimization problem as simply:

$$d^* \in \text{argmin}_d R(d, \mathbf{x}, \mathbf{y}). \tag{47}$$

We partition the domain $d \in \sigma(D)$ into $M+1$ cases: One Leaf Case and M Tree Cases. We solve each case independently, then optimize over the solutions of each case:

$$\sigma(D) = \text{Leaf} \cup \text{Tree}_1 \cup \text{Tree}_2 \cup \cdots \cup \text{Tree}_M$$

$$d^*_{\text{Leaf}} \in \text{argmin}_{d \in \sigma(\text{Leaf})} R(d, \mathbf{x}, \mathbf{y})$$
$$d^*_{\text{Tree}_i} \in \text{argmin}_{d \in \sigma(\text{Tree}_i)} R(d, \mathbf{x}, \mathbf{y})$$
$$\text{argmin}_{d \in \sigma(D)} \in \text{argmin}_{d \in \{d^*_{\text{Leaf}}, d^*_{\text{Tree}_1}, d^*_{\text{Tree}_2}, \ldots, d^*_{\text{Tree}_M}\}}.$$

The Leaf Case forms a base case in a recursion, while each Tree Case is a recursive case that further decomposes into two instances of tree optimization of the form described in (46).

Leaf Case: In this case, d^*_{Leaf} is a tree consisting of a single fixed leaf. This tree's only prediction r^*_{fix} is a choice of two possible classes $\{0, 1\}$.

$$r^*_{\text{fix}} \in \text{argmin}_{r_{\text{fix}} \in \{\text{true,false}\}} R((d_{\text{fix}}, r_{\text{fix}}, \emptyset, \emptyset, 1, 1), \mathbf{x}, \mathbf{y}),$$
$$d^*_{\text{Leaf}} = (d_{\text{fix}}, r^*_{\text{fix}}, \emptyset, \emptyset, 1, 1) \tag{48}$$

where a tie would be broken randomly.

Tree Case: For every possible i in the set feature indices $\{1, 2, 3, ..., M\}$ we designate an i^{th} Tree Case and an $d^*_{\text{Tree}_i}$ as the optimal descendent of a tree D^i. We define D^i as a tree consisting of a root split on feature i and two resulting split leaves d^{Left} and d^{Right} so that:

$$\text{Tree}_i = \sigma(D^i) \quad \text{(the children of } D^i\text{)}$$
$$D^i = (\emptyset, \emptyset, d_{\text{split}}, r_{\text{split}}, 0, 2) = (\emptyset, \emptyset, \{d^{\text{Left}}, d^i\}, \{r^{-i}, r^i\}, 0, 2)$$
$$d^*_{\text{Tree}_i} \in \text{argmin}_{d \in \sigma(D^i)} R(d, \mathbf{x}, \mathbf{y}). \tag{49}$$

Instead of directly solving (49), we further decompose this into two smaller tree optimization problems that match the format of (46). Since we are working with the weighted misclassification loss, we can optimize subtrees extending from from d^{-i} and d^i independently. We define data within the support set of d^{-i} as $\mathbf{x}^{-i}$, $\mathbf{y}^{-i}$. We define data within the support set of d^i as $\mathbf{x}^i$, $\mathbf{y}^i$. For each D^i, we define an optimization over the extensions of the left split leaf d^{-i} as:

$$\text{Left}^i = (\emptyset, \emptyset, \{d^{-i}\}, r_{\text{split}}, 0, 1).$$

$$d^*_{\text{Left}^i} \in \text{argmin}_{d \in \sigma(\text{Left}^i)} R(d, \mathbf{x}^{-i}, \mathbf{y}^{-i}). \tag{50}$$

By symmetry, we define an optimization over the extensions of the right split leaf d^i:

$$\text{Right}^i = (\emptyset, \emptyset, \{d^i\}, r_{\text{split}}, 0, 1)$$

$$d^*_{\text{Right}^i} \in \text{argmin}_{d \in \sigma(\text{Right}^i)} R(d, \mathbf{x}^i, \mathbf{y}^i). \tag{51}$$

$d^*_{\text{Left}^i}$ is the optimal subtree that classifies $\mathbf{x}^{-i}$ and $d^*_{\text{Right}^i}$ is the optimal subtree that classifies $\mathbf{x}^i$. Together, with a root node splitting on the i^{th} feature, $d^*_{\text{Left}^i}$ combines with $d^*_{\text{Right}^i}$ to form $d^*_{\text{Tree}_i}$. Thus, we can solve (50) and (51) to get the solution of (49).

In (46) we defined a decomposition of an optimization problem over the domain $\sigma(D)$, where D is a tree consisting of a single split leaf. Both expressions (50) and (51) are also optimizations over the domain of children of a tree consisting of a single split leaf. Recall that the descendants of any tree consisting of only a single split leaf covers the space of all possible trees, therefore the trees are optimized over an unconstrained domain. We can thus rewrite (50) as:

$$d^*_{\text{Left}^i} \in \text{argmin}_d R(d, \mathbf{x}^{-i}, \mathbf{y}^{-i}). \tag{52}$$

Symmetrically we can also rewrite (51) as:

$$d^*_{\text{Right}^i} \in \text{argmin}_d R(d, \mathbf{x}^i, \mathbf{y}^i). \tag{53}$$

Observe that (52) and (53) are simply tree optimization problems over a specific set of data (in this case $\mathbf{x}^{-i}, \mathbf{y}^{-i}$ and $\mathbf{x}^i, \mathbf{y}^i$). Hence, these tree optimizations form a recursion, and each can be solved as though they were (47).

Termination: To ensure this recursion terminates, we consider only splits where $\mathbf{x}^{-i}$ and $\mathbf{x}^i$ are strict subsets of $\mathbf{x}$. This ensures that the support strictly decreases until a minimum support is reached, which prunes all of $\text{Tree}_1 \cup \text{Tree}_2 \cup \cdots \cup \text{Tree}_M$ leaving only the leaf case described in (48).

Identifying Reusable Work: As we perform this decomposition, we identify each problem using its data set $\mathbf{x}$, $\mathbf{y}$ by storing a bit vector to indicate it as a subset of the initial data set. At each recursive step, we check to see if a problem has already been visited by looking for an existing copy of this bit vector.

Figure 10 shows a graphical representation of the algorithm. Note that we use the following shortened notations in the figure:

$$(\mathbf{x}, \mathbf{y})^k = (\mathbf{x}^k, \mathbf{y}^k) \tag{54}$$

$$(\mathbf{x}, \mathbf{y})^{-k} = (\mathbf{x}^{-k}, \mathbf{y}^{-k}) \tag{55}$$

$$(\mathbf{x}, \mathbf{y})^{k,l} = (\mathbf{x}^{k,l}, \mathbf{y}^{k,l}). \tag{56}$$

Equation (54) denotes a data set $(\mathbf{x}, \mathbf{y})$ filtered by the constraint that samples must respond positive to feature k. Equation (55) denotes a data set $(\mathbf{x}, \mathbf{y})$ filtered by the constraint that samples must respond negative to feature k. Equation (56) denotes a data set $(\mathbf{x}, \mathbf{y})$ filtered by the constraint that samples must respond positive to both feature k and feature l.

F. Incremental Similar Support Bound Proof

We will work only with weighted misclassification loss for the following theorem, so that the loss is additive over the data:

$$\ell(d, \mathbf{x}, \mathbf{y}) = \sum_i \text{weight}_i \text{loss}(x_i, y_i).$$

Define the maximum possible weighted loss:

$$\ell^{\max} = \max_{x,y} [\text{weight}(x, y) \times \text{loss}(x, y)].$$

The following bound is our important incremental similar support bound, which we leverage in order to effectively remove many similar trees from the search region by looking at only one of them.

Theorem F.1. *(Incremental Similar Support Bound) Consider two trees $d = (d_{\text{fix}}, d_{\text{split}}, K, H)$ and $D = (D_{\text{fix}}, D_{\text{split}}, K, H)$ that differ only by their root node (hence they share the same K and H values). Further, the root nodes between the two trees are similar enough that the support going to the left and right branches differ by at most ω fraction of the observations. (That is, there are ωN observations that are captured either by the left branch of d and right branch of D or vice versa.) Define $S_{\text{uncertain}}$ as the maximum of the support within d_{split} and D_{split}:*

$$S_{\text{uncertain}} = \max(\text{supp}(d_{\text{split}}), \text{supp}(D_{\text{split}})).$$

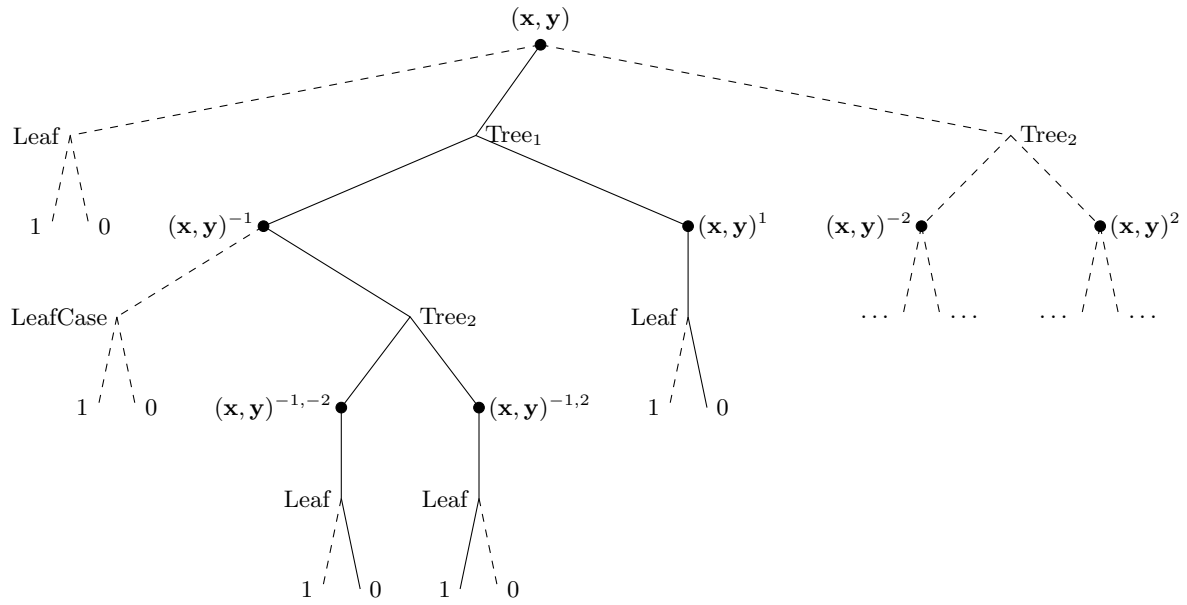


Figure 10. Graphical representation of dependency graph Produced by this algorithm. Filled vertices show problems identified by a support set. Solid edges show one possible tree that can be extracted from the graph.

For any child tree d' grown from d (grown from the nodes in d_{split}, that would not be excluded by the hierarchical objective lower bound) and for any child tree D' grown from D (grown from the nodes in D_{split}, that would not be excluded by the hierarchical objective lower bound), we have:

$$|R(d', \mathbf{x}, \mathbf{y}) - R(D', \mathbf{x}, \mathbf{y})| \leq (\omega + 2S_{\text{uncertain}})\ell^{\max}.$$

This theorem tells us that any two child trees of d and D that we will ever generate during the algorithm will have similar objective values. The similarity depends on ω, which is how many points are adjusted by changing the top split, and the other term involving $S_{\text{uncertain}}$ is determined by how much of the tree is fixed. If most of the tree is fixed, then there can be little change in loss among the children of either d or D', leading to a tighter bound. In standard classification tasks, the value of $\ell^{\max}$ is usually 1, corresponding to a classification error for an observation.

Proof. We will proceed in three steps. The first step is to show that

$$R(d, \mathbf{x}, \mathbf{y}) - R(D, \mathbf{x}, \mathbf{y}) \leq \omega\ell^{\max}.$$

The second step is to show:

$$R(d, \mathbf{x}, \mathbf{y}) \leq R(d', \mathbf{x}, \mathbf{y}) + S_{\text{uncertain}}\ell^{\max},$$

for all feasible children d' of d. The same bound will hold for D and any of its children D'. The third step is to show

$$R(d', \mathbf{x}, \mathbf{y}) \leq R(d, \mathbf{x}, \mathbf{y}) + S_{\text{uncertain}}\ell^{\max},$$

which requires different logic than the proof of Step 2. Together, Steps 2 and 3 give

$$|R(d', \mathbf{x}, \mathbf{y}) - R(d, \mathbf{x}, \mathbf{y})| \leq S_{\text{uncertain}}\ell^{\max}.$$

From here, we use the triangle inequality and the bounds from the three steps to obtain the final bound:

$$\begin{aligned}
&|R(d', \mathbf{x}, \mathbf{y}) - R(D', \mathbf{x}, \mathbf{y})| \\
&\quad = \quad |R(d', \mathbf{x}, \mathbf{y}) - R(d, \mathbf{x}, \mathbf{y}) + R(d, \mathbf{x}, \mathbf{y}) - R(D, \mathbf{x}, \mathbf{y}) + R(D, \mathbf{x}, \mathbf{y}) - R(D', \mathbf{x}, \mathbf{y})| \\
&\quad \leq \quad |R(d', \mathbf{x}, \mathbf{y}) - R(d, \mathbf{x}, \mathbf{y})| + |R(d, \mathbf{x}, \mathbf{y}) - R(D, \mathbf{x}, \mathbf{y})| + |R(D, \mathbf{x}, \mathbf{y}) - R(D', \mathbf{x}, \mathbf{y})| \\
&\quad \leq \quad S_{\text{uncertain}}\ell^{\max} + \omega\ell^{\max} + S_{\text{uncertain}}\ell^{\max},
\end{aligned}$$

which is the statement of the theorem. Let us now go through the three steps.

First step: Define "move" as the set of indices of the observations that either go down the left branch of the root of d and the right of D, or that go down the right of d and the left of D. The remaining data will be denoted "/move." These remaining data points will be classified the same way by both d and D. The expression above follows from the additive form of the objective R:

$$\begin{aligned} R(d, \mathbf{x}, \mathbf{y}) &= \ell(d, \mathbf{x}^{\text{move}}, \mathbf{y}^{\text{move}}) + \ell(d, \mathbf{x}^{/\text{move}}, \mathbf{y}^{/\text{move}}) + \lambda H, \\ R(D, \mathbf{x}, \mathbf{y}) &= \ell(D, \mathbf{x}^{\text{move}}, \mathbf{y}^{\text{move}}) + \ell(D, \mathbf{x}^{/\text{move}}, \mathbf{y}^{/\text{move}}) + \lambda H, \end{aligned}$$

and since $\ell(d, \mathbf{x}^{/\text{move}}, \mathbf{y}^{/\text{move}}) = \ell(D, \mathbf{x}^{/\text{move}}, \mathbf{y}^{/\text{move}})$ since this just considers overlapping leaves, we have:

$$|R(d, \mathbf{x}, \mathbf{y}) - R(D, \mathbf{x}, \mathbf{y})| \leq |\ell(d, \mathbf{x}^{\text{move}}, \mathbf{y}^{\text{move}}) - \ell(D, \mathbf{x}^{\text{move}}, \mathbf{y}^{\text{move}})| \leq \omega \ell^{\max}.$$

(For the last inequality, the maximum is attained when one of $\ell(d, \mathbf{x}^{\text{move}}, \mathbf{y}^{\text{move}})$ and $\ell(D, \mathbf{x}^{\text{move}}, \mathbf{y}^{\text{move}})$ is zero and the other attains its maximum possible value.)

Second step: Recall that d' is a child of d so that d'_{fix} contains d_{fix}. Let us denote the leaves in d'_{fix} that are not in d_{fix} by $d'_{\text{fix}}/d_{\text{fix}}$. Then,

$$\begin{aligned} R(d', \mathbf{x}, \mathbf{y}) &= \ell(d'_{\text{fix}}, \mathbf{x}, \mathbf{y}) + \ell(d'_{\text{split}}, \mathbf{x}, \mathbf{y}) + \lambda H_{d'} \\ &= \ell(d_{\text{fix}}, \mathbf{x}, \mathbf{y}) + \ell(d'_{\text{fix}}/d_{\text{fix}}, \mathbf{x}, \mathbf{y}) + \ell(d'_{\text{split}}, \mathbf{x}, \mathbf{y}) + \lambda H_{d'}. \end{aligned}$$

Adding $\ell(d_{\text{split}}, \mathbf{x}, \mathbf{y})$ to both sides,

$$\begin{aligned} R(d', \mathbf{x}, \mathbf{y}) + \ell(d_{\text{split}}, \mathbf{x}, \mathbf{y}) &= \ell(d_{\text{fix}}, \mathbf{x}, \mathbf{y}) + \ell(d_{\text{split}}, \mathbf{x}, \mathbf{y}) + \ell(d'_{\text{fix}}/d_{\text{fix}}, \mathbf{x}, \mathbf{y}) + \ell(d'_{\text{split}}, \mathbf{x}, \mathbf{y}) + \lambda H_{d'} \\ &= R(d, \mathbf{x}, \mathbf{y}) + \ell(d'_{\text{fix}}/d_{\text{fix}}, \mathbf{x}, \mathbf{y}) + \ell(d'_{\text{split}}, \mathbf{x}, \mathbf{y}) + \lambda(H_{d'} - H) \\ &\geq R(d, \mathbf{x}, \mathbf{y}), \end{aligned}$$

since the terms we removed were all nonnegative. Now,

$$\begin{aligned} R(d, \mathbf{x}, \mathbf{y}) &\leq R(d', \mathbf{x}, \mathbf{y}) + \ell(d_{\text{split}}, \mathbf{x}, \mathbf{y}) \\ &\leq R(d', \mathbf{x}, \mathbf{y}) + \text{supp}(d_{\text{split}}) \ell^{\max} \\ &\leq R(d', \mathbf{x}, \mathbf{y}) + S_{\text{uncertain}} \ell^{\max}. \end{aligned}$$

Third step: Here we will use the hierarchical objective lower bound. We start by noting that since we have seen d, we have calculated its objective $R(d, \mathbf{x}, \mathbf{y})$, and it must be as good or worse than than the current best value that we have seen so far (or else it would have replaced the current best). So $R^c \leq R(d, \mathbf{x}, \mathbf{y})$. The hierarchical objective lower bound (Theorem B.1) would be violated if the following holds. This expression states that $b(d', \mathbf{x}, \mathbf{y})$ is worse than $R(d, \mathbf{x}, \mathbf{y})$ (which is worse than the current best), which means we would have already excluded d' from consideration:

$$R^c \leq R(d, \mathbf{x}, \mathbf{y}) < b(d', \mathbf{x}, \mathbf{y}) = R(d', \mathbf{x}, \mathbf{y}) - \ell(d'_{\text{split}}, \mathbf{x}, \mathbf{y}).$$

This would be a contradiction. Thus, the converse holds:

$$R(d', \mathbf{x}, \mathbf{y}) - \ell(d'_{\text{split}}, \mathbf{x}, \mathbf{y}) = b(d', \mathbf{x}, \mathbf{y}) \leq R(d, \mathbf{x}, \mathbf{y}).$$

Thus,

$$R(d', \mathbf{x}, \mathbf{y}) \leq R(d, \mathbf{x}, \mathbf{y}) + \ell(d'_{\text{split}}, \mathbf{x}, \mathbf{y}).$$

Now to create an upper bound for $\ell(d'_{\text{split}}, \mathbf{x}, \mathbf{y})$ as $\ell^{\max}$ times the support of d'_{split}. Since d' is a child of d, its support on the split leaves is less than or equal to that of d, $\text{supp}(d'_{\text{split}}) \leq \text{supp}(d_{\text{split}})$. Thus, $\ell(d'_{\text{split}}, \mathbf{x}, \mathbf{y}) \leq \ell^{\max} \text{supp}(d_{\text{split}}) \leq \ell^{\max} S_{\text{uncertain}}$. Hence, we have the result for the final step of the proof, namely:

$$R(d', \mathbf{x}, \mathbf{y}) \leq R(d, \mathbf{x}, \mathbf{y}) + S_{\text{uncertain}} \ell^{\max}.$$

□

G. Subset Bound Proof

We will work with the loss that is additive over the data for the following theorem. The following bound is our subset bound, which we leverage in order to effectively remove the thresholds introduced by the continuous variables, thus pruning the search space.

Theorem G.1. *(Subset Bound). Define* $d = (d_{\text{fix}}, \delta_{\text{fix}}, d_{\text{split}}, \delta_{\text{split}}, K, H)$ *and* $D = (D_{\text{fix}}, \Delta_{\text{fix}}, D_{\text{split}}, \Delta_{\text{split}})$ *to be two trees with same root node. Let* f_1 *and* f_2 *be the features used to split the root node. Let* t_1, t_2 *be the left and right sub-trees under the root node split by* f_1 *in* d *and let* $(\mathbf{x}_{t_1}, \mathbf{y}_{t_1})$ *and* $(\mathbf{x}_{t_2}, \mathbf{y}_{t_2})$ *be the samples captured by* t_1 *and* t_2 *respectively. Similarly, let* T_1, T_2 *be the left and right sub-trees under the root node split by* f_2 *in* D *and let* $(\mathbf{x}_{T_1}, \mathbf{y}_{T_1})$ *and* $(\mathbf{x}_{T_2}, \mathbf{y}_{T_2})$ *be the samples captured by* T_1 *and* T_2 *respectively. Suppose* t_1, t_2 *are the optimal trees for* $(\mathbf{x}_{t_1}, \mathbf{y}_{t_1})$ *and* $(\mathbf{x}_{t_2}, \mathbf{y}_{t_2})$ *respectively, and* T_1, T_2 *are the optimal trees for corresponding* $(\mathbf{x}_{T_1}, \mathbf{y}_{T_1})$ *and* $(\mathbf{x}_{T_2}, \mathbf{y}_{T_2})$. *If* $R(t_1, \mathbf{x}_{t_1}, \mathbf{y}_{t_1}) \leq R(T_1, \mathbf{x}_{T_1}, \mathbf{y}_{T_1})$ *and* $(\mathbf{x}_{t_2}, \mathbf{y}_{t_2}) \subseteq (\mathbf{x}_{T_2}, \mathbf{y}_{T_2})$, *then* $R(d, \mathbf{x}, \mathbf{y}) \leq R(D, \mathbf{x}, \mathbf{y})$.

This theorem tells us that when f_1 and f_2 are from different thresholds of a continuous variable, for example age ≤ 20 and age ≤ 18, $(\mathbf{x}_{t_2}, \mathbf{y}_{t_2}) \subseteq (\mathbf{x}_{T_2}, \mathbf{y}_{T_2})$ is always true. In this case, we need only develop and compare the two left sub-trees.

Proof. Since $(\mathbf{x}_{t_2}, \mathbf{y}_{t_2}) \subseteq (\mathbf{x}_{T_2}, \mathbf{y}_{T_2})$ and t_2 and T_2 are optimal trees for $(\mathbf{x}_{t_2}, \mathbf{y}_{t_2})$ and $(\mathbf{x}_{T_2}, \mathbf{y}_{T_2})$ respectively,

$$R(t_2, \mathbf{x}_{t_2}, \mathbf{y}_{t_2}) \leq R(T_2, \mathbf{x}_{t_2}, \mathbf{y}_{t_2}) \leq R(T_2, \mathbf{x}_{T_2}, \mathbf{y}_{T_2}).$$

Given $R(t_1, \mathbf{x}_{t_1}, \mathbf{y}_{t_1}) \leq R(T_1, \mathbf{x}_{T_1}, \mathbf{y}_{T_1})$,

$$\begin{aligned} R(t_1, \mathbf{x}_{t_1}, \mathbf{y}_{t_1}) + R(t_2, \mathbf{x}_{t_2}, \mathbf{y}_{t_2}) &\leq R(T_1, \mathbf{x}_{T_1}, \mathbf{y}_{T_1}) + R(T_2, \mathbf{x}_{T_2}, \mathbf{y}_{T_2}) \\ R(d, \mathbf{x}, \mathbf{y}) &\leq R(D, \mathbf{x}, \mathbf{y}). \end{aligned} \tag{57}$$

□

H. Complexity of Decision Tree Optimization

In this section, we show the complexity of decision tree optimization.

Theorem H.1. *Let* $f(M)$ *be the number of binary decision trees that can be constructed for the dataset with* N *samples,* M *binary features and* K *label classes. The complexity of* $f(M)$ *is* $O(M!)$.

Proof. We show the proof by induction.

Base Case:

When $M = 0$, the dataset cannot be split and a predicted label is assigned to all samples. Therefore, the number of binary decision trees for the given dataset is equal to the number of classes K. Hence, $f(0) = K$.

Inductive Step:

When $M > 0$, the whole dataset can be split into two subsets by any of the M features. With M possible ways to do the first split, $2M$ subsets of data are created. Since binary features only have two different values, 0 and 1, a binary feature used to produce the subsets cannot be used again. Therefore, each of the $2M$ subsets can only be separated using trees constructed from at most $M - 1$ binary features. This produces a recursive definition of $f(M)$. That is $f(M) = 2Mf(M - 1)$.

Combining Step:

Each inductive step reduces M by one, guaranteeing its arrival at the base case. By combining the base case with the inductive step, a non-recursive definition of $f(M)$ is produced.

$$\begin{aligned} f(0) &= K \\ f(1) &= 2 \times 1 \times f(0) = 2^1 \times 1 \times K \\ f(2) &= 2 \times 2 \times f(1) = 2^2 \times 2 \times 1 \times K \\ f(3) &= 2 \times 3 \times f(2) = 2^3 \times 3 \times 2 \times 1 \times K \\ &\vdots \\ f(M) &= 2 \times M \times f(M-1) = 2^M \times M! \times K \end{aligned} \tag{58}$$

Since the term with the highest complexity in $(2^M)(M!)(K)$ is factorial, the complexity of $f(M)$ is $O(M!)$ □

I. Experiments

In this section, we elaborate on the experimental setup, data sets, pre-processing and post-processing. Additionally, we present extra experimental results that were omitted from the main paper due to space constraints.

I.1. Data Sets

We used a total of 11 data sets: Seven of them are from the UCI Machine Learning Repository (Dheeru & Karra Taniskidou, 2017), *(monk1, monk2, monk3, tic-tac-toe, balance scale, car evaluation, iris)*, one from LIBSVM (Chang & Lin, 2011), *(FourClass)*. The other three datasets are the ProPublica recidivism dataset (Larson et al., 2016) *(COMPAS)*, the Fair Isaac credit risk data sets (FICO et al., 2018) *(FICO)*, and the mobile advertisement data sets (Wang et al., 2017) *(coupon)*. We predict which individuals are arrested within two years of release (N = 5,020) on the recidivism data set, whether an individual will default on a loan for the FICO dataset, and whether a customer is going to accept a coupon for a bar considering demographic and contextual attributes.

I.2. Preprocessing

Missing Values: We exclude all observations with missing values.

monk 1, **monk 2**, **monk 3**, **tic-tac-toe**, **balance scale**, and **car evaluation**: We preprocessed these data sets, which contain only categorical features, by using a binary feature to encode every observable categorical value.

iris: We encode a binary feature to represent every threshold between adjacent values for all four continuous features (*sepal length*, *sepal width*, *petal length*, *petal width*). From the 3-class classification problem, we form 3 separate binary classification problems. Each binary classification is 1 of the 3 classes against the remaining classes. These three problems are referred to as **iris-setosa**, **iris-versicolor**, and **iris-virginica**.

Four Class: This dataset contains simulated points in a two-dimensional, bounded space with two classes that have irregular spreads over the space (241 positive samples and 448 negative samples). We split two continuous features into six categories (e.g. $\text{feature1} \leq 50$ and $\text{feature1} \leq 100$) and the value for each column is either 0 or 1.

ProPublica Recidivism (COMPAS): We discretized each continuous variable by using a binary feature to encode a threshold between each pair of adjacent values.

ProPublica Recidivism (COMPAS-2016): We selected features *age, count of juvenile felony, count of juvenile misdemeanor, count of juvenile other crimes, count of prior crimes*, and the target *recidivism within two years*. We replace *count of juvenile felony, count of juvenile misdemeanor, count of juvenile other crimes* with a single count called *count of juvenile crimes* which is the sum of *count of juvenile felony, count of juvenile misdemeanor, count of juvenile other crimes*. We discretized the *count of prior crimes* into four ranges *count of prior crimes = 0*, *count of prior crimes = 1*, *count of prior crimes between 2 to 3*, and *count of prior crimes* > 3. These four ranges are each encoded as binary features. Therefore, after preprocessing, these data contain 2 continuous features, 4 binary features, and one target.

ProPublica Recidivism (COMPAS-binary): We use the same discretized binary features of **compas** produced in (Hu et al., 2019) which are the following: *sex = Female, age* $<$ *21, age* $<$ *23, age* $<$ *26, age* $<$ *46, juvenile felonies = 0, juvenile misdemeanors = 0, juvenile crimes = 0, priors = 0, priors = 1, priors = 2 to 3, priors* $>$ *3.*

Fair Isaac Credit Risk (FICO): We discretized each continuous variable by using a binary feature to encode a threshold between each pair of adjacent values.

Fair Isaac Credit Risk (FICO-binary): We use the same discretized binary features of **compas** produced in (Hu et al., 2019) which are the following: *External Risk Estimate* $<$ *0.49 , External Risk Estimate* $<$ *0.65, External Risk Estimate* $<$ *0.80, Number of Satisfactory Trades* $<$ *0.5, Trade Open Time* $<$ *0.6, Trade Open Time* $<$ *0.85, Trade Frequency* $<$ *0.45, Trade Frequency* $<$ *0.6, Delinquency* $<$ *0.55, Delinquency* $<$ *0.75, Installment* $<$ *0.5, Installment* $<$ *0.7, Inquiry* $<$ *0.75, Revolving Balance* $<$ *0.4, Revolving Balance* $<$ *0.6, Utilization* $<$ *0.6, Trade W. Balance* $<$ *0.33.*

Mobile Advertisement (coupon): We discretized each continuous variable by using a binary feature to encode a threshold between each pair of adjacent values. We discretized each categorical variable by using a binary feature to encode each

observable categorical value.

Mobile Advertisement (bar-7): In order to predict whether a customer is going to accept a coupon for a bar, we selected features *age, passengers, bar, restaurant20to50, direction same*, and target. For features *age, bar* and *direction* we used the same encoding as we used for **coupon**. We modified *passengers* so that it is replaced with the binary feature *passengers* > 0. We modified *restaurant20to50* so that it is replaced with the binary feature *restaurant20to50=0*, which is 0 if the number of times that they eat at a restaurant with average expense less than $20 per person is less than 4 times per month and 1 otherwise.

Data Set Summary: Table 2 presents a summary of the datasets.

Data Set	Samples	Categorical Features	Continuous Features	Encoded Binary Features
monk 1	124	11	0	11
monk 2	169	11	0	11
monk 3	122	11	0	11
car evaluation	1729	15	0	15
balance scale	625	16	0	16
tic-tac-toe	958	18	0	18
iris	151	0	4	123
iris-setosa	151	0	4	123
iris-versicolor	151	0	4	123
iris-virginica	151	0	4	123
coupon	12684	26	0	129
bar-7	1913	5	0	14
Four Class	862	0	2	345
COMPAS	12382	2	20	647
COMPAS-2016	5020	4	2	85
COMPAS-binary	6907	12	0	12
FICO	1000	0	23	1407
FICO-binary	10459	17	0	17

Table 2. Comparison of different data sets (and their preprocessed derivatives). **Categorical Features** denote the number of features that are inherently categorical or represent discrete ranges of a continuous feature. These features generally have a low cardinality. **Continuous Features** denote the number of features that take on many integer or real number values. These features generally have a high cardinality. **Encoded Binary Features** denote the number of binary features required to encode all categorical and continuous values observed in the dataset. This is value is approximately the total cardinality of all categorical and continuous features.

I.3. Optimization Algorithms

CART: We run CART as a reference point for what is achievable with a greedy algorithm that makes no optimality guarantee. The algorithm is run using the Python implementation from Sci-Kit Learn. The depth and leaf count are constrained in order to adjust the resulting tree size.

BinOCT: BinOCT is modified to run using only a single thread to make comparison across algorithms fair. This algorithm runs on the academic version of CPLEX. The depth is constrained to adjust the resulting tree size.

DL8.5: DL8.5 is implemented in C++ and is run as a native extension of the Python interpreter. The depth is constrained to adjust the resulting tree size.

Because BinOCT and DL8.5 have hard constraints, rather than OSDT or GOSDT's soft constraints, GOSDT and OSDT's optimization problem is substantially harder than that of BinOCT and DL8.5. GOSDT and OSDT effectively consider a large number of possible tree sizes whereas the other algorithms consider only full trees of a given depth.

OSDT: OSDT is implemented in Python and run directly. The regularization coefficient is varied to adjust the resulting tree size.

PyGOSDT: PyGOSDT is an early Python implementation of GOSDT. The regularization coefficient is varied to adjust the resulting tree size.

GOSDT: GOSDT is implemented in C++ and run as a native executable. The regularization coefficient is varied to adjust the resulting tree size.

I.4. Computing Infrastructure

The experiments for optimizing rank statistics were run on a personal laptop with a 2.6GHz i5-7300U processor and 8GB of RAM.

All other experiments were run on a 2.30 GHz (30 MB L3 cache) Intel Xeon E7-4870 v2 processor with 60 cores across 4 NUMA nodes. We disabled hyper-threading. The server has 512 GB RAM uniformly distributed (128 GB each) across the four NUMA nodes. The host OS is Ubuntu 16.04.6 LTS. We set a 5-minute time limit on all experiments, unless otherwise stated. All algorithms that support multi-threading are modified to run sequentially.

I.5. Experiments: Rank Statistics

Collection and Setup: We ran this experiment on the data set **FourClass**. We train models to optimize various objectives with 30 minute time limits. When the time limit is reached, our algorithm returns the current best tree considering the objectives.

Results: Figure 11 shows the training ROC and test ROC of decision trees generated for six different objectives. Optimizing different objectives produces different trees with different FP and FN. Some interesting observations are that the pAUC_{ch} model performs as well as the AUC_{ch} model on the left part of the ROC curve, but then sacrifices some area under the middle and right of the curve (which is not as relevant to its objective) to obtain a sparser model (sparsity is relevant to the objective). The pAUC_{ch} and AUC_{ch} results illustrate how the objective allows us to trade off parts of the ROC curve (that are not important for some applications) with overall sparsity. Another interesting observation is that some of the models are extremely sparse: recall that each leaf is a single diagonal line on the ROC curve, so one can count the number of leaves by looking at the number of diagonal lines. In some cases, a well-chosen single split point can lead to a model with an excellent TPR/FPR tradeoff somewhere along the ROC curve.

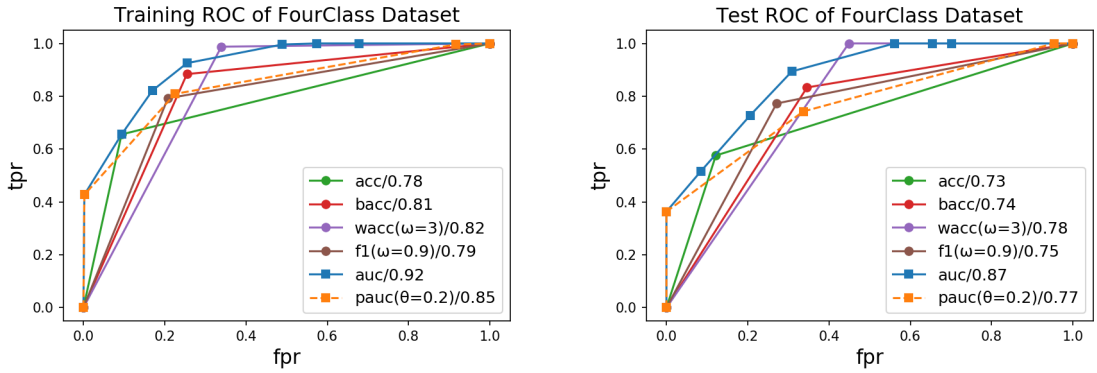


Figure 11. Training ROC and test ROC of FourClass dataset ($\lambda = 0.01$). A/B in the legend at the bottom right of each subfigures shows the objective and its parameter/area under the ROC.

I.6. Experiment: Accuracy vs Sparsity

Collection and Setup: We ran this experiment on the 6 data sets **car evaluations, COMPAS-binary, tic-tac-toe, monk 1, monk 2**, and **monk 3**. For the monk data sets, we used only the samples from the training set. For each data set we train models using varying configurations (described in the following sections) to produce models with varying number of leaves. For any single configuration, we perform a 5-fold cross validation to measure training accuracy and test accuracy for each fold. All runs that exceed the time limit of 5 minutes are discarded.

We omit PyGOSDT since it differs only from GOSDT in program speed, and would provide no additional information for

this experiment.

Below are the configurations used for each algorithm:

- *CART* Configurations: We ran this algorithm with 6 different configurations: depth limits ranging from 1 to 6, and a corresponding maximum leaf limit of 2, 4, 8, 16, and 64.
- *BinOCT* and *DL8.5* Configurations: We ran these algorithms with 6 different configurations: depth limits ranging from 1 to 6.
- *OSDT* and *GOSDT* Configurations: We ran these algorithms with 29 different regularization coefficients: 0.2, 0.1, 0.09, 0.08, 0.07, 0.06, 0.05, 0.04, 0.03, 0.02, 0.01, 0.009, 0.008, 0.007, 0.006, 0.005, 0.004, 0.003, 0.002, 0.001, 0.0009, 0.0008, 0.0007, 0.0006, 0.0005, 0.0004, 0.0003, 0.0002, and 0.0001.

Calculations: For each combination of data set, algorithm, and configuration, we produce a set of up to 5 models, depending on how many runs exceeded the time limit. We summarize the measurements (e.g., training accuracy, test accuracy, and number of leaves) across the set of up to 5 models by plotting the median. We compute the 25^{th} and 75^{th} percentile and show them as lower and upper error values respectively.

Results: Figure 12 shows that the objective optimized by GOSDT (same as OSDT) reliably produces a more efficient frontier between training accuracy and number of leaves. Figure 13 shows the same plots with test accuracy and number of leaves. The difference between frontiers sometimes becomes insignificant due to error introduced from generalization, particularly when the training accuracies between algorithms were close together. That is, if CART achieves a solution that is almost optimal, then it tends to achieve high test accuracy as well. *Without methods like GOSDT or OSDT, one would not be able to determine whether CART's training solution is close to optimal for a given number of leaves.* Further, if the training accuracies of the different algorithms are different (e.g., as in the monk2 data), this difference is reflected in an improved test accuracy for OSDT or GOSDT.

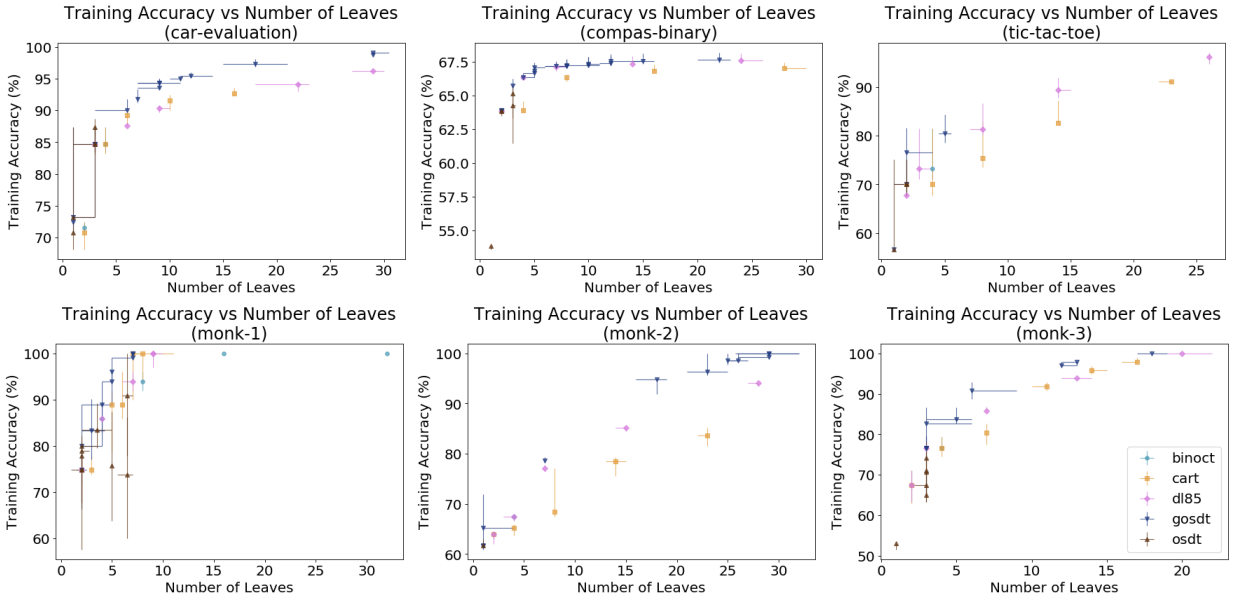


Figure 12. Training accuracy achieved by BinOCT, CART, DL8.5, GOSDT, and OSDT as a function of the number of leaves.

I.7. Experiment: Scalability

Collection and Setup: We ran this experiment on 4 data sets: **bar-7, compas-2016, compas**, and **fico**. The four data sets vary in the number of binary features required to fully represent their information. The number of binary features are, respectively, 14, 85, 647, and 1407. For each data set we show runtime as a function of both sample size and number of binary features used.

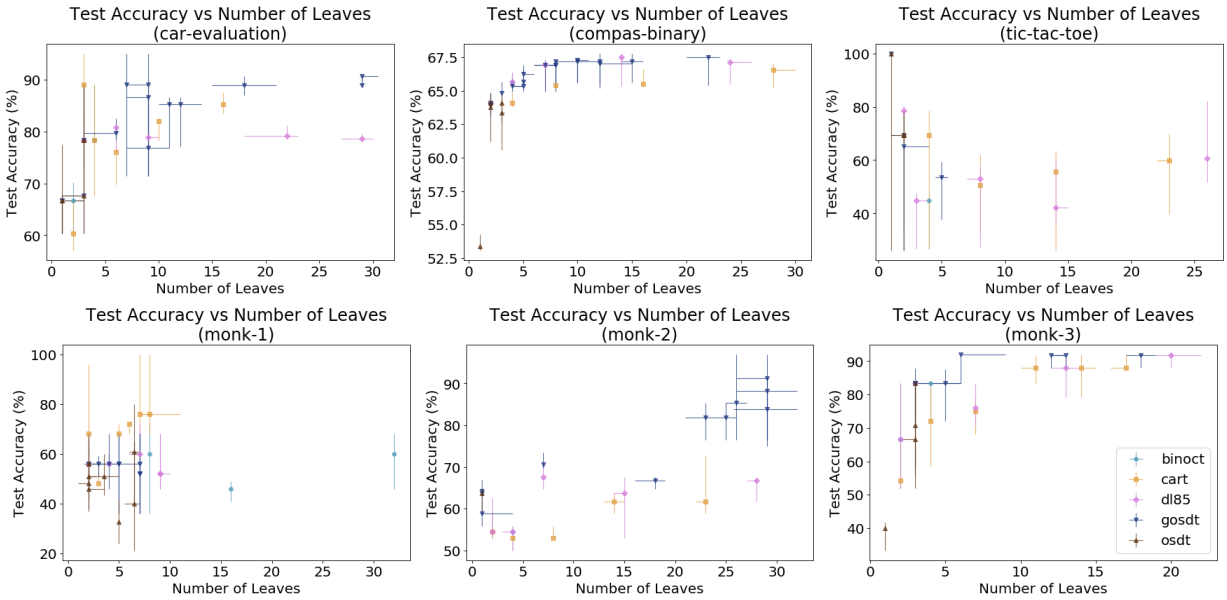


Figure 13. Test accuracy achieved by BinOCT, CART, DL8.5, GOSDT, and OSDT as a function of the number of leaves.

Each data set is preprocessed so that categorical features produce one binary feature per unique value, and continuous features produce one binary feature per pair of adjacent values. The samples are then randomly shuffled. We measure run time on increasingly larger subsets of this data (with all binary features included), this is our measure of run time as a function of sample size. We also measure run time on increasingly larger numbers of binary features (with all samples included), this is our measure of run time as a function of binary features. For all experiments we continue increasing the difficulty until either the difficulty covers the full data set or a time limit of 5 minutes has been exceeded 3 times by the same algorithm.

Note that when varying the number of binary features, we include all samples. This means that adding a feature to a large data set (e.g., COMPAS and FICO) generally increases the difficulty more than adding a feature to a small data set (e.g., bar-7 and COMPAS-2016). Likewise, when varying the number of samples, we include all binary features. This means that adding a feature to a high-dimensional data set (e.g., COMPAS and FICO) generally increases the difficulty more than adding a sample to a low-dimensional data set (e.g., bar-7 and COMPAS-2016). As a result, the sample size is not a good measure of difficulty when comparing across different data sets of completely different features. The number of binary features is a more robust measure when comparing across different data sets.

Below are the configurations used for each algorithm tested:

- *CART* is configured to have a maximum depth of $\log_2(32)$ and a maximum leaf count of 32.
- *BinOCT* and *DL8.5* are configured to have a maximum depth of $\log_2(32)$.
- *OSDT* and *GOSDT* are configured with a regularization coefficient of $\frac{1}{32}$.

While we initially attempted to include BinOCT in this experiment, we were unable to find an instance where BinOCT reached completion with a maximum depth of $\log_2(32)$ and a time limit of 5 minutes. Consequently, BinOCT was not included in this experiment.

Calculations: We provide two measures of speed. Training time measures the number of seconds required for an algorithm to complete with a certificate of optimality. Slow-down measures the ratio of the algorithm's training time against its fastest training time over values of problem difficulty. We vary and measure problem difficulty in two separate ways. "Number of binary features" indicates how many of the binary features generated by our binary encoding were included for training. "Number of samples" indicates how many samples were included for training.

Results: Figure 14 shows how each algorithm's training time varies as additional binary features are included. Figure 15 shows how each algorithm's training time varies as additional samples are included.

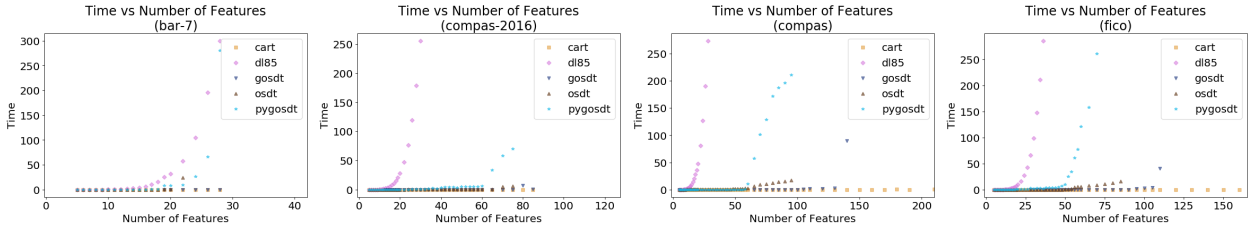
For *bar-7* and *compas-2016*, we observe a logarithmic time complexity when increasing sample size. These problems are sufficiently represented and solvable at a small sample size. As a result, additional samples contribute diminishing increase in the difficulty of the problem. Under these circumstances, GOSDT, PyGOSDT, and OSDT have a significant performance advantage over DL8.5.

For all data sets we observe an approximately factorial time complexity when increasing the number of features. This is consistent with the theoretical worst-case time complexity of full tree optimization (see Theorem H.1). The sharp increase in run time results in a limit on the size of problems solvable in practice by each algorithm. We observe that while all full tree optimization algorithms have such a limit, GOSDT usually has a higher limit than other algorithms.

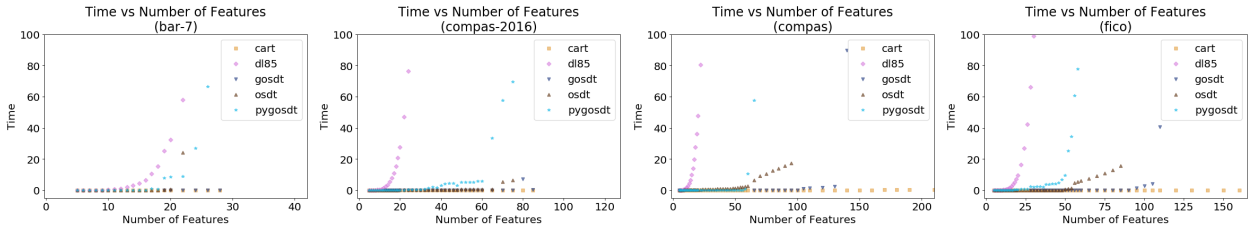
Figure 16 show how each algorithm's relative slow-down varies with additional binary features. Figure 17 show how each algorithm's relative slow-down varies with additional samples. This reduces the effects of constant overhead, showing the asymptotic behavior of each algorithm. Our observations from Figure 14 and Figure 15 still hold under this analysis. Additionally, we observe that the slow-down of GOSDT and PyGOSDT under the *bar-7* data set appears to become approximately constant; this is likely a result of additional samples belonging to already-present equivalence classes (the set of equivalence classes saturates). Recall that both PyGOSDT and GOSDT reduce the data set size to only the equivalence classes that are present in the data set, and thus scale in this quantity rather than the number of samples.

Overall, we observe that both GOSDT, PyGOSDT and OSDT have an advantage over *DL8.5* which becomes increasingly clearer as we test on data sets of greater difficulty. GOSDT and OSDT appears to perform better than PyGOSDT, with GOSDT having a slight advantage over OSDT on larger data sets.

Previous comparisons do not account for differences in implementation language. We observe that that GOSDT is several orders of magnitude faster and more scalable than DL8.5, both of which are implemented in C++. However, PyGOSDT is not quite as performant as OSDT, both of which are implemented in Python. This suggests, for data sets similar to the ones in this experiment, there are advantageous characteristics of OSDT that are worth further exploration for extensions of GOSDT.



(a) Training Time vs Number of Features (Full Scale)

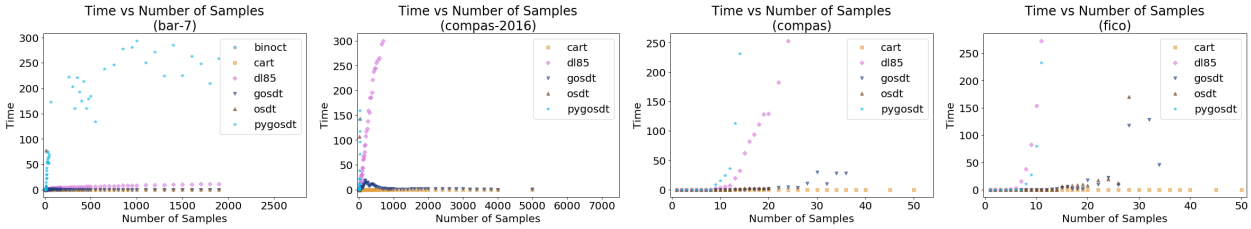


(b) Training Time vs Number of Features (Zoomed In)

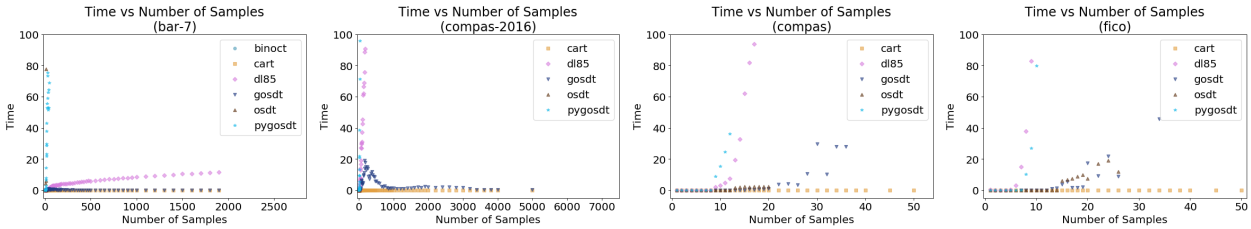
Figure 14. Time required to reach optimality (or to finish tree construction for non-optimal methods) for BinOCT, CART, DL8.5, GOSDT (C++), PyGOSDT (Python) and OSDT as a function of the number of binary features used to encode the continuous dataset ($\lambda = 0.3125$ or max depth = 5).

I.8. Experiment: Time to Optimality

Collection and Setup: We ran this experiment on 4 data sets: **bar-7, tic-tac-toe, car-evaluation, compas-binary, fico-binary, monk-1, monk-2**, and **monk-3**. For each experiment, we run *OSDT, GOSDT*, and *PyGOSDT* with a regularization



(a) Training Time vs Number of Samples (Full Scale)



(b) Training Time vs Number of Samples (Zoomed In)

Figure 15. Time required to reach optimality (or to finish tree construction for non-optimal methods) for BinOCT, CART, DL8.5, GOSDT (C++), PyGOSDT (Python) and OSDT as a function of the number of samples taken from the continuous dataset (λ = 0.3125 or max depth = 5).

coefficient of 0.005. For each run we track the progress of the algorithm by plotting the minimum objective score seen so far. Once the algorithm terminates or reaches a time limit of 30 minutes, the values are stored in a file.

Results: Figure 18 shows the different behaviors between GOSDT, PyGOSDT, and OSDT. In general, both PyGOSDT and GOSDT complete their certificate of optimality earlier than OSDT.

Note that PyGOSDT's implementation does not include high-priority bound updates. This causes PyGOSDT to maintain a higher objective score before making a sharp drop upon completion (with a certificate of optimality). GOSDT, on the other hand, behaves similarly to OSDT because both algorithms aggressively prioritize lowering the best observed objective score. We observe that under the tic-tac-toe data set this appears to be less advantageous. While PyGOSDT's progress initially appears less promising, it completed remarkably faster than both GOSDT and OSDT. This suggests that optimal prioritization is dependent on specifics of the optimization problem.

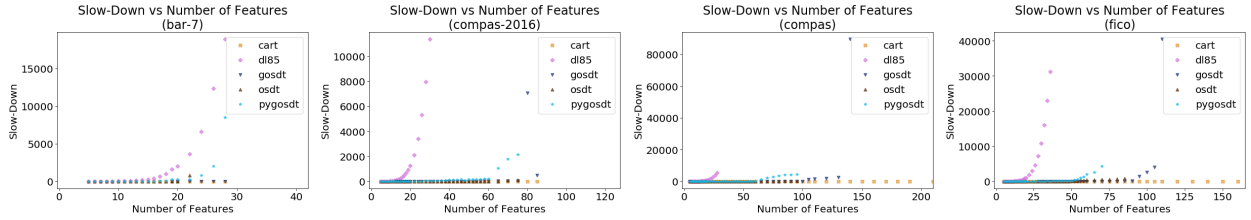
I.9. Optimal Trees

We present some of the trees that achieved the peak median accuracy from Section I.6. Figure 19 shows a comparison between the results of training BinOCT (a) and GOSDT (b) on the Monk 1 data set. GOSDT is able to produce a model with 20% higher accuracy than BinOCT even though both trees have 8 leaves. Figure 20 shows a comparison between DL8.5 (a) and GOSDT (b) on the Monk 2 data set. GOSDT is able to produce a model with 3% higher accuracy than DL8.5 even though both trees have 7 leaves. Figure 21 shows a comparison between BinOCT (a), DL8.5 (b), and GOSDT (c) on the Tic-Tac-Toe data set. GOSDT is able to produce a model with higher accuracy than both BinOCT and DL8.5 when all trees have 16 leaves.

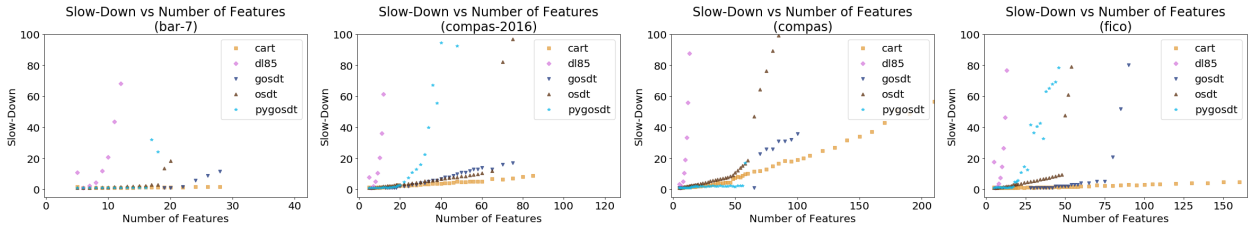
Comparison to True Model: For the results shown in Figure 19, we know that the true model used to generate the data in Monk 1 is a set of logical rules:

$$class = (jacket = red) \vee (head = body).$$

The data set we train on does not encode binary features for equality between two features (e.g., $head = body$) and categorical variables $head$ and $body$ are only encoded using $k-1$ binary rules (this means one value from each categorical variable will be expressed with a negation of all other values). Altogether, this means our encoding forces the true model



(a) Slow-Down vs Number of Features (Full Scale)



(b) Slow-Down vs Number of Features (Zoomed In)

Figure 16. Slow-down experienced by BinOCT, CART, DL8.5, GOSDT (C++), PyGOSDT (Python) and OSDT as a function of the number of binary features used to encode the continuous dataset ($\lambda = 0.3125$ or max depth = 5).

to instead be expressed as the following:

$$\begin{aligned} class = &(jacket = red) \\ &\vee (head = round \wedge body = round) \\ &\vee (head = square \wedge body = square) \\ &\vee (head \neq round \wedge head \neq square \wedge body \neq round \wedge body \neq square) \end{aligned}$$

We can interpret the trees produced by BinOCT as the following set of logical rules:

$$\begin{aligned} class = &(jacket = red \wedge head = round) \\ &\vee (jacket \neq red \wedge head = round \wedge body = round) \\ &\vee (jacket = red \wedge head \neq round \wedge body = round) \\ &\vee (jacket \neq green \wedge head \neq round \wedge body \neq round). \end{aligned}$$

We can interpret the trees produced by GOSDT as the following set of logical rules:

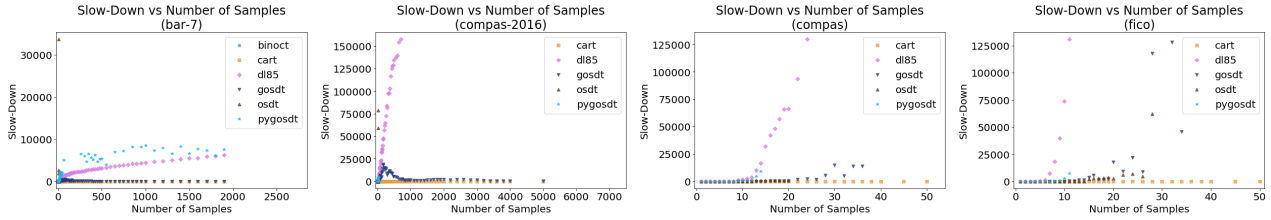
$$\begin{aligned} class = &(jacket = red) \\ &\vee (head = round \wedge body = round) \\ &\vee (head = square \wedge body = square) \\ &\vee (head \neq round \wedge head \neq square \wedge body \neq round \wedge body \neq square). \end{aligned}$$

In this instance, BinOCT produces a model that is similar to the true model but has a few mismatches. This is mainly due to the structural constraints of BinOCT. GOSDT, after exploring a larger space while still penalizing complexity, is able to produce a model that perfectly matches with the ground truth.

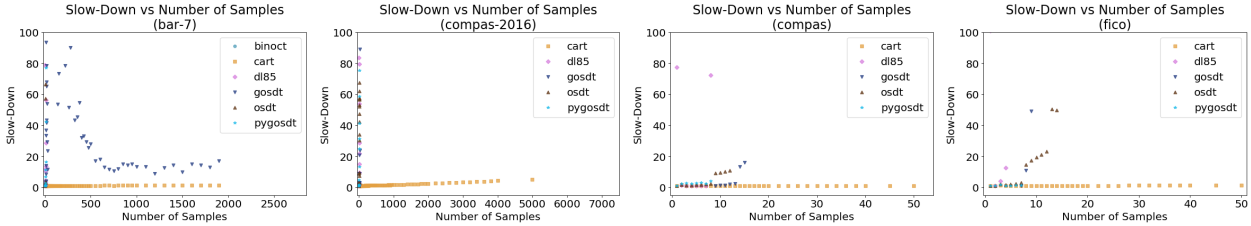
I.10. Summary of Experimental Results

Experiment G.5 shows that the new set of objective functions allows GOSDT to produce trees with a more efficient ROC curve than the standard accuracy objective assumed by other algorithms.

Experiment G.6 shows that the regularized risk objective used by OSDT and GOSDT produces the most efficient training accuracy vs. sparsity frontier. When placed under time constraints, GOSDT is able to produce more of the highly accurate models along this frontier than OSDT.



(a) Slow-Down vs Number of Samples (Full Scale)



(b) Slow-Down vs Number of Samples (Zoomed In)

Figure 17. Slow-down experienced by BinOCT, CART, DL8.5, GOSDT (C++), PyGOSDT (Python) and OSDT as a function of the number of samples taken from the continuous dataset ($\lambda = 0.3125$ or max depth = 5).

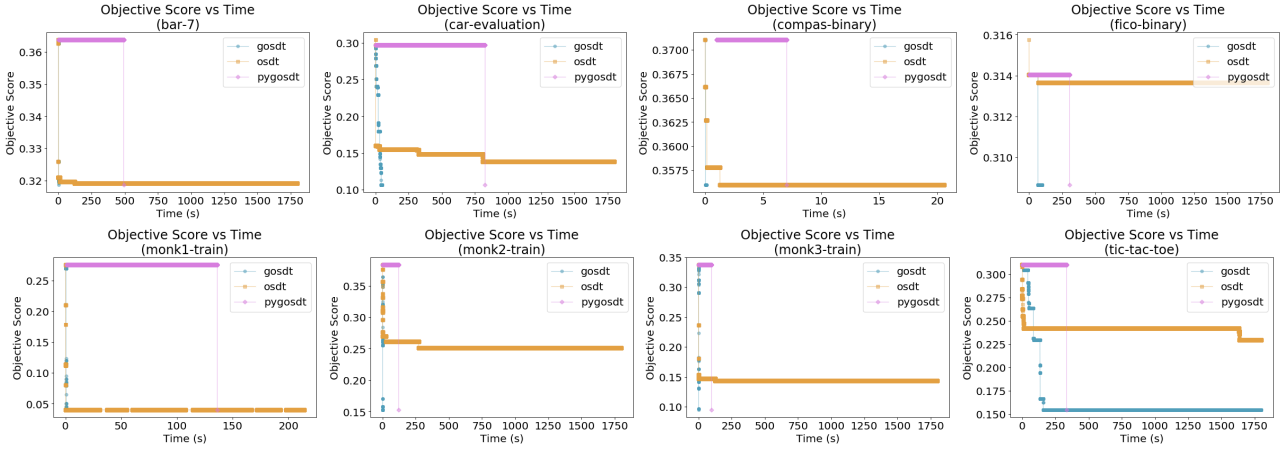


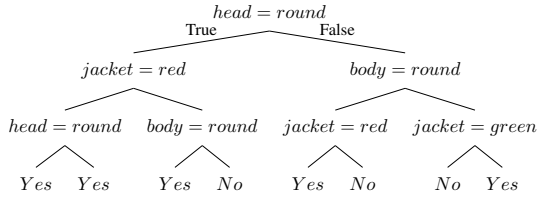
Figure 18. Best objective score of OSDT, GOSDT, and PyGOSDT over the course of their run time. ($\lambda = 0.005$)

Experiment G.7 shows that GOSDT is able to handle significantly more binary features than BinOCT, DL8.5, and, to a lesser extent, OSDT. Since binary features are used to encode thresholds over continuous features, GOSDT is able to handle continuous datasets of higher cardinality compared to other aforementioned methods.

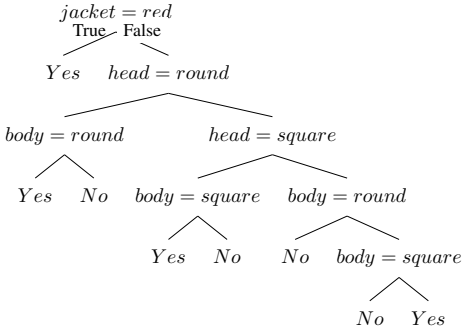
Experiment G.8 shows that GOSDT outpaces OSDT and PyGOSDT when it comes to reducing the optimality gap, this allows it to terminate with stronger optimality guarantees in the event of a premature termination.

Experiment G.9 shows that optimizing an efficient training accuracy vs. sparsity frontier allows GOSDT to more accurately capture the ground truth compared to BinOCT when subject to the same sparsity constraints.

To summarize, we began this experimental section by showing the benefits of optimizing more sophisticated objective functions. We then showed the benefits of a more efficient algorithm to support these objectives. Finally, we closed this section by combining these two elements to produce provably optimal and interpretable models and showcase their advantages.

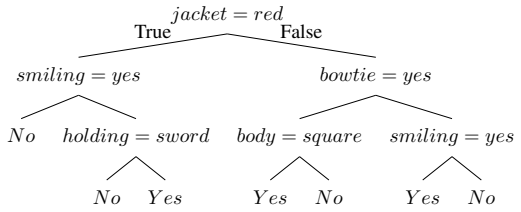


(a) BinOCT (training accuracy: 90.9%, test accuracy: 84.0%)

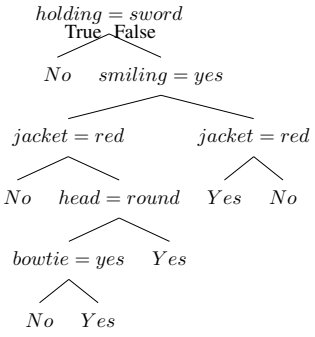


(b) GOSDT (training accuracy: 100%, test accuracy: 100%)

Figure 19. Eight-leaf decision trees generated by BinOCT and GOSDT on the Monk1 dataset. The tree generated by BinOCT includes two useless splits (the head=round in the bottom left), while GOSDT avoids this problem. BinOCT is 91% accurate, GOSDT is 100% accurate.



(a) DL8.5 (training accuracy: 76.3%, test accuracy: 70.6%)

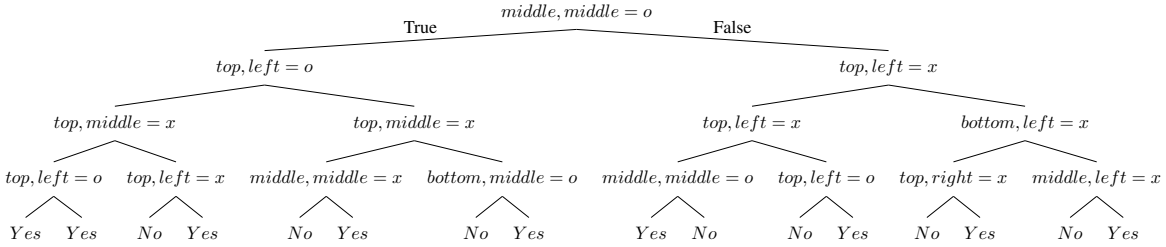


(b) GOSDT (training accuracy: 79.3%, test accuracy: 73.5%)

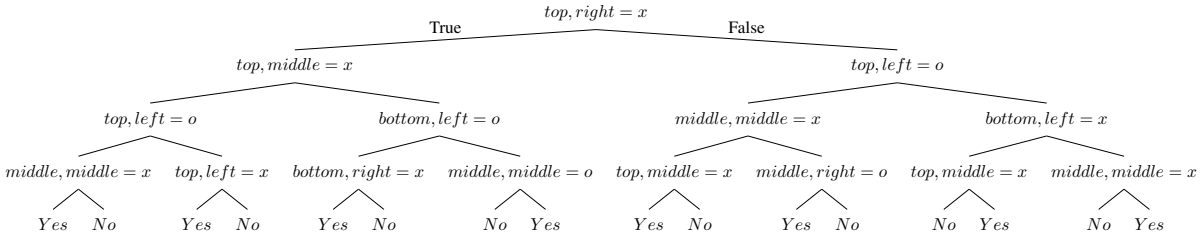
Figure 20. Seven-leaf decision trees generated by DL8.5 and GOSDT on the Monk2 dataset. With the same number of leaves, DL8.5 is 76.3% accurate, GOSDT is 79.3% accurate.

J. Algorithm

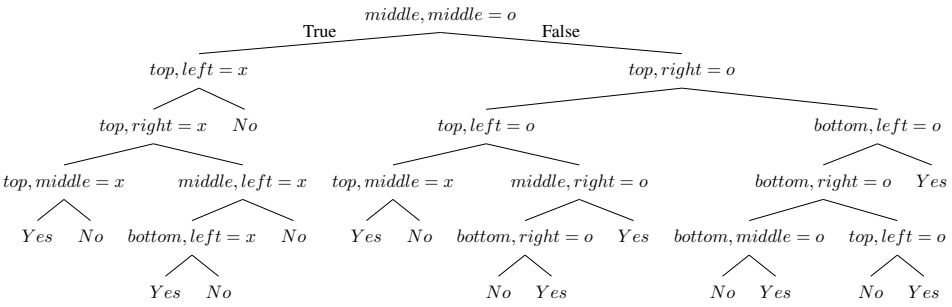
In addition to the main GOSDT algorithm (Algorithm 1), we present the subroutines *get_lower_bound* (Algorithm 2), *get_upper_bound* (Algorithm 3), *fails_bound* (Algorithm 4), and *split* (Algorithm 5) used during optimization. We also present an extraction algorithm (Algorithm 6) used to construct the optimal tree from the dependency graph once the main GOSDT algorithm completes.



(a) BinOCT (training accuracy: 82.4%, test accuracy 34.4%)



(b) DL8.5 (training accuracy: 86.9%, test accuracy: 61.5%)



(c) GOSDT (training accuracy: 90.9%, test accuracy: 70.8%)

Figure 21. 16-leaf decision trees generated by BinOCT, DL8.5, and GOSDT on the tic-tac-toe dataset. The tree generated by BinOCT includes some useless splits such as top,left=o on the bottom left and middle,middle=o near the center of the bottom row. These extra splits repeat earlier decisions from the tree, so they are clearly useless and lead to at least one empty leaf. DL8.5 also prefers to generate complete binary trees. GOSDT is more effective in generating sparse trees. With the same number of leaves, BinOCT is 82.4% accurate, DL8.5 is 86.9% accurate, and GOSDT is 90.9% accurate.

Algorithm 1 GOSDT$(R, \mathbf{x}, \mathbf{y}, \lambda)$

input: R, Z, z^-, z^+, λ *// risk, unique sample set, negative sample set, positive sample set, regularizer*
$Q = \emptyset$ *// priority queue*
$G = \emptyset$ *// dependency graph*
$s_0 \leftarrow \{1, ..., 1\}$ *// bit-vector of 1's of length U*
$p_0 \leftarrow$ FIND_OR_CREATE_NODE(G, s_0) *// node for root*
Q.push(s_0) *// add to priority queue*
while $p_0.lb \neq p_0.ub$ **do**
  $s \leftarrow Q$.pop() *// index of problem to work on*
  $p \leftarrow G$.find(s) *// find problem to work on*
  if $p.lb = p.ub$ **then**
    continue *// problem already solved*
  $(lb', ub') \leftarrow (\infty, \infty)$ *// very loose starting bounds*
  for each feature $j \in [1, M]$ **do**
    $s_l, s_r \leftarrow$ split(s, j, Z) *// create children if they don't exist*
    $p_l^j \leftarrow$FIND_OR_CREATE_NODE(G, s_l)
    $p_r^j \leftarrow$FIND_OR_CREATE_NODE(G, s_r)
   // create bounds as if j were chosen for splitting
    $lb' \leftarrow \min(lb', p_l^j.lb + p_r^j.lb)$
    $ub' \leftarrow \min(ub', p_l^j.ub + p_r^j.ub)$
   // signal the parents if an update occurred
  if $p.lb \neq lb'$ **or** $p.ub \neq ub'$ **then**
    $(p.lb, p.ub) \leftarrow (lb', ub')$
    for $p_\pi \in G$.parent(p) **do**
      // propagate information upwards
      Q.push$(p_\pi$.id, priority $= 1)$
  if $p.lb = p.ub$ **then**
    continue *// problem solved just now*
   // loop, enqueue all children that are dependencies
  for each feature $j \in [1, M]$ **do**
   // fetch p_l^j and p_r^j in case of update from other thread
    repeat line 14-16
    $lb' \leftarrow p_l^j.lb + p_r^j.lb$
    $ub' \leftarrow p_l^j.ub + p_r^j.ub$
    if $lb' < ub'$ **and** $lb' \leq p.ub$ **then**
      Q.push$(s_l$, priority $= 0)$
      Q.push$(s_r$, priority $= 0)$
return

---

subroutine FIND_OR_CREATE_NODE(G,s)
  if G.find$(s) =$ NULL *// p not yet in dependency graph*
    $p.id \leftarrow s$ *// identify p by s*
    $p.lb \leftarrow$ get_lower_bound(s, Z, z^-, z^+)
    $p.ub \leftarrow$ get_upper_bound(s, Z, z^-, z^+)
    if fails_bounds(p) **then**
      $p.lb = p.ub$ *// no more splitting allowed*
    G.insert(p) *// put p in dependency graph*
  return G.find(s)

Algorithm 2 get_lower_bound$(s, Z, z^-, z^+) \rightarrow lb$)

input: s, Z, z^-, z^+ *// support, unique sample set, negative sample set, positive sample set*
output: lb *// Risk lower bound*
// Compute the risk contributed if applying a class to every equivalance class independently
for each equivalence class $u \in [1, U]$ **define**
// Values provided in Z
$z_u^- = \frac{1}{N}\sum_{i=1}^{N} \mathbb{1}[y_i = 0 \wedge x_i = z_u]$
$z_u^+ = \frac{1}{N}\sum_{i=1}^{N} \mathbb{1}[y_i = 1 \wedge x_i = z_u]$
// Risk of assigning a class to equivalence class u
$z_u^{\min} = \min(z_u^-, z_u^+)$
// Add all risks for each class u
// Add a single λ which is a lower bound of the complexity penalty
$lb \leftarrow \lambda + \sum_u s_u z_u^{\min}$
return lb

Algorithm 3 get_upper_bound$(s, Z, z^-, z^+) \rightarrow ub$

input: s, Z, z^-, z^+ *// support, unique sample set, negative sample set, positive sample set*
output: ub *// Risk upper bound*
// Compute the risk contributed if applying a single class to all samples in s
for each equivalence class $u \in [1, U]$ **define**
// Add up the positive and negative class weights under equivalence class u
$z_u^- = \frac{1}{N}\sum_{i=1}^{N} \mathbb{1}[y_i = 0 \wedge x_i = z_u]$
$z_u^+ = \frac{1}{N}\sum_{i=1}^{N} \mathbb{1}[y_i = 1 \wedge x_i = z_u]$
// Total the positive and negatives over all classes u, choosing the smaller total as the misclassification
// Add a single λ for the complexity penalty of a leaf
$ub \leftarrow \lambda + \min(\sum_u s_u z_u^-, \sum_u s_u z_u^+)$
return ub

Algorithm 4 fails_bounds$(p) \rightarrow v$

input: p *// current problem*
output: v *// boolean indicating valid problem*
// If this expression is true then the lower bound on incremental accuracy is crossed by all descendents
// This works because since $ub - lb$ is an upperbound on incremental accuracy for any descendent
$incremental\_accuracy \leftarrow p.ub - p.lb \leq \lambda$
// If this expression is true then the lower bound on leaf classification accuracy is crossed
$leaf\_accuracy \leftarrow p.ub \leq 2\lambda$
if $(incremental\_accuracy = True) \vee (leaf\_accuracy = True)$ **then**
return $True$
return $False$

Algorithm 5 split$(s, j, Z) \rightarrow s_l, s_r$

input: s, j, Z *// support set, feature index, unique sample set*
output: s_l, s_r *// left and right splits*
// Create the left key which is the subset of s such that feature j tests negative
$s_l = \{\mathbb{1}[s_u = 1 \wedge Z_{u,j} = 0] | 1 \leq u \leq U\}$
// Create the right key which is the subset of s such that feature j tests positive
$s_r = \{\mathbb{1}[s_u = 1 \wedge Z_{u,j} = 1] | 1 \leq u \leq U\}$
return s_l, s_r

Algorithm 6 extract$(t) \rightarrow s$ *// Extract optimal tree after running the algorithm*

input: s *// Key of the problem from which we want to build a tree*
output: t *// Optimal tree*
$p \leftarrow$ FIND_OR_CREATE_NODE(G, s) *// Find the node associated to this key*
$t \leftarrow$ None *// Create a null tree*
$base\_bound \leftarrow p.ub$ *// The risk if we end this node as a leaf*
$base\_prediction \leftarrow 0$ *// The prediction if we end this node as a leaf*
$split\_bound \leftarrow \infty$ *// The risk if we split this node*
$split\_feature \leftarrow 0$ *// The index of the feature we should use to split this node*
for each feature $j \in [1, M]$ **do** *// Check all possible features*
    $s_l, s_r \leftarrow$ split(s, j, Z) *// Key of the the children for this split*
    $p_l^j \leftarrow$FIND_OR_CREATE_NODE(G, s_l) *// Find left child*
    $p_r^j \leftarrow$FIND_OR_CREATE_NODE(G, s_r) *// Find right child*
    // Check if the risk of this split is better than the best split risk so far
    if $p_l^j.ub + p_r^j.ub < split\_bound$ **then**
        $split\_bound \leftarrow p_l^j.ub + p_r^j.ub$ *// Update the best split risk*
        $split\_feature \leftarrow j$ *// Best feature index to split on which minimizes loss upper bound*
// Calculate the total positive and negative weights of each equivalence class
for each equivalence class $u \in [1, U]$ **define**
    // Values come from equivalence class matrix Z as seen in Algorithm 3
    $z_u^- = \frac{1}{N}\sum_{i=1}^{N} \mathbb{1}[y_i = 0 \wedge x_i = z_u]$ *// total negatives*
    $z_u^+ = \frac{1}{N}\sum_{i=1}^{N} \mathbb{1}[y_i = 1 \wedge x_i = z_u]$ *// total positives*
// Select only the positive and negative weights captured by s
$negatives \leftarrow \sum_u s_u z_u^-$
$positives \leftarrow \sum_u s_u z_u^+$
// Set the leaf prediction based on class with the higher selected total weight
if $negatives < positives$ **then**
    // Leaf predicts the majority class as 1 since positive weights are higher
    $base\_prediction.pred \leftarrow 1$
// Base case: If the risk of remaining as a leaf is better than splitting, remain as leaf
if $base\_bound \leq split\_feature$
    // Construct and return a leaf node
    $t.type \leftarrow leaf$
    $t.prediction \leftarrow base\_prediction$
    return t
// Recursive case: One of the splits performs better than the leaf
// Generate left and right splits based on best split feature
$s_l, s_r \leftarrow$ split$(s, split_feature, Z)$
// Recurse onto child keys to create left and right subtrees
$t_l \leftarrow$ extract(s_l)
$t_r \leftarrow$ extract(s_r)
// Construct and return a split node containing the left and right subtrees
$t.type \leftarrow tree$
$t.split \leftarrow split\_feature$
$t.left \leftarrow t_l$
$t.right \leftarrow t_r$
return t