DATA-AWARE SENSITIVITY QUANTIFICATION OF TREE ENSEMBLES

Anonymous authors
Paper under double-blind review

ABSTRACT

Decision tree ensembles (DTEs) are widely used for classification in domains such as finance and healthcare, motivating the need to formally analyze their behavior. One important property is sensitivity, where changes to a designated subset of input features can alter the model's prediction. Most existing work studies qualitative notions of sensitivity, while the few quantitative approaches measure sensitivity over the input space without accounting for how likely different inputs are under the underlying data distribution. In this work, we introduce a data-aware notion of quantitative sensitivity. Our input consists of a DTE together with a Bayesian network capturing the distribution of the underlying data. We first identify the regions of the input space that exhibit sensitivity and then weight each region by its probability, so that the measure reflects how likely sensitivity is to occur in practice, and changes with the data distribution even though the model does not. We develop a knowledge-compilation-based algorithm that computes weighted sensitivity within certified error and confidence bounds, and implement it in our tool Weighted-XCount, which scales well beyond direct weighted model counting and shows that weighted and unweighted sensitivity can differ sharply.

1 INTRODUCTION

Whether a model's undesirable behavior matters in practice depends on which inputs it actually encounters. Machine learning models are increasingly deployed in real-world applications, where the data encountered at deployment may differ substantially from the data on which the models were developed or evaluated (Quiñonero-Candela et al., 2009; Koh et al., 2021). Under such distribution shifts (Moreno-Torres et al., 2012; Shimodaira, 2000), regions of the input space that were rarely encountered under one distribution may become significantly more likely under another. Understanding the behavior of a model therefore requires not only identifying regions in which undesirable behavior can occur, but also accounting for how likely those regions are under the underlying data distribution.

One important property in this context is *sensitivity*. Given a machine learning model and a designated subset of input features, the sensitivity problem asks whether changes to these features can alter the model's prediction. Sensitivity is closely related to robustness, fairness (Dwork et al., 2012; Galhotra et al., 2017; Gupta et al., 2025), and model reliability, and has therefore received increasing attention in the formal analysis of machine learning models, both neural networks (Athavale et al., 2024; Biswas & Rajan, 2023) and tree ensembles (Kantchelian et al., 2016; Calzavara et al., 2022; Devos et al., 2021; 2024; Chen et al., 2019; Wang et al., 2020; Ignatiev et al., 2022; Izza et al., 2025). Recent work has studied sensitivity through the notion of *sensitive pairs*: pairs of inputs that differ only on the designated features but receive different predictions (Ahmad et al., 2025; Varshney et al., 2026). While the existence of such a pair establishes that the model is sensitive, it provides only a qualitative answer. A single sensitive pair may arise in a region of the input space that is rarely, or perhaps never, encountered in practice.

This motivates the study of *quantitative sensitivity*: rather than asking merely whether sensitivity exists, one asks how prevalent sensitivity is across the input space, in the spirit of quantitative verification of neural networks (Downing et al., 2024; Kim et al., 2025). Existing quantitative notions (Naik et al., 2026) measure the size of the sensitive portion of the input space, for example by counting sensitive regions or measuring their volume. However, such notions implicitly treat differ-

Reviewers: please read the Reviewer Guidelines (iclr.cc/Conferences/2027/ReviewerGuidelines) and the AI Policy for Reviewers (iclr.cc/Conferences/2027/AIPolicyForReviewers). If you used AI to expand, edit, or polish your review, please provide the input text to the LLM. Better still, consider skipping the LLM and submitting your original text: we, and the authors, are much more interested in your unedited thoughts than in what an LLM has to say. AI-assisted or not, you are putting your name and reputation behind your review: LLM-generated falsehoods, hallucinations or misrepresentations are subject to disciplinary action, which may include desk-rejecting all papers you have authored.

ent parts of the input space uniformly. This can be misleading when the underlying data is highly non-uniform. Two sensitive regions of equal size may have very different practical significance if one contains substantial probability mass while the other is almost never encountered, and this difference can change with the data distribution even when the model does not: a sensitive region may account for only 1% of the probability mass under the training distribution but 20% under a shifted deployment distribution, while an unweighted measure based on the size of the region stays the same. An input-space measure alone thus cannot capture how likely sensitivity is to arise for the population of interest. This suggests a *data-aware* notion of quantitative sensitivity in which sensitive regions are weighted according to their probability under the underlying data distribution.

To make this idea computationally tractable, we consider a discretized input space and associate each discretized input, or region, with a probability mass. The resulting measure does not simply count how much of the input space is sensitive; instead, it measures how much probability mass is associated with sensitivity. In particular, the same model can have different weighted sensitivities under different underlying distributions, naturally capturing the effect of a change in the data distribution, which we illustrate in Section 5.

In this paper we focus on decision tree ensembles (DTEs), which are among the most widely used machine-learning models for structured and tabular data (Blockeel et al., 2023). They are trained with algorithms such as XGBoost (Chen & Guestrin, 2016), LightGBM (Ke et al., 2017), and random forests (Breiman, 2001; Pedregosa et al., 2011), and are routinely used in domains including finance and healthcare (Madaan et al., 2021; Chang et al., 2018; Ghiasi & Zendehboudi, 2021). Their discrete and highly structured nature also makes them particularly amenable to symbolic reasoning and formal verification. This enables us not only to estimate sensitivity empirically from samples, but to compute it with formal guarantees.

Concretely, our input consists of a DTE together with a Bayesian network (Koller & Friedman, 2009) representing the joint distribution over the input features. We first identify the regions of the input space that exhibit sensitivity and then weight these regions according to their probability under the Bayesian network. This gives rise to the *data-aware quantitative sensitivity problem*: given a DTE and a Bayesian network describing the input distribution, compute the total probability mass of the input space that exhibits sensitivity with respect to a designated set of features. Our main contributions are as follows:

- We introduce a data-aware notion of quantitative sensitivity for decision tree ensembles and formulate the resulting weighted sensitivity problem for a DTE together with a Bayesian network that captures dependencies between input features. Sensitive regions are weighted by their probability, so the measure responds to the data distribution even when the model is unchanged.
- We develop a knowledge-compilation-based algorithm for computing weighted sensitivity with certified error and confidence bounds. It extends the existing Algebraic Decision Diagram (ADD)-based, compositional approach for counting sensitive regions to probability mass, adapting a union-counting analysis from cardinality to the weighted setting to obtain an (ϵ, δ) guarantee.
- We implement our approach in the tool WEIGHTED-XCount and evaluate it on DTEs of varying sizes and depths across multiple datasets. It solves substantially more instances than a direct weighted-model-counting baseline and, on the instances where the exact count is available, its measured errors stay well within the guaranteed bound.
- We show empirically that the weights matter: for a fixed model, weighted sensitivity differs sharply from the unweighted count, and it drops under a controlled shift of the data distribution away from the sensitive regions, which an unweighted count cannot reflect.

Related work. *Verification of tree ensembles and neural networks.* For decision tree ensembles, robustness verification has used abstract interpretation (Ranzato & Zanella, 2020), dynamic programming (Wang et al., 2020) and clique-based search (Devos et al., 2021; 2024), while sensitivity has been checked through pseudo-Boolean (Ahmad et al., 2025) and MILP (Varshney et al., 2026) encodings. These approaches decide whether a sensitive pair exists rather than how much of the input space is sensitive. For neural networks, quantitative verification has been studied (Downing et al., 2024; Kim et al., 2025), for example by certifying the percentage of inputs that are fair or un-

fair (Kim et al., 2025). The counting approach for DTEs (Naik et al., 2026) quantifies sensitivity but gives every region the same weight. Closest to our data-aware view is fairness verification under an input distribution: Ghosh et al. (2022) model correlations between features with a Bayesian network and compute group and causal fairness metrics of linear classifiers. We instead consider decision tree ensembles and quantify sensitivity as the probability mass of the sensitive regions, with certified error and confidence bounds.

Model counting and knowledge compilation. Exact model counters such as d4 (Lagniez et al., 2024) and Ganak (Soos & Meel, 2025; Sharma et al., 2019) work by compiling the formula into a form in which counting is easy, while approximate counters such as ApproxMC (Yang & Meel, 2023; Soos et al., 2020) scale further by providing PAC guarantees instead of exact counts. Decision diagrams and related normal forms have also been used for weighted model counting (Dudek et al., 2020; de Colnet et al., 2024), and Bayesian networks can be encoded as weighted CNF formulas (Chavira & Darwiche, 2006), which is how we construct our baseline. Our algorithm lifts the compositional approach of Naik et al. (2026), which uses ADDs, splits the problem into subproblems and combines their counts by approximate counting of the union of sets (Meel et al., 2021; Soos et al., 2024), from counting regions to computing probability mass under a Bayesian network.

2 PRELIMINARIES

In this work, we consider a supervised learning setting where an input vector $\mathbf{x} = (x_1, \ldots, x_k) \in \mathbb{R}^k$ over a set $F = \{1, \ldots, k\}$ of features is mapped to a real output by a decision tree ensemble (DTE).

Decision tree ensembles. A DTE $\mathcal{T} = \{T_1, \ldots, T_n\}$ is a collection of n decision trees (Breiman, 2001). Each tree $T_i : \mathbb{R}^k \to \mathbb{R}$ is a rooted binary tree whose **internal nodes** split on a guard predicate $x_f < \theta$, for a feature $f \in F$ and a threshold $\theta \in \mathbb{R}$, and whose **leaf nodes** store a constant prediction value $v_l \in \mathbb{R}$. The output $T(\mathbf{x})$ of a tree is obtained by traversing it from the root to a leaf according to the guards. We focus on additive ensembles, such as gradient boosted trees (Ke et al., 2017), whose prediction is the sum $\mathcal{T}(\mathbf{x}) = \sum_{i=1}^{n} T_i(\mathbf{x})$. Random forests (Breiman, 2001), which average the trees, differ only by a factor of $1/n$, which is absorbed into the gap margin G below.

Model sensitivity. We study how changes to a designated subset $S \subseteq F$ of sensitive features affect the prediction, where the features in $F \setminus S$ are non-sensitive. In the context of individual fairness (Dwork et al., 2012), S models protected attributes such as race, gender or age. Distances are Hamming distances: for two vectors $\mathbf{a}$ and $\mathbf{b}$ of equal length, $\mathsf{HammingDist}(\mathbf{a}, \mathbf{b})$ is the number of positions at which they differ.

Definition 1 (Sensitivity)**.** A DTE $\mathcal{T}$ is **sensitive** with respect to S if there exist inputs $\mathbf{x}, \mathbf{x}' \in \mathbb{R}^k$ that agree on all features in $F \setminus S$, are within Hamming distance d of each other (on the Boolean encoding of Section 3), and satisfy $|\mathcal{T}(\mathbf{x}) - \mathcal{T}(\mathbf{x}')| > G$, where d is a distance threshold and $G \geq 0$ is a user-specified prediction-gap margin.

Algebraic decision diagrams. An *Algebraic Decision Diagram* (ADD) (Bahar et al., 1997) generalizes a Binary Decision Diagram (BDD) (Bryant, 1986) to represent a function $f : \{0,1\}^M \to \mathbb{R}$ over Boolean variables $b_1, \ldots, b_M$ (the encoding variables of Section 3) as a rooted directed acyclic graph. Each non-terminal node is labeled by a variable and has a low edge, followed when the variable is 0, and a high edge, followed when it is 1; each terminal node carries a real value; and variables appear in a fixed order along every path. The value $f(\alpha)$ of an assignment α is the label of the terminal reached by following the edges from the root. The core advantage of ADDs in our context is the efficient APPLY operator, which performs pointwise arithmetic on the symbolic structures. We construct the ADD of the entire ensemble by summing the ADDs of the individual trees, $\mathcal{T}_{\text{ADD}} = \sum_{i=1}^{n} T_{i,\text{ADD}}$.

Bayesian networks. A Bayesian network $\text{BN} = (\mathcal{G}, \Theta)$ (Koller & Friedman, 2009) consists of a directed acyclic graph $\mathcal{G}$ whose nodes are random variables $b_1, \ldots, b_M$, and a collection Θ of conditional probability distributions $\Pr(b_j \mid \text{Pa}(b_j))$, where $\text{Pa}(b_j)$ denotes the parents of b_j in $\mathcal{G}$. It represents the joint distribution $\Pr_{\text{BN}}(b_1, \ldots, b_M) = \prod_{j=1}^{M} \Pr(b_j \mid \text{Pa}(b_j))$. We use a BN to model the distribution of the input data over the Boolean encoding of the features introduced in Section 3.

3 PROBLEM STATEMENT

In this section, we recall the counting notion of sensitivity of Naik et al. (2026) and then develop its data-aware, weighted version, which takes the input data distribution into account. Decision tree ensembles, by their design, split the input space of the model into regions such that the output of the model for all points belonging to a region is the same. This natural property of these models can be used to ask questions of a quantitative variety as the input space is discretized into a finite number of regions. Given a decision tree model and its corresponding set of guard predicates, for a feature $f \in F$ let Γ_f be the ordered set of thresholds $\theta_{f,1}, \theta_{f,2}, \ldots, \theta_{f,m_f}$ which appear in the guards of the ensemble on feature f, where $m_f = |\Gamma_f|$. Furthermore, let B_f be the ordered set $(b_{f,i})_{i=1}^{m_f}$ of Boolean variables corresponding to feature f, where $b_{f,i}$ represents the decision $x_f < \theta_{f,i}$. These variables must satisfy the monotonicity constraint: $b_{f,1} \Rightarrow b_{f,2} \Rightarrow \cdots \Rightarrow b_{f,m_f}$. This ensures that Boolean assignments correspond to valid numeric orderings.

Let B be the ordered set of all Boolean variables corresponding to all features in F. Formally, $B = (B_1, B_2, \ldots, B_k)$. Let $M = |B| = \sum_{f \in F} m_f$. Thus, any $\mathbf{b} \in \{0,1\}^M$ is an M dimensional vector $(\mathbf{b}_{f,i})_{i \in [1,m_f], f \in F}$.

Definition 2. We define a mapping function $\phi : \mathbb{R}^k \to \{0,1\}^M$ that maps a input vector $\mathbf{x}$ (consisting of k feature values) to an output Boolean vector $\mathbf{b}$ (consisting of M Boolean variables) as follows : $\phi(\mathbf{x})_{f,i} = 1 if x_f < \theta_{f,i}$, and 0 otherwise. We also define a valuation map $V : \{0,1\}^M \to \mathbb{R}$ be defined as: $V(\mathbf{b}) = \mathcal{T}(\mathbf{x})$ for some $\mathbf{x}$ such that $\mathbf{b} = \phi(\mathbf{x})$. Note that V is well-defined.

From the definition of sensitivity, we want to check if our model is sensitive to a subset of features S. To operationalize this, we consider pairs of inputs $\mathbf{x}$ and $\mathbf{x}'$ and their corresponding Boolean encodings $\mathbf{b} = \phi(\mathbf{x})$ and $\mathbf{b}' = \phi(\mathbf{x}')$. The following three conditions must be satisfied for $\mathbf{x}$ and $\mathbf{x}'$ to witness a sensitivity violation: **(C1)** All Boolean variables of non-sensitive features must be identical between $\mathbf{b}$ and $\mathbf{b}'$: $\bigwedge_{i \in [1,m_f], f \in F \setminus S} \mathbf{b}_{f,i} = \mathbf{b}'_{f,i}$ **(C2)** The sensitive features must differ as quantified by a Hamming distance threshold d: $\mathsf{HammingDist}(\mathbf{b}, \mathbf{b}') \leq d$ **(C3)** A sensitivity violation occurs when the ensemble's predictions for $\mathbf{x}$ and $\mathbf{x}'$ differ by at least a user-specified margin G: $|V(\mathbf{b}) - V(\mathbf{b}')| > G$

Each input $\mathbf{x}$ is mapped to a Boolean vector $\mathbf{b}$, dividing the input space into a finite set of regions, each corresponding to a Boolean vector. We enumerate all regions for which there exists at least one other region such that this pair satisfies the notion of sensitivity defined above.

Definition 3. We can define the set R^* of all such sensitive regions as follows:

$$R^* = \left\{ \mathbf{b} \in \{0,1\}^M \;\middle|\; \exists\, \mathbf{b}' \in \{0,1\}^M : \mathbf{C1} \wedge \mathbf{C2} \wedge \mathbf{C3} \right\}$$

Definition 4. Given a DTE $\mathcal{T}$ over a set of input features F, with $S \subseteq F$, $d \in \mathbb{N}$ and $G \in \mathbb{R}_{\geq 0}$, we define the counting objective of Naik et al. (2026) as $C(S, d, G) = |R^*|$

Since every Boolean vector $\mathbf{b}$ uniquely identifies a discrete region of the input space, we use the terms interchangeably.

Now, we propose to modify the notion of count defined above to a weighted version that takes into account the distribution of the input data. In many real-world scenarios, not all regions of the input space are equally likely to occur. Therefore, it is important to consider the probability mass associated with each region when quantifying sensitivity. Let $\Omega \subseteq \{0,1\}^M$ be the set of all regions $\mathbf{b}$ as defined above. We assume that the input data distribution is modeled by a Bayesian network BN over the Boolean variables B. Let the Bayesian network characterise a probability distribution over the regions, with the weight of a region $\mathbf{b}$ given by $\Pr_{\mathrm{BN}}(\mathbf{b})$.

Definition 5. The **weighted counting objective** can be defined as the total probability mass of all sensitive regions:

$$\boxed{C_w \;=\; \Pr_{\mathrm{BN}}(R^*) \;=\; \sum_{\mathbf{b} \in R^*} \Pr_{\mathrm{BN}}(\mathbf{b}).}$$

where R^* is the set of all sensitive regions. C_w is a probability mass in $[0,1]$.

Algorithm 1 Weighted-XCount($\mathcal{T}, F, S, d, G, \epsilon, \delta, BN$)

1: $n \leftarrow |\mathcal{T}|$, $k \leftarrow |F|$, BitMasks $\leftarrow \emptyset$, Probs $\leftarrow \emptyset$
2: **for** $f \in S$ **do**
3: BitMasks$(f) \leftarrow$ BitMaskGen(m_f)
4: GlobalMaskSet $\leftarrow \prod_{f \in S}$ BitMasks(f) ▷ $\prod$ is concatenation of bit-vectors
5: Probs $\leftarrow \{(mask_1, mask_2) | mask_1, mask_2 \in$ GlobalMaskSet $\wedge$ HammingDist$(mask_1, mask_2) \leq d\}$
6: Construct the BDD for each pair and let Probs $= (q_1, \ldots, q_m)$
7: If every subproblem has zero mass, **return** 0
8: Reorder Probs so that the first subproblem has positive mass
9: $X \leftarrow \emptyset$ ▷ X is a multiset
10: Thresh $\leftarrow \max\Big(24 \cdot \frac{\ln(24/\delta)}{(1-\epsilon)\epsilon^2},\ 6\big(\ln\frac{6}{\delta} + \ln m\big)\Big)$
11: $U_1 \leftarrow$ ProcessSubproblem$(\mathcal{T}, q_1, G)$
12: $p \leftarrow \frac{\text{Thresh}}{\text{BNmass}(U_1, q_1, \text{BN})}$ ▷ The denominator is positive after the preceding filter
13: **for** $q_i \in$ Probs **do**
14: $U_i \leftarrow$ ProcessSubproblem$(\mathcal{T}, q_i, G)$ ▷ U_i is the BDD for the ith subproblem
15: $t \leftarrow$ BNmass(U_i, q_i, BN)
16: **for** $\mathbf{b} \in X$ **do**
17: **if** $\mathbf{b}$ belongs to the full-assignment set induced by (U_i, q_i) **then**
18: remove $\mathbf{b}$ from X
19: $N_i \leftarrow$ Poisson$(t \cdot p)$
20: **while** $N_i + |X| \geq$ Thresh **do**
21: Remove each assignment from X with 0.5 probability
22: $p \leftarrow p/2$
23: $N_i \leftarrow$ Binomial$(N_i, 0.5)$
24: $Samples \leftarrow$ SampleBDD$(U_i, N_i, q_i, \text{BN})$
25: X.Append$(Samples)$
26: **return** $|X|/p$

Unlike the counting objective, C_w weighs each region by its probability instead of counting it as one, so the unweighted counting techniques of Naik et al. (2026) do not apply directly; Section 4 describes how we adapt them.

4 Approximate Weighted Counting Algorithm

In this section, we describe the primary technical contributions of our work : a procedure Weighted-XCount that takes as input decision tree ensemble $\mathcal{T} = \{T_i\}_{i=1}^{n}$ having n binary trees over a feature space F of k features and a Bayesian network BN, and computes C_w, i.e, the total probability mass of the regions of input space that are sensitive for a given feature set $S \subseteq F$, given a maximum perturbation of d bits and a nonnegative difference margin G. We build on the ADD-based approach outlined in Naik et al. (2026) - as the change is restricted to the variables of the sensitive features (by a given bit-distance), we break the computation into subproblems, where each subproblem is a pair of bit vectors, with each bit vector being an assignment over variables of sensitive features, and the two differing in d bits. Moreover, as these bits are fixed for each subproblem, certain branches of the ADDs become unreachable, hence the computations can be performed on a pruned version of ADDs. Furthermore, instead of exact weighted counting over each subproblem, we adopt an approximate weighted counting approach where we compute an estimate $\widehat{C_w}$ of the exact C_w by taking the weighted count of the union of all solutions over all subproblems within a (ϵ, δ) guarantee where $\widehat{C_w}$ we get is within a factor of $(1 \pm \epsilon)$ of the actual weighted sum with at least $(1 - \delta)$ confidence.

4.1 Description of the algorithm

In Algorithm 1, we present our main method Weighted-XCount for computing the weighted count of sensitive regions of the input space. In lines 2-5, our algorithm computes the set of all bit-masks for each sensitive feature. Since the sensitive features are subject to monotonicity constraints, if there are m_f bits of a particular feature f, then there are $(m_f + 1)$ possible boolean evaluations. A

set of these evaluations, referred to as bit-masks, in the form of bit-vectors is computed and stored in BitMasks. Then it takes a cross-product of these bit-masks to get a global set of bit-masks, which is a concatenation of the bit-masks for each feature. Then we compute the set of all pairs of bit-masks that are within a Hamming distance of d from each other, which forms the set of subproblems Probs.

We let m denote the number of subproblems. If they all have zero mass, the algorithm returns 0. Otherwise, we process a positive-mass subproblem first so that the initial sampling rate is well-defined. The multiset X stores sampled full assignments, including both sensitive and non-sensitive variables.

For each pair q_i, ProcessSubproblem returns a BDD U_i over the unfixed variables. Together, (U_i, q_i) induces a set of full assignments; BNmass computes its total BN mass. Before drawing new samples, the algorithm removes from X every copy of a full assignment in this induced set. It then draws N_i new assignments with replacement, each proportional to its joint BN mass. Removing the overlap before resampling is what makes X represent the union, rather than the sum, of the subproblem masses. If the provisional sample count reaches Thresh, the algorithm halves p and independently retains each existing or pending sample with probability $1/2$. Finally, it returns $|X|/p$ as the estimate $\widehat{C_w}$.

Due to the weighted nature of the problem, we cannot simply count the number of satisfying assignments. Instead, we need to account for the weights of the assignments when sampling and estimating the total count. If the total weight of the satisfying assignments is small, we cannot sample enough satisfying assignments if the initial sampling rate is small; hence we initialize p to be large if the total weight of the first subproblem is small. Since the total weight of the satisfying assignments of the subproblems only grows as we process more subproblems, we can keep halving p to ensure that we do not exceed the threshold Thresh for the size of X. The value p remains at the appropriate value for the rest of the subproblems. In Appendix B, we provide the implementation details of the subroutines used in Algorithm 1.

4.2 Theoretical Guarantees

The proof below adapts the Poissonized union-counting analysis of Soos et al. (2024) from unweighted cardinality to probability mass. The adaptation replaces uniform sampling by sampling proportional to $\Pr_{\mathrm{BN}}(\mathbf{b})$; we give the argument in full because this weighted step is essential here.

Theorem 1. *For $G \geq 0$ and $0 < \epsilon, \delta < 1$, let $\widehat{C_w}$ be the output of Weighted-*XCount*. Then* $\Pr[(1-\epsilon)C_w \leq \widehat{C_w} \leq (1+\epsilon)C_w] \geq 1 - \delta$.

Proof. Each Boolean assignment $\mathbf{b}$ represents one input region and has probability $\Pr_{\mathrm{BN}}(\mathbf{b})$. For any set of regions A, let $w(A) = \sum_{\mathbf{b}\in A} \Pr_{\mathrm{BN}}(\mathbf{b})$ denote its total probability mass.

Let A_i be the set of complete regions represented by the BDD U_i and its mask pair q_i. Thus, A_i is the set handled by the ith subproblem. The variable-elimination computation in Algorithm 8 gives $\mathsf{BNmass}(U_i, q_i, \mathrm{BN}) = w(A_i)$, and SampleBDD draws each $\mathbf{b} \in A_i$ with probability $\Pr_{\mathrm{BN}}(\mathbf{b})/w(A_i)$. After processing the first i subproblems, let $R_i := A_1 \cup \cdots \cup A_i$ be the set of regions encountered so far, and let $W_i := w(R_i)$ be their total mass. We first show that $R_m = R^*$. Consider a region $\mathbf{b} \in R_m$. By the definition of R_m, the region belongs to A_i for some i. The mask pair q_i satisfies conditions C1 and C2. Moreover, U_i certifies that the signed difference between the predictions of the two regions is greater than G. Since $G \geq 0$, their absolute difference is also greater than G. Therefore, $\mathbf{b} \in R^*$.

For the other direction, consider a sensitive region $\mathbf{b} \in R^*$. By definition, there is a region $\mathbf{b}'$ such that the pair $(\mathbf{b}, \mathbf{b}')$ satisfies C1–C3. The masks of $\mathbf{b}$ and $\mathbf{b}'$ occur in both orders in Probs. Since their absolute prediction difference is greater than G, one of these orders has signed difference greater than G. For that ordered pair, ProcessSubproblem places both endpoints, including $\mathbf{b}$, in A_i. Hence, $\mathbf{b} \in R_m$. We have thus shown that $R_m = R^*$. This symbolic construction follows Naik et al. (2026).

It now follows directly that $W_m = w(R_m) = w(R^*) = C_w$. If $C_w = 0$, every subproblem has zero mass. In this case, the algorithm returns 0, and the theorem follows. For the rest of the proof, assume that $C_w > 0$.

Let $T = \mathsf{Thresh}$. Because the first subproblem has positive mass, the initial sampling rate is well-defined as $g = T/w(A_1)$. The algorithm only halves this rate. Therefore every possible sampling rate has the form $p_j = g/2^j$ for some integer $j \geq 0$.

We next couple all possible executions of the algorithm. For every i and $\mathbf{b} \in A_i$, independently generate $Z_i(\mathbf{b}) \sim \mathsf{Poisson}(g \Pr_{\mathrm{BN}}(\mathbf{b}))$ copies and attach an infinite sequence of independent fair bits to every copy. After the first i subproblems, use the copies generated by the most recent subproblem that contains $\mathbf{b}$. At rate p_j, retain a copy exactly when its first j bits are all 1; call the resulting multiset $X_i^{(j)}$.

Processing A_i replaces all older copies of its regions by the copies generated for subproblem i. Moving from rate p_j to p_{j+1} uses the next fair bit to thin the retained copies. Poisson splitting shows that this construction is equivalent to drawing $N_i \sim \mathsf{Poisson}(p_j w(A_i))$ followed by N_i weighted samples from A_i. It also shows that the binomial thinning of N_i in the algorithm has the same distribution. Thus, the actual adaptive execution follows a path through the coupled family $\{X_i^{(j)}\}_{i,j}$.

For every fixed i, j and $\mathbf{b} \in R_i$, the number of copies of $\mathbf{b}$ in $X_i^{(j)}$ is therefore an independent random variable

$$K_i^{(j)}(\mathbf{b}) \sim \mathsf{Poisson}\big(p_j \Pr_{\mathrm{BN}}(\mathbf{b})\big). \tag{1}$$

This is the weighted analogue of the Poisson splitting and thinning argument used by Soos et al. (2024).

In particular, sums of independent Poisson variables are Poisson. Hence, $|X_i^{(j)}| \sim \mathsf{Poisson}(p_j W_i)$. The algorithm adaptively chooses the final rate, so we must account for that choice rather than applying a tail bound only at a fixed rate. Couple all the multisets $X_i^{(j)}$ using the sampling and thinning operations above. Let E_j be the event that the final rate is p_j. Also, let B_j be the event that $|X_m^{(j)}|$ does not belong to $[(1-\epsilon)p_j W_m, (1+\epsilon)p_j W_m]$. If E_j occurs, the returned estimate is $|X_m^{(j)}|/p_j$; therefore the failure event is $\bigcup_{j\geq 0}(E_j \cap B_j)$.

Let j^* be the smallest integer for which $p_{j^*} < T/(4W_m)$. Because $g = T/w(A_1) \geq T/W_m$, we have $j^* > 2$. Splitting the possible final rates at j^* and applying a union bound gives

$$\Pr[\text{failure}] \leq \sum_{j=0}^{j^*-1} \Pr[B_j] + \Pr\left[\bigcup_{j\geq j^*} E_j\right]. \tag{2}$$

We bound the two terms separately. For $j < j^*$, the Poisson-size identity above, the choice of j^*, and the standard two-sided Poisson Chernoff bound imply

$$\sum_{j=0}^{j^*-1} \Pr[B_j] \leq 2e^{-\epsilon^2 T/12} + 2e^{-\epsilon^2 T/6} + \cdots < 4e^{-\epsilon^2 T/12} \leq \frac{\delta}{6}. \tag{3}$$

The final inequality uses $T \geq 12\ln(24/\delta)/\epsilon^2$, which is weaker than the threshold used in Algorithm 1.

For the second term in equation 2, the final rate can be at most p_{j^*} only if, for some $i \in \{1, \ldots, m\}$, the thinning condition is triggered while the rate is p_{j^*-1}. By the minimality of j^*, $p_{j^*-1} < T/(2W_m)$. Thus, the provisional multiset after incorporating subproblem i has Poisson mean $p_{j^*-1}W_i < T/2$. The one-sided Poisson Chernoff bound shows that the probability of thinning at this rate for subproblem i is at most $e^{-T/6} \leq \delta/(6m)$. Here the last inequality follows from $T \geq 6(\ln(6/\delta) + \ln m)$. A union bound over the m subproblems yields $\Pr[\bigcup_{j\geq j^*} E_j] \leq \delta/6$. Combining this bound with equation 2 and equation 3, the failure probability is at most $\delta/3 \leq \delta$. Since $W_m = C_w$, the returned value satisfies the claimed multiplicative bound with probability at least $1 - \delta$. □

5 Experiments

We now evaluate the performance and accuracy of our proposed method, Weighted-XCount, on a set of decision tree ensemble models trained on tabular datasets. Through the experimental evaluation,

we want to address the following research questions: **RQ1.** How scalable is Weighted-XCount with respect to exact weighted model counting approaches? **RQ2.** Does Weighted-XCount satisfy the theoretical guarantees on error? **RQ3.** What is the importance of assigning weights to the regions?

Implementation details. We implemented our tool Weighted-XCount in C++, using the CUDD package for the implementation of ADDs and BDDs. The tool takes the following command inputs: a decision tree ensemble in the form of a JSON file, the input feature(s) of the ensemble to test for sensitivity (S), the precision that is required of the leaf values of the ensemble, the gap margin (G), the bit distance allowed between sensitive features (d), the confidence parameter (δ) and error parameter (ϵ). The output of the tool is the sum estimate $\widehat{C_w}$, within a factor of $(1 \pm \epsilon)$ with a confidence of (1-δ) of the actual sum C_w as outlined in Section 4.

Benchmarks. We evaluated our approach on eight standard tabular datasets: *Diabetes*, *Adult*, *Covtype*, *Protein structure*, *Webspam*, *Wine-quality*, *Supersymmetry*, and *California housing*. To construct a comprehensive benchmark suite, we trained multiple distinct ensemble models for each dataset. We varied the ensemble size from 10 to 100 in increments of 10 (10 sizes), and tree depths from 3 to 6 (4 depths). For each model generated, we randomly chose a set of 8 features present in the model as the features to be tested for sensitivity analysis. This leads to total benchmark instances of $10 \times 4 \times 8 = 320$ for each dataset (there are 319, 319, 317, 319 instances for *Diabetes*, *Supersymmetry*,*California housing* and *Protein structure*, respectively, as smaller ensemble models of these datasets have fewer than 8 features, in such case we have picked all features present in the model without any randomization). Table 1 lists out all the datasets. We also train a Bayesian network on the training split of each dataset using the pgmpy python library (Ankan & Textor, 2024) with standard learning algorithms.

Dataset	Instances
Diabetes	319
Adult	320
Covtype	320
Webspam	320
Wine-quality	320
Supersymmetry	319
California housing	317
Protein structure	319

Table 1: Benchmark instances for each dataset

All experiments were conducted on a high-performance server with an **AMD EPYC 7742 64-Core Processor** and 629GiB of RAM, running **Ubuntu 22.04.5 LTS**. To ensure fair comparison and resource isolation, we enforced a 4GB memory limit per instance and a strict timeout of 1800 seconds.

Comparisons baseline. We compare the performance of our method with respect to a weighted model counting pipeline, which is constructed as follows: we encode the given Bayesian network as a weighted CNF (Chavira & Darwiche, 2006), conjoining it with CNF encoding of the sensitivity problem and giving it to weighted model counter PySDD (Meert & Choi, 2017). For consistency, we utilized a gap margin $G = 1$, bit-distance $d = 1$, $\epsilon = 0.1$, $\delta = 0.1$, and set the leaf value precision to 3 decimal places across all experiments. However, for the datasets *California housing*, *Webspam* and *Wine-quality* we used $G = 0.1$, and for *Supersymmetry* $G = 0.01$, because the leaf values of these datasets are much smaller, so that a gap margin of 1 leaves no possible solutions.

Performance results. We compare Weighted-XCount against the weighted-CNF baseline, which computes exact counts. As shown in Figure 1, Weighted-XCount completes far more instances within the time and memory limits: out of 2,554 instances it solves 1,669, whereas the baseline solves only 84. This answers RQ1: our approximate method is substantially more scalable than exact weighted model counting. It is not a complete answer to scalability, however: 885 of the 2,554 instances remain unsolved within the limits, the slowest solved instances approach the 1800-second timeout, and we do not break the results down by ensemble size or depth.

Accuracy wrt approximation guarantees. Figure 2 shows the relative error of the approximate weighted count given by Weighted-XCount with respect to the exact count, which is available only for the 84 instances solved by the baseline. The error is well within the given error bound of 10% ($\epsilon = 0.1$): 83 out of 84, i.e., 98.81%, of these instances report an error within 2%. This answers RQ2: on these instances the observed errors respect the (ϵ, δ) guarantee. It does not tell us how the error behaves on the larger instances, where no exact count is available.

Importance of the weights in counting. To demonstrate the importance of taking weighted count, we consider the following scenario : consider a dataset and a decision tree model trained on it.

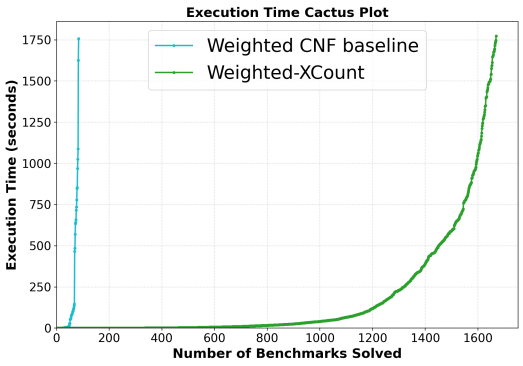


Figure 1: Cactus plots comparing execution time across all benchmarks.

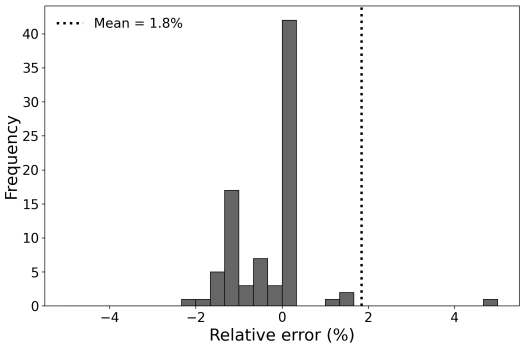


Figure 2: Error with respect to the exact weighted count

We calculate the number of sensitive regions for this model. Considering this count against the total number of regions, we find that the model is X% sensitive. Now, as we identified the sensitive regions in the previous computation, we trim the dataset to only include points which do not fall in the sensitive regions. We then train a Bayesian network on this trimmed dataset and use it to assign weights to the regions. Now we take the decision tree model and the Bayesian network to calculate the weighted count of the sensitive regions. As the weights are derived from the Bayesian network and add up to 1, the weighted count that we get lies between 0 and 1. From the weighted count, we find that the model is Y% sensitive. We should observe that Y is significantly less than both X and the weighted count obtained with a Bayesian network trained on the untrimmed data, as the importance of the sensitive regions has been reduced (captured by the Bayesian network trained on the trimmed dataset). We empirically test this by taking the *California housing* dataset and training a decision tree ensemble of size 3 and depth 3 on it. Figure 3 reports, for each sensitive feature, the unweighted count, the weighted count under a Bayesian network trained on the untrimmed data, and the weighted count under a Bayesian network trained on the trimmed data. Weighting alone changes the measure substantially (for feature 0, 47.00% unweighted versus 28.80% weighted) and can increase it (for feature 5, 3.42% versus 4.59%); trimming then lowers the weighted count for every feature. This answers RQ3: the weights change the measured sensitivity, and the weighted measure responds to a change in the data distribution, which an unweighted count cannot do.

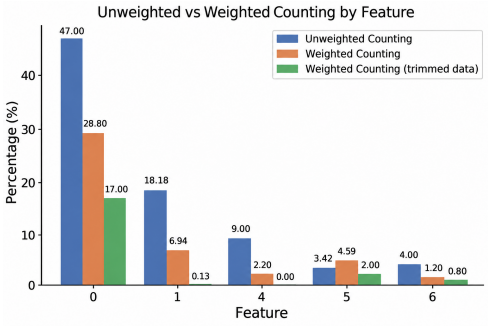


Figure 3: Comparison of unweighted and weighted counts (full and trimmed dataset)

Hyperparameters. We also investigated how Weighted-XCount varies with training hyperparameters related to overfitting. The weighted count decreases as the learning rate decreases, while gamma, alpha and minimum child weight show no significant trend (Appendix C).

6 Conclusion

We introduced data-aware quantitative sensitivity for decision tree ensembles, which weights each sensitive region by its probability under a Bayesian network, and gave an algorithm that computes it with certified error and confidence bounds. The algorithm solves substantially more instances than a direct weighted-model-counting baseline, and our experiments show that weighting matters: weighted and unweighted sensitivity can differ sharply, and only the weighted measure responds to a change in the data distribution. Future work includes evaluating on naturally occurring distribution shifts and scaling to larger ensembles.

AI USE STATEMENT

In this work, we used generative AI tools for aiding and polishing writing and drafting sections of the paper. We have not used generative AI tools for retrieval and discovery, or for research ideation, or for proving mathematical claims, and generating synthetic datasets is not applicable to this work. Additionally, we used generative AI tools for generating experimental set-up scripts and testing of the code. We have reviewed all AI-assisted work by undertaking sanity checks through manually written test-cases for the code, as well as manually verified the AI-drafted sections of the paper. We take responsibility for the final content of this work, including text, claims or artifacts produced with the aid of generative AI.

ETHICS STATEMENT

The authors believe that, though the paper addresses fairness, we do not see any immediate ethics issues.

REPRODUCIBILITY STATEMENT

Complete proof of the theoretical claim made in the paper is present in the paper. We have also disclosed the list of datasets and other softwares/libraries that we have used in this work. We will also be making the code open source in a public repository in case of receiving acceptance.

REFERENCES

Arhaan Ahmad, Tanay Vineet Tayal, Ashutosh Gupta, and S. Akshay. Sensitivity verification for additive decision tree ensembles. In *The Thirteenth International Conference on Learning Representations, ICLR 2025, Singapore, April 24-28, 2025*. OpenReview.net, 2025. URL https://openreview.net/forum?id=h0vC0fm1q7.

Ankur Ankan and Johannes Textor. pgmpy: A python toolkit for bayesian networks. *Journal of Machine Learning Research*, 25(265):1–8, 2024. URL http://jmlr.org/papers/v25/23-0487.html.

Anagha Athavale, Ezio Bartocci, Maria Christakis, Matteo Maffei, Dejan Nickovic, and Georg Weissenbacher. Verifying global two-safety properties in neural networks with confidence. In Arie Gurfinkel and Vijay Ganesh (eds.), *Computer Aided Verification - 36th International Conference, CAV 2024, Montreal, QC, Canada, July 24-27, 2024, Proceedings, Part II*, volume 14682 of *Lecture Notes in Computer Science*, pp. 329–351. Springer, 2024. doi: 10.1007/978-3-031-65630-9_17. URL https://doi.org/10.1007/978-3-031-65630-9_17.

R Iris Bahar, Erica A Frohm, Charles M Gaona, Gary D Hachtel, Enrico Macii, Abelardo Pardo, and Fabio Somenzi. Algebraic decision diagrams and their applications. *Formal methods in system design*, 10(2):171–206, 1997.

Sumon Biswas and Hridesh Rajan. Fairify: Fairness verification of neural networks. In *45th IEEE/ACM International Conference on Software Engineering, ICSE 2023, Melbourne, Australia, May 14-20, 2023*, pp. 1546–1558. IEEE, 2023. doi: 10.1109/ICSE48619.2023.00134. URL https://doi.org/10.1109/ICSE48619.2023.00134.

Hendrik Blockeel, Laurens Devos, Benoît Frénay, Géraldin Nanfack, and Siegfried Nijssen. Decision trees: from efficient prediction to responsible ai. *Frontiers in Artificial Intelligence*, 6, 2023. ISSN 2624-8212. doi: 10.3389/frai.2023.1124553. URL https://www.frontiersin.org/journals/artificial-intelligence/articles/10.3389/frai.2023.1124553.

Leo Breiman. Random forests. *Machine learning*, 45(1):5–32, 2001.

Randal E Bryant. Graph-based algorithms for boolean function manipulation. *IEEE Transactions on Computers*, 100(8):677–691, 1986.

Stefano Calzavara, Lorenzo Cazzaro, Claudio Lucchese, and Federico Marcuzzi. Explainable global fairness verification of tree-based classifiers. *2023 IEEE Conference on Secure and Trustworthy Machine Learning (SaTML)*, pp. 1–17, 2022. URL https://api.semanticscholar.org/CorpusID:252545325.

Yung-Chia Chang, Kuei-Hu Chang, and Guan-Jhih Wu. Application of extreme gradient boosting trees in the construction of credit risk assessment models for financial institutions. *Applied Soft Computing*, 73:914–920, 2018. ISSN 1568-4946. doi: https://doi.org/10.1016/j.asoc.2018.09.029. URL https://www.sciencedirect.com/science/article/pii/S1568494618305465.

Mark Chavira and Adnan Darwiche. Encoding cnfs to empower component analysis. In Armin Biere and Carla P. Gomes (eds.), *Theory and Applications of Satisfiability Testing - SAT 2006*, pp. 61–74, Berlin, Heidelberg, 2006. Springer Berlin Heidelberg. ISBN 978-3-540-37207-3.

Hongge Chen, Huan Zhang, Si Si, Yang Li, Duane S. Boning, and Cho-Jui Hsieh. Robustness verification of tree-based models. In Hanna M. Wallach, Hugo Larochelle, Alina Beygelzimer, Florence d'Alché-Buc, Emily B. Fox, and Roman Garnett (eds.), *Advances in Neural Information Processing Systems 32: Annual Conference on Neural Information Processing Systems 2019, NeurIPS 2019, December 8-14, 2019, Vancouver, BC, Canada*, pp. 12317–12328, 2019. URL https://proceedings.neurips.cc/paper/2019/hash/cd9508fdaa5c1390e9cc329001cf1459-Abstract.html.

Tianqi Chen and Carlos Guestrin. XGBoost: A scalable tree boosting system. In *Proceedings of the 22nd ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, KDD '16, pp. 785–794, New York, NY, USA, 2016. ACM. ISBN 978-1-4503-4232-2. doi: 10.1145/2939672.2939785. URL http://doi.acm.org/10.1145/2939672.2939785.

Alexis de Colnet, Stefan Szeider, and Tianwei Zhang. Compilation and fast model counting beyond CNF. In *Proceedings of the Thirty-Third International Joint Conference on Artificial Intelligence, IJCAI 2024, Jeju, South Korea, August 3-9, 2024*, pp. 3315–3323. ijcai.org, 2024. URL https://www.ijcai.org/proceedings/2024/367.

Laurens Devos, Wannes Meert, and Jesse Davis. Versatile verification of tree ensembles. In Marina Meila and Tong Zhang (eds.), *Proceedings of the 38th International Conference on Machine Learning*, volume 139 of *Proceedings of Machine Learning Research*, pp. 2654–2664. PMLR, 18–24 Jul 2021. URL https://proceedings.mlr.press/v139/devos21a.html.

Laurens Devos, Lorenzo Cascioli, and Jesse Davis. Robustness verification of multi-class tree ensembles. *Proceedings of the AAAI Conference on Artificial Intelligence*, 38(19):21019–21028, Mar. 2024. doi: 10.1609/aaai.v38i19.30093. URL https://ojs.aaai.org/index.php/AAAI/article/view/30093.

Mara Downing, William Eiers, Erin DeLong, Anushka Lodha, Brian Ozawa Burns, Ismet Burak Kadron, and Tevfik Bultan. Quantitative symbolic robustness verification for quantized neural networks. In Kazuhiro Ogata, Dominique Méry, Meng Sun, and Shaoying Liu (eds.), *Formal Methods and Software Engineering - 25th International Conference on Formal Engineering Methods, ICFEM 2024, Hiroshima, Japan, December 2-6, 2024, Proceedings*, volume 15394 of *Lecture Notes in Computer Science*, pp. 125–145. Springer, 2024. doi: 10.1007/978-981-96-0617-7_8. URL https://doi.org/10.1007/978-981-96-0617-7_8.

Jeffrey M. Dudek, Vu Phan, and Moshe Y. Vardi. ADDMC: weighted model counting with algebraic decision diagrams. In *The Thirty-Fourth AAAI Conference on Artificial Intelligence, AAAI 2020, The Thirty-Second Innovative Applications of Artificial Intelligence Conference, IAAI 2020, The Tenth AAAI Symposium on Educational Advances in Artificial Intelligence, EAAI 2020, New York, NY, USA, February 7-12, 2020*, pp. 1468–1476. AAAI Press, 2020. doi: 10.1609/AAAI.V34I02.5505. URL https://doi.org/10.1609/aaai.v34i02.5505.

Cynthia Dwork, Moritz Hardt, Toniann Pitassi, Omer Reingold, and Richard Zemel. Fairness through awareness. In *Proceedings of the 3rd innovations in theoretical computer science conference*, pp. 214–226, 2012.

Sainyam Galhotra, Yuriy Brun, and Alexandra Meliou. Fairness testing: testing software for discrimination. In *Proceedings of the 2017 11th Joint Meeting on Foundations of Software Engineering*, ESEC/FSE 2017, pp. 498–510, New York, NY, USA, 2017. Association for Computing Machinery. ISBN 9781450351058. doi: 10.1145/3106237.3106277. URL https://doi.org/10.1145/3106237.3106277.

Mohammad M. Ghiasi and Sohrab Zendehboudi. Application of decision tree-based ensemble learning in the classification of breast cancer. *Computers in Biology and Medicine*, 128:104089, 2021. ISSN 0010-4825. doi: https://doi.org/10.1016/j.compbiomed.2020.104089. URL https://www.sciencedirect.com/science/article/pii/S0010482520304200.

Bishwamittra Ghosh, Debabrota Basu, and Kuldeep S. Meel. Algorithmic fairness verification with graphical models. In *Thirty-Sixth AAAI Conference on Artificial Intelligence, AAAI 2022, Thirty-Fourth Conference on Innovative Applications of Artificial Intelligence, IAAI 2022, The Twelveth Symposium on Educational Advances in Artificial Intelligence, EAAI 2022 Virtual Event, February 22 - March 1, 2022*, pp. 9539–9548. AAAI Press, 2022. doi: 10.1609/AAAI.V36I9.21187. URL https://doi.org/10.1609/aaai.v36i9.21187.

Ashutosh Gupta, Thomas A. Henzinger, Konstantin Kueffner, Kaushik Mallik, and David Pape. Monitoring robustness and individual fairness. In Luiza Antonie, Jian Pei, Xiaohui Yu, Flavio Chierichetti, Hady W. Lauw, Yizhou Sun, and Srinivasan Parthasarathy (eds.), *Proceedings of the 31st ACM SIGKDD Conference on Knowledge Discovery and Data Mining, V.2, KDD 2025, Toronto ON, Canada, August 3-7, 2025*, pp. 790–801. ACM, 2025. doi: 10.1145/3711896.3737054. URL https://doi.org/10.1145/3711896.3737054.

Alexey Ignatiev, Yacine Izza, Peter J. Stuckey, and João Marques-Silva. Using maxsat for efficient explanations of tree ensembles. In *Thirty-Sixth AAAI Conference on Artificial Intelligence, AAAI 2022, Thirty-Fourth Conference on Innovative Applications of Artificial Intelligence, IAAI 2022, The Twelveth Symposium on Educational Advances in Artificial Intelligence, EAAI 2022 Virtual Event, February 22 - March 1, 2022*, pp. 3776–3785. AAAI Press, 2022. doi: 10.1609/AAAI.V36I4.20292. URL https://doi.org/10.1609/aaai.v36i4.20292.

Yacine Izza, Alexey Ignatiev, Sasha Rubin, João Marques-Silva, and Peter J. Stuckey. Most general explanations of tree ensembles. In *Proceedings of the Thirty-Fourth International Joint Conference on Artificial Intelligence, IJCAI 2025, Montreal, Canada, August 16-22, 2025*, pp. 5463–5471. ijcai.org, 2025. doi: 10.24963/IJCAI.2025/608. URL https://doi.org/10.24963/ijcai.2025/608.

Alex Kantchelian, J. D. Tygar, and Anthony D. Joseph. Evasion and hardening of tree ensemble classifiers. In Maria-Florina Balcan and Kilian Q. Weinberger (eds.), *Proceedings of the 33nd International Conference on Machine Learning, ICML 2016, New York City, NY, USA, June 19-24, 2016*, volume 48 of *JMLR Workshop and Conference Proceedings*, pp. 2387–2396. JMLR.org, 2016. URL http://proceedings.mlr.press/v48/kantchelian16.html.

Guolin Ke, Qi Meng, Thomas Finley, Taifeng Wang, Wei Chen, Weidong Ma, Qiwei Ye, and Tie-Yan Liu. Lightgbm: A highly efficient gradient boosting decision tree. *Advances in neural information processing systems*, 30:3146–3154, 2017.

Brian Hyeongseok Kim, Jingbo Wang, and Chao Wang. Fairquant: Certifying and quantifying fairness of deep neural networks. In *47th IEEE/ACM International Conference on Software Engineering, ICSE 2025, Ottawa, ON, Canada, April 26 - May 6, 2025*, pp. 527–539. IEEE, 2025. doi: 10.1109/ICSE55347.2025.00016. URL https://doi.org/10.1109/ICSE55347.2025.00016.

Pang Wei Koh, Shiori Sagawa, Henrik Marklund, Sang Michael Xie, Marvin Zhang, Akshay Balsubramani, Weihua Hu, Michihiro Yasunaga, Richard Lanas Phillips, Irena Gao, Tony Lee, Etienne David, Ian Stavness, Wei Guo, Berton Earnshaw, Imran Haque, Sara M Beery, Jure Leskovec, Anshul Kundaje, Emma Pierson, Sergey Levine, Chelsea Finn, and Percy Liang. Wilds: A benchmark of in-the-wild distribution shifts. In Marina Meila and Tong Zhang (eds.), *Proceedings of the 38th International Conference on Machine Learning*, volume 139 of *Proceedings of Machine Learning Research*, pp. 5637–5664. PMLR, 18–24 Jul 2021.

Daphne Koller and Nir Friedman. *Probabilistic Graphical Models: Principles and Techniques*. MIT Press, 2009.

Jean-Marie Lagniez, Pierre Marquis, and Armin Biere. Dynamic blocked clause elimination for projected model counting. In Supratik Chakraborty and Jie-Hong Roland Jiang (eds.), *27th International Conference on Theory and Applications of Satisfiability Testing, SAT 2024, Pune, India, August 21-24, 2024*, volume 305 of *LIPIcs*, pp. 21:1–21:17. Schloss Dagstuhl - Leibniz-Zentrum für Informatik, 2024. doi: 10.4230/LIPICS.SAT.2024.21. URL https://doi.org/10.4230/LIPIcs.SAT.2024.21.

Mehul Madaan, Aniket Kumar, Chirag Keshri, Rachna Jain, and Preeti Nagrath. Loan default prediction using decision trees and random forest: A comparative study. *IOP Conference Series: Materials Science and Engineering*, 1022(1):012042, jan 2021. doi: 10.1088/1757-899X/1022/1/012042. URL https://dx.doi.org/10.1088/1757-899X/1022/1/012042.

Kuldeep S. Meel, N. V. Vinodchandran, and Sourav Chakraborty. Estimating the size of union of sets in streaming models. In Leonid Libkin, Reinhard Pichler, and Paolo Guagliardo (eds.), *PODS'21: Proceedings of the 40th ACM SIGMOD-SIGACT-SIGAI Symposium on Principles of Database Systems, Virtual Event, China, June 20-25, 2021*, pp. 126–137. ACM, 2021. doi: 10.1145/3452021.3458333. URL https://doi.org/10.1145/3452021.3458333.

Wannes Meert and Arthur Choi. Pysdd. In Adnan Darwiche, Pierre Marquis, Dan Suciu, and Stefan Szeider (eds.), *Recent Trends in Knowledge Compilation, Report from Dagstuhl Seminar 17381*, September 2017.

Jose G. Moreno-Torres, Troy Raeder, Rocío Alaiz-Rodríguez, Nitesh V. Chawla, and Francisco Herrera. A unifying view on dataset shift in classification. *Pattern Recognition*, 45(1):521–530, 2012. doi: 10.1016/j.patcog.2011.06.019.

Ajinkya Naik, Chaitanya Garg, S. Akshay, Ashutosh Gupta, and Kuldeep S. Meel. Quantifying sensitivity for tree ensembles: A symbolic and compositional approach. In Eva Darulova, Anthony W. Lin, and Philipp Rümmer (eds.), *Computer Aided Verification*, pp. 551–575, Cham, 2026. Springer Nature Switzerland. ISBN 978-3-032-32526-6.

Fabian Pedregosa, Gaël Varoquaux, Alexandre Gramfort, Vincent Michel, Bertrand Thirion, Olivier Grisel, Mathieu Blondel, Peter Prettenhofer, Ron Weiss, Vincent Dubourg, et al. Scikit-learn: Machine learning in python. *Journal of machine learning research*, 12(Oct):2825–2830, 2011.

Joaquin Quiñonero-Candela, Masashi Sugiyama, Anton Schwaighofer, and Neil D. Lawrence (eds.). *Dataset Shift in Machine Learning*. MIT Press, Cambridge, MA, 2009. doi: 10.7551/mitpress/9780262170055.001.0001.

Francesco Ranzato and Marco Zanella. Abstract interpretation of decision tree ensemble classifiers. *Proceedings of the AAAI Conference on Artificial Intelligence*, 34(04):5478–5486, Apr. 2020. doi: 10.1609/aaai.v34i04.5998. URL https://ojs.aaai.org/index.php/AAAI/article/view/5998.

Shubham Sharma, Subhajit Roy, Mate Soos, and Kuldeep S. Meel. Ganak: A scalable probabilistic exact model counter. In *Proceedings of International Joint Conference on Artificial Intelligence (IJCAI)*, 2019.

Hidetoshi Shimodaira. Improving predictive inference under covariate shift by weighting the log-likelihood function. *Journal of Statistical Planning and Inference*, 90(2):227–244, 2000. doi: 10.1016/S0378-3758(00)00115-4.

Mate Soos and Kuldeep S. Meel. Engineering an efficient probabilistic exact model counter. In *Proceedings of the International Conference on Computer Aided Verification (CAV)*, 2025.

Mate Soos, Stephan Gocht, and Kuldeep S. Meel. Tinted, detached, and lazy CNF-XOR solving and its applications to counting and sampling. In Shuvendu K. Lahiri and Chao Wang (eds.), *Computer Aided Verification - 32nd International Conference, CAV 2020, Los Angeles, CA, USA, July 21-24, 2020, Proceedings, Part I*, volume 12224 of *Lecture Notes in Computer Science*, pp. 463–484. Springer, 2020. doi: 10.1007/978-3-030-53288-8_22. URL https://doi.org/10.1007/978-3-030-53288-8_22.

Mate Soos, Uddalok Sarkar, Divesh Aggarwal, Sourav Chakraborty, Kuldeep S. Meel, and Maciej Obremski. Engineering an efficient approximate DNF-counter. *arXiv preprint arXiv:2407.19946*, 2024.

Namrita Varshney, Ashutosh Gupta, Arhaan Ahmad, Tanay Vineet Tayal, and S. Akshay. Data-aware and scalable sensitivity analysis for decision tree ensembles. In *The Fourteenth International Conference on Learning Representations, ICLR 2026, Rio de Janeiro, April 23-27, 2026*. OpenReview.net, 2026. URL https://openreview.net/pdf?id=q8KqAvdfZK.

Yihan Wang, Huan Zhang, Hongge Chen, Duane S. Boning, and Cho-Jui Hsieh. On lp-norm robustness of ensemble decision stumps and trees. In *Proceedings of the 37th International Conference on Machine Learning, ICML 2020, 13-18 July 2020, Virtual Event*, volume 119 of *Proceedings of Machine Learning Research*, pp. 10104–10114. PMLR, 2020. URL http://proceedings.mlr.press/v119/wang20aa.html.

Jiong Yang and Kuldeep S. Meel. Rounding meets approximate model counting. In Constantin Enea and Akash Lal (eds.), *Computer Aided Verification - 35th International Conference, CAV 2023, Paris, France, July 17-22, 2023, Proceedings, Part II*, volume 13965 of *Lecture Notes in Computer Science*, pp. 132–162. Springer, 2023. doi: 10.1007/978-3-031-37703-7_7. URL https://doi.org/10.1007/978-3-031-37703-7_7.

A Appendix

B Details of Section 4

This section contains the details of the procedures used by the primary algorithms listed in Section 4.

Implementation of ProcessSubproblem**.** Each subproblem is processed by the function ProcessSubproblem. The function PruneTree uses the bit-mask input to prune the branches of the trees of the ensemble which are rendered unreachable due to fixing the nodes according to the values of bit-mask. Procedure EncodeADD encodes each tree as an ADD, by coverting each tree node into an ADD node and each leaf value to terminal node (leaf) of the ADD . We take a pair-wise difference of each tree in the ensemble to get DiffADDs that encodes the difference and add all such ADDs to get DiffSum ADD, which encodes the sum of all differences, essentially encoding the difference between the output of the two ensembles. This holds true as the difference between two sums over the ADDs is equal to the sum of the pair-wise differences between the corressponding ADDs of the two ensembles. The procedure ToBDD (Algorithm 6) converts this ADD to a BDD by keeping the internal nodes same and values of the leaves into boolean values based on whether they exceed G. This resultant BDD is the object over which the approximate satisfying assignments are computed. Let U_i denote the set of satisfying assignments of the BDD corresponding to the ith subproblem.

Algorithm 2 ProcessSubproblem($\mathcal{T}, p, G$)

$\mathcal{T}' \leftarrow \emptyset$,$\mathcal{T}'' \leftarrow \emptyset$
for $i = 1$ to $|\mathcal{T}|$ **do**
    $\mathcal{T}'$.Append(PruneTree($T_i.root, p.mask_1$))
    $\mathcal{T}''$.Append(PruneTree($T_i.root, p.mask_2$))
DiffSum $\leftarrow 0$
for $i = 1$ to $|\mathcal{T}|$ **do**
    DiffADD $\leftarrow$ EncodeADD($T_i^{'}.root$) $-$ EncodeADD($T_i^{''}.root$)
    DiffSum $\leftarrow$ DiffSum $+$ DiffADD
$\varphi \leftarrow$ ToBDD(DiffSum, G)
return φ

The subroutine BitMaskGen takes as input an integer k and gives $(k+1)$ bit vectors, each of size k, such that for each bit vector the monotonicity property is followed - bit preceding a given bit must be greater that or equal to it.

Algorithm 3 BitMaskGen(k)

$MaskSet \leftarrow \emptyset$
$BitMask[k] \leftarrow 1$
for $i = 1$ to k **do**
    $BitMask[i] \leftarrow 0$
    $MaskSet \leftarrow BitMask$
return $MaskSet$

The subroutine PruneTree takes a node of the tree as input and recursively prunes it by traversing the tree and checking decision at each node against the given assignment set m.

Algorithm 4 PruneTree(n, m)

if n is leaf **then return**
$n.yes \leftarrow$ PruneTree($n.yes, m$)
$n.no \leftarrow$ PruneTree($n.no, m$)
if $(n.feature, n.threshold) \notin m$ **then return** n
$i \leftarrow$ index of $(n.feature, n.threshold)$ in m
nodes $keep \leftarrow nullptr$, $discard \leftarrow nullptr$
if $m[i] == 1$ **then**
    $keep \leftarrow n.yes$
    $discard \leftarrow n.no$
else
    $keep \leftarrow n.no$
    $discard \leftarrow n.yes$
return $keep$

The subroutine EncodeADD takes a decision tree node and recursively encodes it as an ADD by traversing through the tree by making the tree nodes as the nodes of the ADD, and the leaf value as the terminal node of the ADD which stores the real value.

Algorithm 5 EncodeADD(n)

init ADD node A
if n is a leaf node **then return** $A \leftarrow n.val$
else
    $A.true \leftarrow$ EncodeADD($n.yes$)
    $A.false \leftarrow$ EncodeADD($n.no$)
return A

The subroutine ToBDD converts an ADD to a BDD by taking a threshold value and constructing the terminal nodes of the BDD according to the decision whether the real value in the terminal node of the ADD is greater than the threshold.

Algorithm 6 ToBDD(n)

if n is a leaf node **then**
    if $n.val > G$ **then return** 1
    elsereturn 0
else
    ToBDD($n.yes$)
    ToBDD($n.no$)
return

The subroutine SampleBDD takes a BDD, an integer N and a pair of bit-masks, and samples satisfying assignments from the BDD uniformly, i.e, proportional to its weight with respect to the weight

of all satisfying assignments, and appends it with either one of the masks (randomly selected) to form the complete assignment and returns all such complete assignments.

Algorithm 7 SampleBDD(BDD,N,$pair$)

$S \leftarrow \emptyset$
for $i = 1$ to N **do**
    Sample assignment x proportional to its weight wrt all satisfying assignments
    $m \leftarrow$ Sample one from bitmask pair $(pair.m_1, pair.m_2)$ with respect to weight of the mask.
    S.Append$((x, m))$
return S

Implementation of BNmass**.** The subroutine BNmass computes the probability mass, under the Bayesian network BN, of the satisfying assignments of the BDD φ of a subproblem $(mask_1, mask_2)$. Since the sensitive variables are fixed by the masks, the mass is the sum of two weighted counts over the non-sensitive variables $b_1, \ldots, b_{M'}$, one per mask. Each is computed by bucket (variable) elimination over ADDs: the BDD φ is lifted to a $0/1$ ADD $[\varphi]$, each conditional probability table of BN is encoded as an ADD whose sensitive variables are fixed to their values in the mask, and the non-sensitive variables are summed out one at a time in reverse variable order. The intermediate products (buckets) β_j are retained and reused by SampleBDD to draw assignments proportional to their weight.

Algorithm 8 BNmass($\varphi, pair, BN$)

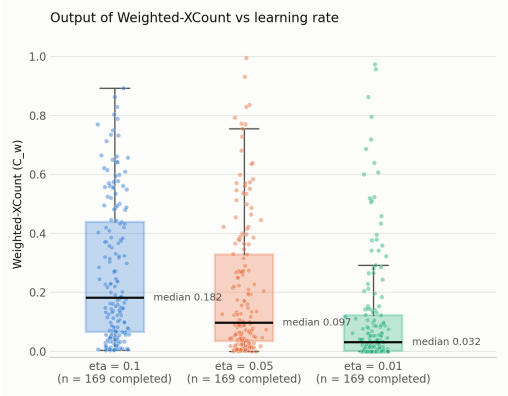
$W_1 \leftarrow$ MaskMass$(\varphi, pair.mask_1, BN)$
$W_2 \leftarrow$ MaskMass$(\varphi, pair.mask_2, BN)$
return $W_1 + W_2$

procedure MASKMASS(φ, m, BN)
    $\mathcal{F} \leftarrow \{[\varphi]\}$ ▷ $0/1$ ADD of the BDD φ
    for each node v of BN with parents $\mathrm{Pa}(v)$ **do**
        $\psi_v \leftarrow \sum_c \mathbb{1}[b_{\mathrm{Pa}(v)} = c] \cdot \big(b_v \cdot \theta_{v|c}(1) + (1 - b_v) \cdot \theta_{v|c}(0)\big)$
        Fix every sensitive variable in $\{v\} \cup \mathrm{Pa}(v)$ in ψ_v to its value in m
        $\mathcal{F} \leftarrow \mathcal{F} \cup \{\psi_v\}$
    $w \leftarrow 1$
    for $j = M'$ **down to** 1 **do** ▷ non-sensitive variables, reverse order
        $\mathcal{B}_j \leftarrow \{\psi \in \mathcal{F} \mid b_j \in \mathrm{supp}(\psi)\}$
        if $\mathcal{B}_j = \emptyset$ **then**
            $w \leftarrow 2 \cdot w$ ▷ b_j is unconstrained
            continue
        $\beta_j \leftarrow \prod_{\psi \in \mathcal{B}_j} \psi$ ▷ bucket, retained for SampleBDD
        $\mathcal{F} \leftarrow (\mathcal{F} \setminus \mathcal{B}_j) \cup \big\{\beta_j|_{b_j=0} + \beta_j|_{b_j=1}\big\}$ ▷ sum out b_j
    for each $\psi \in \mathcal{F}$ **do** ▷ all remaining factors are constants
        $w \leftarrow w \cdot \psi$
    return w

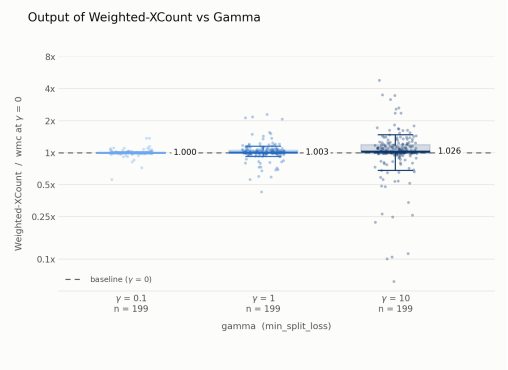
C HYPERPARAMETERS AND WEIGHTED-XCount

To probe the relation between the hyperparameters and the weighted count of sensitive regions, we consider a set of experiments where we observe the effect of varying each hyperparameter on the output of Weighted-XCount. We conduct a set of experiments by selecting *California housing* dataset, training models with varying hyperparameters and compare the output of Weighted-XCount with the change in hyperparameters. We specifically focus on parameters that are associated with overfitting, to test the hypothesis whether our measure of sensitivity i.e the output of Weighted-XCount is linked to overfitting. The experimental evaluation shows that the sensitivity measure

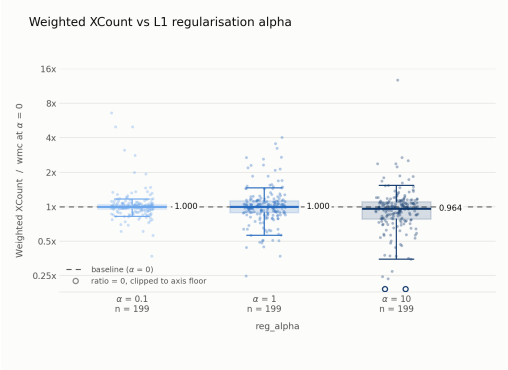
given Weighted-XCount is affected only in the case of varying of the learning, with the variation of other hyperparameters show no significant trend.



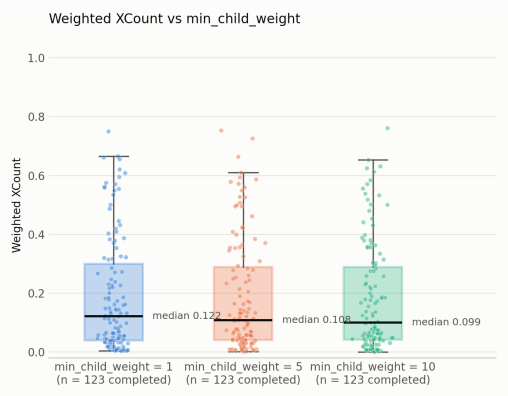
(a) Learning rate



(b) Gamma



(c) Alpha



(d) Minimum child weight

Figure 4: Weighted-XCount as a function of each hyperparameter.