An Efficient Adversarial Attack for Tree Ensembles

Chong Zhang  Huan Zhang  Cho-Jui Hsieh
Department of Computer Science, UCLA
chongz@cs.ucla.edu, huan@huan-zhang.com, chohsieh@cs.ucla.edu

Abstract

We study the problem of efficient adversarial attacks on tree based ensembles such as gradient boosting decision trees (GBDTs) and random forests (RFs). Since these models are non-continuous step functions and gradient does not exist, most existing efficient adversarial attacks are not applicable. Although decision-based black-box attacks can be applied, they cannot utilize the special structure of trees. In our work, we transform the attack problem into a discrete search problem specially designed for tree ensembles, where the goal is to find a valid “leaf tuple” that leads to mis-classification while having the shortest distance to the original input. With this formulation, we show that a simple yet effective greedy algorithm can be applied to iteratively optimize the adversarial example by moving the leaf tuple to its neighborhood within hamming distance 1. Experimental results on several large GBDT and RF models with up to hundreds of trees demonstrate that our method can be thousands of times faster than the previous mixed-integer linear programming (MILP) based approach, while also providing smaller (better) adversarial examples than decision-based black-box attacks on general $\ell_p$ ($p = 1, 2, \infty$) norm perturbations. Our code is available at https://github.com/chong-z/tree-ensemble-attack.

1 Introduction

It has been widely studied that machine learning models are vulnerable to adversarial examples (Szegedy et al., 2013; Goodfellow et al., 2015; Athalye et al., 2018), where a small imperceptible perturbation on the input can easily alter the prediction of a model. A series of adversarial attack methods have been proposed on continuous models such as neural networks, which can be generally split into two types. The gradient based methods formulate the attack into an optimization problem on a specially designed loss function for attacks, where the gradient can be acquired through either back-propagation in the white-box setting (Carlini, Wagner, 2017; Madry et al., 2018), or numerical estimation in the soft-label black-box setting (Chen et al., 2017; Tu et al., 2018; Ilyas et al., 2018). The decision based (or hard-label black-box) methods only have access to the output label, which usually starts with an initial adversarial example and minimizes the perturbation along the decision boundary (Brendel et al., 2018; Brunner et al., 2018; Cheng et al., 2019, 2020; Chen et al., 2019c).

In this paper we study the problem of efficient adversarial attack on tree based ensembles such as gradient boosting decision trees (GBDT) and random forests (RFs), which have been widely used in practice (Chen, Guestrin, 2016; Ke et al., 2017; Zhang et al., 2017; Prokhorenkova et al., 2018). We minimize the perturbation to find the smallest possible attack, to uncover the true weakness of a model. Different from neural networks, tree based ensembles are non-continuous step functions and existing gradient based methods are not applicable. Decision based methods can be applied but they usually require a large number of queries and may easily fall into local optimum due to rugged decision boundary. In general, finding the exact minimal adversarial perturbation for tree ensembles is NP-complete (Kantchelian et al., 2015), and a feasible approximation solution is necessary to evaluate the robustness of large ensembles.
The major difficulty of attacking tree ensembles is that the prediction remains unchanged within regions on the input space, where the region could be large and makes continuous updates inefficient. To overcome this difficulty, we transform the continuous $\mathbb{R}^d$ input space into a discrete $\{1, 2, \ldots, N\}^K$ “leaf tuple” space, where $N$ is the number of leaves per tree and $K$ is the number of trees. On the leaf tuple space we define the distance between two input examples to be the number of trees that have different prediction leaves (i.e., hamming distance), and define the neighborhood of a tuple to be all valid tuples within a small hamming distance. In practice, we propose the attack that iteratively optimizes the adversarial leaf tuple by moving it to the best adversarial tuple within the neighborhood of distance 1. Intuitively we could reach a far away adversarial tuple through a series of smaller updates, based on the fact that each tree makes prediction independently.

In experiments, we compare $\ell_{1, 2, \infty}$ norm perturbation metrics across 10 datasets, and show that our method is thousands of times faster than MILP (Kantchelian et al. 2015) on most of the large ensembles, and 3~72x faster than decision based and empirical attacks on all datasets while achieving a smaller distortion. For instance, with the standard (natural) GBDT on the MNIST dataset with 10 classes and 200 trees per class, our method finds the adversarial example with only 2.07 times larger $\ell_\infty$ perturbation than the optimal solution produced by MILP and only uses 0.237 seconds per test example, whereas MILP requires 375 seconds. As for other approximate attacks, SignOPT (Cheng et al. 2020) finds a 13.93 times larger $\ell_\infty$ perturbation (compared to MILP) using 3.7 seconds, HSJA (Chen et al. 2019c) achieves a 8.36 times larger $\ell_\infty$ perturbation using 1.8 seconds, and Cube (Andriushchenko, Hein 2019) achieves a 4 times larger $\ell_\infty$ perturbation using 4.42 seconds. Additionally, although $\ell_p$ distance is widely used in previous attacks and a small $\ell_p$ perturbation is usually invisible, our method is general and can also be adapted to other distance metrics.

2 Background and Related Work

Problem Setting While the main idea can be applied to multi-class classification models and targeted attacks, for simplicity we consider a binary classification model $f : \mathbb{R}^d \rightarrow \{-1, 1\}$ consisting of $K$ decision trees. Each tree $t$ is a weak learner $f_t : \mathbb{R}^d \rightarrow \mathbb{R}$ of $N$ leaves, and the ensemble returns the sign $f(x) = \text{sign}(\sum_{t=1}^{K} f_t(x))$. Given a victim input example $x_0$ with $y_0 = f(x_0)$, we want to find the minimal adversarial perturbation $\delta^*_p$, determining the adversarial robustness under $\ell_p$ norm:

$$\delta^*_p = \min_{\delta} \|\delta\|_p \quad \text{s.t.} \quad f(x_0 + \delta) \neq y_0. \quad (1)$$

Exact Solutions In general computing the exact (optimal) solution for Eq. (1) requires exponential time. (Kantchelian et al. 2015) showed that the problem is NP-complete for general ensembles and proposed a MILP based method; On the other hand, faster algorithms exist for models of special form: Zhang et al. (2020) restricted both the input and prediction of every tree $t$ to binary values $f_t : \{-1, 1\}^d \rightarrow \{-1, 1\}$ and provided an integer linear program (ILP) based formulation about 4 times faster than (Kantchelian et al. 2015). Andriushchenko, Hein (2019) showed that the exact robustness of boosted decision stumps (i.e., depth = 1) can be solved in polynomial time; Chen et al. (2019b) proposed a polynomial time algorithm to solve a single decision tree.

Approximate Solutions To get a feasible solution for general models a series of methods have been proposed to compute the lower bound (robustness verification) and the upper bound (adversarial attacks) of $\delta^*_p$. Chen et al. (2019b) formulated the robustness verification problem into a max–clique problem on a multi-partite graph and produced a lower bound on $\ell_\infty$. Wang et al. (2020) extended the verification method and produced a lower bound on general $\ell_p$ norms; Lee et al. (2019) verified the $\ell_0$ robustness on the randomly smoothed ensembles which is not directly related to our work. On the other hand, decision based attacks (Brendel et al. 2018; Brunner et al. 2018; Cheng et al. 2019, 2020; Chen et al. 2019c) can be applied here to produce an upper bound since they don’t have architecture or smoothness assumptions on $f(\cdot)$, however they are usually ineffective due to the discrete nature of tree models; Andriushchenko, Hein (2019) proposed the Cube attack for tree ensembles that does stochastic updates along the $\ell_\infty$ boundary, which typically achieves better results than decision based attacks; Yang et al. (2019) focused on the theoretical analysis of search space decomposition, and proposed RBA-Appr to search over a subset of the $N^K$ convex polyhedrons containing training examples; Zhang et al. (2020) restricted both the input and prediction of all trees to binary values and provided a heuristic attack on $\ell_0$ by assigning empirical weights to each feature, in both the white-box setting and a special “black-box” setting via training substitute models. In contrast,
We propose an iterative approach where the algorithm starts with an initial adversarial example $x$ which is the region that will fall into this node following the feature split thresholds along the traverse starts from the root node of each tree. Each internal node of index $i$ has two children and a feature split threshold $f_i$ respectively.

In most of the existing attacks, Eq. (2) is solved by a continuous optimization algorithm where the measurements are not suitable here. And if we define the neighborhood as a large $\eta$ ball, due to the non-continuity of trees, $f(x)$ becomes intractable to enumerate. To handle these difficulties we introduce the concept of leaf tuple and rewrite Eq. (2) into a discrete form. Given an input example $x = [x_1, \ldots, x_d]$ the traverse starts from the root node of each tree. Each internal node of index $i$ has two children and a feature split threshold $(j^i, \eta^i)$, and $x$ will be passed to the left child if $x_{j^i} \leq \eta^i$ and to the right child otherwise. The leaf node has a prediction label $v^i$, and will output $v^i$ when $x$ reaches here. We use $C(x) = (i(1)(x), \ldots, i(K)(x))$ to denote the index tuple of $K$ prediction leaves from input $x$. In general we use the subscript $j$ to denote the $j_{th}$ dimension, and the superscripts $i^t$ or $j^t$ to denote the $i_{th}$ node and $i_{th}$ tree respectively.

**Definition 1. (Bounding Box)** $B^i = (l_i^1, r_i^1) \times \cdots \times (l_i^d, r_i^d)$ denotes the bounding box of node $i$, which is the region that $x$ will fall into this node following the feature split thresholds along the traverse path, and each $l$ and $r$ is either $\pm \infty$ or equals to one $\eta^h$, along the traverse path. We use $B(C(x)) = \bigcap_{i \in C(x)} B^i = \bigcap_{i \in C(x)} (l_i^1, r_i^1) \times \cdots \times \bigcap_{i \in C(x)} (l_i^d, r_i^d)$ to denote the Cartesian product of the intersection of $K$ bounding boxes on the ensemble.

**Definition 2. (Valid Tuple)** $C = \{C(x) \mid \forall x \in \mathbb{R}^d\}$ denotes the set of all possible tuples that correspond to at least one point in the input space, and $C = (i(1), \ldots, i(K))$ is a valid tuple iff. $C \in \mathbb{C}$.
Figure 1: An ensemble defined on $[0, 25]^2$ and its corresponding decision boundaries on the input space. Numbers inside circles are indexes of leaves, and the number below each leaf is the corresponding prediction label $\psi^i$. For clarity we mark boundaries belong to tree 1, 2, 3 with red, green, and blue respectively, and fill +1 area with gray. $x_0$ is the victim example and $x'$ is an initial adversarial example. Assume we are optimizing $\ell_1$ perturbation, our method can reach $d$ by changing one leaf of the tuple at a time (black arrows). On the other hand, decision based attacks update the solution along the decision boundary, and easily fall into local minimums such as $x'$ and $a$ (brown arrows) since they look only at the continuous neighborhood. To move from $a$ to $b$, the path on decision boundary is $a \rightarrow (5, 10) \rightarrow b$, but since $a \rightarrow (5, 10)$ increases the distortion they won’t find this path.

**Theorem 1.** ([Chen et al., 2019b]) The intersection $B(\mathcal{C})$ can also be written as the Cartesian product of $d$ intervals (similar to $B^d$) and $\mathcal{C} \in \mathbb{C} \iff B(\mathcal{C}) \neq \emptyset \iff \forall i, j \in \mathcal{C}, B^i \cap B^j \neq \emptyset$. (These concepts were used in [Chen et al., 2019b] for verification instead of attack.)

**Corollary 1.** (Tuple to Example Distance) The shortest distance between a valid leaf tuple and an example, defined as $dist_p(\mathcal{C}, x_0) = \min_{x \in B(\mathcal{C})} \| x - x_0 \|_p$, can be solved in $O(d)$ time.

Observe $\mathcal{C}(x) = \mathcal{C}(x')$, $\forall x \in B(\mathcal{C}(x'))$ and we can transform the intractable number of $x$ into tractable number of leaf tuples $\mathcal{C}$. We abuse the notation $f(\mathcal{C})$ to denote the model prediction $\text{sign} \left( \sum_{i \in \mathcal{C}} \psi^i \right)$, which is a constant within $B(\mathcal{C})$. Combined with Corollary 1 we can rewrite Eq. (2) into the discrete form below. $\text{Neighbor}(\mathcal{C}')$ denotes the neighborhood space around $\mathcal{C}'$, which is a set of leaf tuples that close to $\mathcal{C}'$ in certain distance measurements. Fig. 1 presents an example to demonstrate that it’s less likely to fall into local optimum on our newly defined neighborhood space.

$$\mathcal{C}'_{\text{new}} = \arg\min_{\mathcal{C}} dist_p(\mathcal{C}, x_0) \quad \text{s.t.} \quad \mathcal{C} \in \text{Neighbor}(\mathcal{C}') \cap \mathbb{C}, \; f(\mathcal{C}) \neq y_0. \quad (3)$$

### 3.2 Limitations of Naive Neighborhood Space Definitions

Now we discuss how to define Neighbor$(\mathcal{C}')$ to facilitate our attack. Intuitively the space should be efficient to compute, and has a reasonable coverage to avoid falling into local minimums too easily. In this section we discuss two naive approaches that fail on these two properties, and provide empirical results in Table 2.

**Enumerating all leaves is not efficient (NaiveLeaf):** Given current adversarial example $x'$ and its corresponding leaf tuple $\mathcal{C}' = (i^{(1)}, \ldots, i^{(K)})$, an intuitive approach is to change a single $i^{(t)}$ to a different leaf $i_{\text{new}}^{(t)}$. However the resulting tuple may not be valid, and we will have to query the ensemble to get a valid tuple in $\mathcal{C}$. We provide a possible implementation in Appendix D.2 where we move $x'$ to the closest point within $B^{i_{\text{new}}^{(t)}}$. NaiveLeaf requires multiple full model queries and takes $O(K \cdot 2^t \cdot K_l)$ time per iteration for $K$ trees of depth $l$, which is too time consuming (see Table 2).

**Mutating one feature at a time has poor coverage (NaiveFeature):** Given current adversarial example $x'$ and its corresponding leaf tuple $\mathcal{C}'$, another intuitive approach is to move $x'$ outside of $B(\mathcal{C}')$ on each feature dimension. This approach is efficient since there are at most $2d$ neighborhood, and each neighborhood is only different by one tree (assuming unique split thresholds). However the
We propose Leaf Tuple attack (LT-Attack) in Algorithm 1 that efficiently solves Eq. (3) through two
with the intersection of other

\[ (*) \]: For a fair comparison we disabled the random noise optimization discussed in §3.5. Our
LT-Attack searches in a subspace of NaiveLeaf so \( r_{\text{out}} \) is slightly larger, but it is significantly faster.

Table 2: Average \( \ell_2 \) perturbation over 500 test examples on the standard (natural) GBDT models.
(\("^*\)\): For a fair comparison we disabled the random noise optimization discussed in §3.5. Our
LT-Attack searches in a subspace of NaiveLeaf so \( r_{\text{out}} \) is slightly larger, but it is significantly faster.

| Standard GBDT | NaiveLeaf | NaiveFeature | LT-Attack (Ours)* | Ours vs. NaiveLeaf |
|---------------|-----------|--------------|-------------------|--------------------|
| \( r_{\text{leaf}} \) | \( t \) | \( \bar{r} \) | \( r_{\text{out}} \) | \( \bar{r}_{\text{out}} / r_{\text{out}} \) | Speedup |
| MNIST         | .081     | 2.375       | .229             | .069s              | .108               | .105s | 1.33 | 22.6X |
| F-MNIST       | .080     | 3.935       | .181             | .061s              | .096               | .224s | 1.20 | 17.5X |
| HIGGS         | .008     | 3.175       | .011             | .023s              | .009               | .051s | 1.13 | 102.3X |

The intuition is that each tree can be queried independently, and we want to utilize such property by
limiting the number of affected trees at each iteration. \( \text{Neighbor}_h(\cdot) \) has a nice property where we
can increase \( h \) for larger search scope, or decrease \( h \) to improve speed. Observe that \( \text{Neighbor}_1(C') \)
is a subset of NaiveLeaf (minus invalid leaf tuples that requires an expensive model query), and a
superset of NaiveFeature (plus leaf tuples that may affect multiple features). In experiments we are
able to achieve good results with \( \text{Neighbor}_1(\cdot) \), and we provide an empirical greedy algorithm in
Appendix D.3 to estimate the minimal \( h \) required to reach the exact solution.

3.3 Define Neighborhood Space by Hamming Distance

In this section we introduce a neighborhood space through discrete hamming distance, and show that
it’s fast to compute and has good coverage. We define the distance \( D(C, C') \) between two tuples as
the number of different leaves, and the neighborhood of \( C' \) with distance \( h \) by

\[ \text{Neighbor}_h(C') = \{ \mathcal{C} \mid \forall \mathcal{C} \in \mathcal{C}, \ D(C, C') = h \}. \]  

The intuition is that each tree can be queried independently, and we want to utilize such property by
limiting the number of affected trees at each iteration. \( \text{Neighbor}_h(\cdot) \) has a nice property where we
can increase \( h \) for larger search scope, or decrease \( h \) to improve speed. Observe that \( \text{Neighbor}_1(C') \)
is a subset of NaiveLeaf (minus invalid leaf tuples that requires an expensive model query), and a
superset of NaiveFeature (plus leaf tuples that may affect multiple features). In experiments we are
able to achieve good results with \( \text{Neighbor}_1(\cdot) \), and we provide an empirical greedy algorithm in
Appendix D.3 to estimate the minimal \( h \) required to reach the exact solution.

3.4 An Efficient Algorithm for Solving \( \text{Neighbor}_1(\cdot) \)

We propose Leaf Tuple attack (LT-Attack) in Algorithm 1 that efficiently solves Eq. (3) through two
additional concepts \( T_{\text{Bound}}(\cdot) \) and \( \text{Neighbor}_1^+(\cdot) \) as defined below. Let \( C' \) be any valid adversarial
tuple, and assume unique feature split thresholds. By definition tuples in \( \text{Neighbor}_1(C') \) are only
different from \( C' \) by one leaf, and we use \( \text{Neighbor}_1^+(C') \) to denote the neighborhood that has
different prediction leaf on tree \( t \). Formally

\[ \text{Neighbor}_1^+(C') = \{ \mathcal{C} \mid |\mathcal{C}^t \neq \mathcal{C}^{t'}|, \mathcal{C} \in \text{Neighbor}_1(C') \}. \]  

Definition 3. (Bound Trees) Let \( x' \in \mathcal{B}(C') \) be the example that minimizes \( \text{dist}_p(C', x_0) \), we denote
the indexes of trees that \( \text{bounds} x' \) by \( T_{\text{Bound}}(C') = \{ t \mid \text{OnEdge}(x', \mathcal{B}^{(t)}(\cdot)), \forall t \in \{1, \ldots, K\} \} \). Here \( \text{OnEdge}(x, \mathcal{B}) \) is true \( \text{iff} \. x \) equals to the left or the right bound of \( \mathcal{B} \) on at least one dimension.

Definition 4. (Advanced Neighborhood) We denote the set of neighborhood with smaller (advanced)
perturbation than \( C' \) by \( \text{Neighbor}_1^+(C') = \{ \mathcal{C} \mid \mathcal{C} \in \text{Neighbor}_1(C') \}, \text{dist}_p(C, x_0) < \text{dist}_p(C', x_0) \} \).

Theorem 2. (Bound Neighborhood) Let \( \text{Neighbor}_{\text{Bound}}^+(C') = \bigcup_{t \in T_{\text{Bound}}(C')} \text{Neighbor}_1^+(C'), \text{then} \text{Neighbor}_1^+(C') \subseteq \text{Neighbor}_{\text{Bound}}^+(C'). \)

Theorem 2 suggests that we can solve Eq. (3) by searching over \( \text{Neighbor}_{\text{Bound}}^+(C') \) since it is a
superset of the advanced neighborhood which leads to smaller perturbation. In general the algorithm
consists of an outer loop and an inner \( \text{Neighbor}_{\text{Bound}}^+(\cdot) \) function. The outer loop iterates until no
better adversarial example can be found, while the inner function generates bound neighborhood with
distance 1. The inner function computes \( T_{\text{Bound}} \) and runs the top-down traverse for each \( t \in T_{\text{Bound}} \)
with the intersection of other \( K - 1 \) bounding boxes, denoted by \( B^{(-t)} \). According to Theorem 1 a
leaf node of \( t \) is guaranteed to form a valid tuple if it has non-empty intersection with \( B^{(-t)} \).

To efficiently obtain \( B^{(-t)} \) we cache \( K \) bounding boxes in \( \mathcal{B'} \), and for each feature dimension we
maintain the sorted list of left and right bounds from \( K \) boxes respectively. Note that \( \mathcal{B'} \) of leaf node \( t \)
Algorithm 1: Our proposed LT-Attack for constructing adversarial examples.

Data: White-box model $f$, victim example $x_0, y_0$, initial adversarial example $x'$.

begin
  $r', C'$ ← dist$_p(C(x'), x_0), C(x')$;
  $B'$ ← BuildSortedBoxes($C', f$);
  $N_l$ ← Neighbors$_{Bound}(C', B', f)$;
  $\triangleright O(K\log K)$ - $B'$ maintains $K$ sorted bounding boxes on $d$ feature dimensions.
  has_better_neighbor ← True;
  while has_better_neighbor do
    $N_l$ ← Neighbors$_{Bound}(C', B', f)$;
    $\triangleright$ See complexity in $[\text{3.4.1}]$.
    $N'_l$ ← $\{C | C \in N_l,$ $f(C) \neq y_0\}$;
    $\triangleright O(|N_l|) - f(C)$ can be calculated from $f(C')$ in $O(1)$ using the diff leaf.
    $r'^*, C'^* ← \arg\min\{\text{dist}_p(C, x_0) | C \in N'_l\}$;
    $r'^*, C'^*$ can be calculated from $r'$ in $O(1)$ using the diff leaf.
    has_better_neighbor ← $r'^* < r'$;
    if has_better_neighbor then
      $r', C'$ ← $r'^*, C'^*$;
      $B'$ ← $B'.\text{ReplaceBox}(C')$;
      $\triangleright O(\log K)$ - Remove and add one box.
  end
  return $r', C'$
end

Function Neighbors$_{Bound}(C', B', f)$:

begin
  $N_l$ ← $\emptyset$;
  $T$ ← $T_{Bound}(C', B')$;
  $\triangleright O(d\log K)$ - Need $O(\log K)$ to get the first (tightest) tree on each dimension, and assume
  $C'$ caches the closest $x'$ to $x_0$. We give a closer complexity analysis for $|T_{Bound}(\cdot)|$
  in the following section.
  for $t \in T$ do
    $B^{(t-)} ← B'.\text{RemoveBox}(t)$;
    $\triangleright O(l \log K)$ - Remove the bounding box of
tree $t$ from the sorted list ( lazily), the
box has at most $l$ non-infinite dimensions.
    $I ← \{i | B^t \cap B^{(t-)} \neq \emptyset, i \in S(t) \setminus C^{(t)}\}$;
    $\triangleright O(2^t)$ - Traverse tree $t$ top-down and
return leaves for Neighbors$_1(C')$. $S(t)$
denotes the set of leaves of tree $t$.
    $N_l ← N_l \cup \{C' | C^{(t)} \leftarrow i, i \in I\}$;
    $\triangleright O(|I|)$ - Construct the neighborhood tuple
by replacing the $i_{th}$ leaf. In practice,
we only need to return the diff $(i, i)$.
  end
  return $N_l$
end

comes from feature split thresholds along the top-down traverse path, thus it has at most $l$ non-infinite
dimensions, where $l$ is the depth of the tree. In conclusion we can add/remove a bounding box $B'$
to/from $B'$ in $O(l \log K)$ time. We provide time complexity for most operations in
Algorithm 1 inline, and give a detailed analysis for the size of Neighbors$_{Bound}(C')$ in the next section. See $\text{Appendix D.1}$
for the algorithm generating random initial adversarial examples.

3.4.1 Size of the Bound Neighborhood

Our LT-Attack enumerates all leaf tuples in the bound neighborhood at each iteration, thus the complexity
of each iteration largely depends on the size of Neighbors$_{Bound}(C')$. In this section we
analyze the size $|\text{Neighbors$_{Bound}(C')$}|$ and show it will not be too large on real datasets.

Corollary 2. (Size of Neighbors$_1(C')$) Let $k^{(t)} = |\{\eta \in B^{(t-)}, (j, \eta) \in H(t)\}|$ be the number of
feature split thresholds inside $B^{(t-)}$, we have $|\text{Neighbors$_1(C')$}| \leq 2^{\min(k^{(t)}, l)} - 1$. Here $B^{(t-)} =
\bigcap_{i \in E', i \neq C^{(t)}} B^i$ and $H(t)$ denotes the set of feature split thresholds on all internal nodes of tree $t$.

In practice, $k^{(t)} \ll l$ since $B^{(t-)}$ is the intersection of $K - 1$ bounding boxes and only covers a small
region of the input space $\mathbb{R}^d$. $|T_{Bound}(C')| \leq d$ and is also usually small in real datasets, which can be
explained by the intuition that some features are less important and could reach the same value as $x_0$
easily. Both $|T_{Bound}(C')|$ and $|\text{Neighbors$_1(C')$}|$ characterize the complexity of $|\text{Neighbors$_{Bound}(C')$}|$.
We provide empirical statistics in $\text{Appendix A}$ which suggests that $|\text{Neighbors$_{Bound}(C')$}|$ has the
similar complexity as a single full model query. For instance, on the MNIST dataset with 784 features
and 400 trees we have $|\text{Neighbors$_{Bound}(C')$}| \approx 367.9$, and the algorithm stops in $\sim 159.4$
iterations when it cannot find a better neighborhood. As a comparison, decision based methods
usually require hundreds of full model queries per iteration to estimate the update direction.

3.4.2 Convergence Guarantee with Neighbors$_1(\cdot)$

In this section $\hat{C}$ denotes the converged tuple when the outer loop stops, and we discuss the property
of the converged solution. Trivially $\hat{C}$ has the minimal adversarial perturbation within Neighbors$_1(\hat{C})$,
and we can show that the guarantee is actually stronger.
Table 5: Average $\ell_\infty$ and $\ell_1$ perturbation of 500 test examples (or the entire test set when its size is less than 500) on standard (natural) GBDT models. Datasets are ordered by training data size. Bold and blue highlight the best and the second best entries respectively (not including MILP).

| Standard GBDT | SignOPT | HSIA | RBA-Appr | Cube | LT-Attack (Ours) | MILP | Ours vs. MILP |
|---------------|---------|------|----------|------|----------------|------|--------------|
| $\ell_\infty$ Perturbation | $r$ | time | $r$ | time | $r$ | time | $r_{avg}$ | $r$ | time | $r_{avg}$ | $r$ | time | $r_{avg}$ | Speedup |
| breast-cancer | 350 | 0.18 | 256 | 0.07 | 427 | 0.0008 | 530 | 230s | 0.25 | 0.01s | 222 | 0.01s | 1.06 | 13X |
| diabetes | 083 | 0.343 | 078 | 0.066s | 113 | 0.0009 | 080 | 240s | 0.059 | 0.002s | 056 | 0.004s | 1.05 | 42X |
| MNIST-6 | 480 | 2.73 | 277 | 1.23s | 963 | 0.155 | 143 | 2.43s | 0.097 | 0.022s | 065 | 0.287s | 1.49 | 129.3X |
| icm | 043 | 0.313 | 043 | 0.096s | 074 | 0.020s | 035 | 334s | 0.033 | 0.007s | 031 | 0.060s | 1.06 | 942.9X |
| MNIST | 195 | 3.70 | 1.17s | 28.7s | 983 | 4.11s | 056 | 4.42s | 0.029 | 2.37s | 014 | 0.275s | 1.07 | 540.3X |
| F-MNIST | 155 | 4.38 | 0.067 | 1.81s | 607 | 5.55s | 038 | 5.45s | 0.028 | 0.370s | 013 | 0.15min* | 2.15 | 2473X |
| covtype | 047 | 5.08s | 074 | 2.09s | 086 | 3.05s | 036 | 958s | 0.032 | 0.383s | 028 | 0.10min* | 1.14 | 15736.8X |
| HIGGS | 009 | 465.5 | 012 | 157s | 099 | 55.3s | 005 | 862s | 0.004 | 0.365s | 004 | 0.10min* | 1.00 | 81766.7X |

Theorem 3. (Convergence Guarantee) Let $V^+ = \{i \mid i \in C, C \in \text{Neighbor}_1(\bar{C})\}$ be the union of leaves appeared in the advanced neighborhood $\text{Neighbor}_1(\bar{C})$ (Definition 4), then

\[ \bar{C} \] is the optimal adversarial tuple within valid combinations of $V^+$.  

Note that $V^+$ is the union of leaves, and leaves from multiple tuples of distance 1 could form a new valid tuple of larger distance to $\bar{C}$ (by combining the different leaves together). In other words Theorem 3 suggests that our solution is not only a local optimal in $\text{Neighbor}_1(\bar{C})$, but also better than certain tuples in $\text{Neighbor}_h(\bar{C})$ with $h > 1$. In our illustrated example Fig. 1 assume the algorithm converged at $\bar{C} = C(a) = (4, 5, 9)$ on $\ell_\infty$ norm, here $\text{Neighbor}_1(\bar{C}(a)) \subseteq \{(4, 8, 9), (4, 5, 10)\}$. Theorem 3 claims that there is no better adversarial tuple from any valid combinations within $V^+ = \{(4, 5, 8, 9, 10)\}$ such as $(4, 8, 10)$, even though it is from $\text{Neighbor}_2(\bar{C})$.

3.5 Implementation Details

For the initial point, we draw 20 random initial adversarial examples from a Gaussian distribution, and optimize with a fine-grained binary search before feeding to the proposed algorithm. We return the best adversarial example found among them (see Appendix D.1 for details). The ensemble is likely to contain duplicate feature split thresholds even though it’s defined on $\mathbb{R}^d$, for example it may come from the image space $[255]^d$ and scaled to $\mathbb{R}^d$. Duplicate split thresholds are problematic since we cannot move across the threshold without affecting multiple trees, and to overcome the issue we use a relaxed version of $\text{Neighbor}_1(\cdot)$ to allow changing multiple trees at one iteration, as long as it’s caused by the same split threshold. When searching for the best neighborhood it’s likely to have perturbation ties in $\ell_\infty$ and $\ell_1$ norm, in this case we use a secondary $\ell_2$ norm to break the tie. Eq. (3) looks for the best tuple across all neighborhood which may be unnecessary at early stage of iterations. To improve the efficiency we sort feature dimensions by $\text{abs}(x' - x_0)$ (large first), and terminate the search earlier if a better tuple was found in the top 1 feature. To escape converged local minimums we change each coordinate to a nearby value from Gaussian distribution with 0.1 probability, and continue the iteration if a better adversarial example was found within 300 trials.

4 Experimental Results

We evaluate the proposed algorithm on 9 public datasets [Smith et al., 1988; Lecun et al., 1998; Chang, Lin] [2011; Wang et al., 2012; Baldi et al., 2014; Xiao et al., 2017; Dua, Graff] with both the standard (natural) GBDT and RF models, and on an additional 10th dataset (Bosch) with
We run our experiments with 20 threads per task. Conventionally black-box attacks measure efficiency by the number of queries, here we compare running time since it’s difficult to quantify queries for white-box attacks. To minimize the efficiency variance between programming languages we feed the robustly trained GBDT models. Datasets are ordered by training data size. Bold and blue highlight the best and the second best entries respectively (not including MILP). (***/’*’): Average of 1000 / 500 examples due to long running time.

The robustly trained GBDT. Datasets have a mix of small/large scale and binary/multi classification (statistics in Appendix A), and are normalized to [0, 1] to make results comparable across datasets. We order datasets by training data size in all of our tables, where HIGGS is the largest dataset with 10.5 million training examples. All GBDTs were trained using the XGBoost framework (Chen, Guestrin, 2016) and we use the models provided by Chen et al. (2019b) as target models (except bosch). We order datasets by training data size in all of our tables, where HIGGS is the largest dataset with 10.5 million training examples. All GBDTs were trained using the XGBoost framework (Chen, Guestrin, 2016) and we use the models provided by Chen et al. (2019b) as target models (except bosch). We compare with the following existing adversarial attacks that are applicable to tree ensembles:

- **SignOPT (Cheng et al., 2020)**: The decision based attack that constructs adversarial examples based on hard-label black-box queries. We report the average distortion, denoted as $\bar{r}$ in the results (since the norm of adversarial example is an upper bound of minimal adversarial perturbation $r^\ast$).

- **HSJA (Chen et al., 2019c)**: Another decision based attack for constructing adversarial examples.

- **RBA-Appr (Yang et al., 2019)**: An approximate attack for tree ensembles that constructs adversarial examples by searching over training examples of the opposite class.

- **Cube [Andrejuschken, Hein, 2019]**: An empirical attack for tree ensembles that constructs adversarial examples by stochastically changing a few coordinates to the $\ell_{\infty}$ boundary, and accepts the change if it decreases the functional margin. The method provides good experimental results in general, but lacks theoretical guarantee and could be unreliable on certain datasets such as breast-cancer. Cube doesn’t support $\ell_2$ objective by default and we report the $\ell_2$ perturbation of the constructed adversarial examples from $\ell_{\infty}$ objective attacks.

- **LT-Attack (Ours)**: Our proposed attack that constructs adversarial examples for tree ensembles. We report the average distortion of the adversarial examples, denoted as $\bar{r}$ in the results.

- **MILP (Kantarcioglu et al., 2015)**: The mixed-integer linear programming based method provides the exact minimal adversarial perturbation $r^\ast$ but could be very slow on large models.

We run our experiments with 20 threads per task. Conventionally black-box attacks measure efficiency by the number of queries, here we compare running time since it’s difficult to quantify queries for white-box attacks. To minimize the efficiency variance between programming languages we feed an XGBoost model (Chen, Guestrin, 2016) to SignOPT, HSJA, and Cube, which has an efficient C++ implementation and supports multi-threading batch query. MILP uses a thin wrapper around the Gurobi Solver (Gurobi Optimization, 2020). Baseline methods spend majority of time on XGBoost model inference rather than Python code. For instance, on Fashion-MNIST, SignOPT, HSJA, Cube spent 72.8%, 57.3%, 73.4% of runtime in XGBoost library (C++) calls, respectively. HSJA and
Table 5: Average $\ell_2$ perturbation over 100 test examples on the standard (natural) random forests (RF) models. Datasets are ordered by training data size. Bold and blue highlight the best and the second best entries respectively (not including MILP).

| Model                  | Train Size | Standard RF | Cube | LT-Attack (Ours) | MILP | Ours vs. MILP |
|------------------------|------------|-------------|------|------------------|------|---------------|
| noise2-6               |            | 1.03        | .224 | .415             | .001 | 1.03          |
| diabetes               |            | 2.60        | .285 | .151             | .003 | 1.03          |
| MNIST2-6               |            | .439        | 2.13s| .207             | .045 | 1.07          |
| ijcnn                  |            | .046        | .336 | .028             | .003 | 1.00          |
| MNIST                  |            | .057        | 2.88s| .018             | .057 | 1.00          |
| F-MNIST                |            | 1.14        | 3.51s| .066             | .080 | 1.00          |
| webspam                |            | .005        | .704 | .003             | .033 | 1.00          |
| covtype                |            | .087        | .700 | .055             | .040 | 1.00          |
| HIGGS                  |            | .015        | .423 | .009             | .013 | 1.00          |

SignOPT start with 1 initial adversarial example and run 100~632 and 200 queries per iteration respectively to approximate the gradient, and Cube uses 20 initial examples which utilizes batch query. In Table 3 and Table 4 we show the empirical comparisons on $\ell_\infty$ and $\ell_2$, and provide $\ell_1$ results as well as the attack success rate in Appendix B due to the space limit. We can see that our method provides a tight upper bound $\bar{r}$ compared to the exact $r^*$ from MILP, which means that the adversarial examples found are very close to the one with minimal adversarial perturbation, and our method achieved 1,000~80,000x speedup on some large models such as HIGGS. Our bound is tight especially in the $\ell_2$ case. For instance, on Fashion-MNIST our $\bar{r}/r^*$ ratio is 1.49x and 1.33x for standard and robustly trained models respectively, while the respective Cube ratio is 4.59x and 2.15x using $\sim$21x time, and the respective HSJA ratio is 11.86x and 5.75x using $\sim$4.5x time. For completeness we also include verification results (Chen et al., 2019b; Wang et al., 2020) in Appendix B since they are not directly comparable to adversarial attacks (they output lower bounds for the minimal adversarial perturbation while attacks aim to output an upper bound). To demonstrate the capability on general tree based ensembles we also conduct experiments on the standard (natural) random forests (RFs). We present the average $\ell_2$ perturbation in Table 5 where the $\bar{r}_{out}/r^*$ ratio is close to 1 across all datasets. Additional experimental results on $\ell_\infty$ perturbation can be found in Appendix B.1.

To study the impact of using different number of initial examples, we conduct the experiments with {1, 2, 4, 6, 10, 20, 40} initial examples on SignOPT, HSJA, Cube, and LT-Attack, allocating 2 threads per task. We report the smallest (best) adversarial perturbation among those initial examples. Using more initial examples could lead to smaller (better) adversarial perturbation, but requires linearly increasing computational cost. Fig. 2 presents the $\ell_2$ perturbation vs. runtime per test example in log scale, where our method is able to construct small (good) adversarial examples (y-axis) with a few initial examples, and can be orders of magnitude faster than other methods in the meantime (x-axis).

![Figure 2: Average $\ell_2$ perturbation of 50 test examples vs. runtime per test example in log scale. Methods on the bottom-left corner are better.](image-url)
Broader Impact

To the best of our knowledge, this is the first practical attack algorithm (in terms of both computational time and solution quality) that can be used to evaluate the robustness of tree ensembles. The study of robustness training algorithms for tree ensemble models have been difficult due to the lack of attack tools to evaluate their robustness, and our method can serve as the benchmark tool for robustness evaluation (similar to FGSM, PGD and C&W attacks for neural networks) [Goodfellow et al. 2015] [Madry et al. 2018] [Carlini, Wagner 2017] to stimulate the research in the robustness of tree ensembles.

Acknowledgments and Disclosure of Funding

We acknowledge the support by NSF IIS-1901527, IIS-2008173, ARL-0011469453, Google Cloud and Facebook.

References

Andriushchenko Maksym, Hein Matthias. Provably robust boosted decision stumps and trees against adversarial attacks // Advances in Neural Information Processing Systems 32. 2019. 13017–13028.

Athalye Anish, Carlini Nicholas, Wagner David A. Obfuscated Gradients Give a False Sense of Security: Circumventing Defenses to Adversarial Examples // ICML. 2018.

Baldi Pierre, Sadowski Peter, Whiteson D. O. Searching for exotic particles in high-energy physics with deep learning. // Nature communications. 2014. 5. 4308.

Bosch. Bosch Production Line Performance. 2016. https://www.kaggle.com/c/bosch-production-line-performance/data.

Brendel Wieland, Rauber Jonas, Bethge Matthias. Decision-Based Adversarial Attacks: Reliable Attacks Against Black-Box Machine Learning Models // International Conference on Learning Representations. 2018.

Brunner Thomas, Diehl Frederik, Truong-Le Michael, Knoll Alois. Guessing Smart: Biased Sampling for Efficient Black-Box Adversarial Attacks // 2019 IEEE/CVF International Conference on Computer Vision (ICCV). 2018. 4957–4965.

Carlini Nicholas, Wagner David A. Towards Evaluating the Robustness of Neural Networks // 2017 IEEE Symposium on Security and Privacy (SP). 2017. 39–57.

Chang Chih-Chung, Lin Chih-Jen. LIBSVM: A library for support vector machines // ACM Transactions on Intelligent Systems and Technology. 2011. 2. 27:1–27:27. Software available at http://www.csie.ntu.edu.tw/~cjlin/libsvm.

Chen Hongge, Zhang Huan, Boning Duane S., Hsieh Cho-Jui. Robust Decision Trees Against Adversarial Examples // ICML. 2019a. 1122–1131.

Chen Hongge, Zhang Huan, Si Si, Li Yang, Boning Duane, Hsieh Cho-Jui. Robustness Verification of Tree-based Models // Advances in Neural Information Processing Systems 32. 2019b. 12317–12328.

Chen Jianbo, Jordan Michael I., Wainwright Martin J. HopSkipJumpAttack: A Query-Efficient Decision-Based Adversarial Attack // arXiv preprint arXiv:1904.02144. 2019c.

Chen Pin-Yu, Zhang Huan, Sharma Yash, Yi Jinfeng, Hsieh Cho-Jui. ZOO: Zeroth Order Optimization Based Black-box Attacks to Deep Neural Networks without Training Substitute Models // Proceedings of the 10th ACM Workshop on Artificial Intelligence and Security. 2017.

Chen Tianqi, Guestrin Carlos. XGBoost: A Scalable Tree Boosting System // Proceedings of the 22nd ACM SIGKDD International Conference on Knowledge Discovery and Data Mining. New York, NY, USA: Association for Computing Machinery, 2016. 785–794. (KDD ’16).
Yang Yao-Yuan, Rashtchian Cyrus, Wang Yizhen, Chaudhuri Kamalika. Robustness for Non-Parametric Classification: A Generic Attack and Defense. 2019.

Zhang Fuyong, Wang Yi, Liu Shigang, Wang Hua. Decision-based evasion attacks on tree ensemble classifiers // World Wide Web. 04 2020.

Zhang Huan, Si Si, Hsieh Cho-Jui. GPU-acceleration for Large-scale Tree Boosting // arXiv preprint arXiv:1706.08359. 2017.
A Dataset and Model Statistics

We use 9 datasets and pre-trained models provided in Chen et al. (2019b), which can be downloaded from [GitHub](https://github.com/chenhongge/RobustTrees). Table 6 summarizes the statistics of the datasets as well as the standard (natural) GBDT models, and we report the average complexity statistics for $|\text{Neighbor}_{\text{Bound}}(\cdot)|$ from 500 test examples. For multi-class datasets we count trees either belong to the victim class or the class of the initial adversarial example. Datasets may contain duplicate feature split thresholds and the extra complexity is covered in the statistics. We disabled the random noise optimization discussed in §3.5 to provide a cleaner picture of Algorithm 1. We train standard (natural) RF models using XGBoost’s native RF APIs\(^1\) and provide the statistics in Table 7.

| Dataset       | features | classes | trees | depth $l$ | iterations | $|T_{\text{Bound}}(\cdot)|$ | $|\text{Neighbor}_{\text{Bound}}(\cdot)|$ |
|---------------|----------|---------|-------|-----------|------------|-----------------------------|----------------------------------|
| breast-cancer | 10       | 2       | 4     | 6         | 2.1        | 3.2                      | 5.2                              | 9.2                             |
| diabetes      | 8        | 2       | 20    | 5         | 6.3        | 6.1                      | 3.4                              | 10.6                            |
| MNIST2-6      | 784      | 2       | 1,000 | 4         | 121.7      | 374.2                   | 14.9                             | 256.5                           |
| icnn          | 22       | 2       | 60    | 8         | 26.5       | 7.4                      | 3.3                              | 17.8                            |
| MNIST         | 784      | 10      | 400   | 8         | 159.4      | 124.7                   | 5.0                              | 367.9                           |
| F-MNIST       | 784      | 10      | 400   | 8         | 236.8      | 149.1                   | 6.5                              | 717.4                           |
| webspam       | 54       | 7       | 160   | 8         | 36.7       | 30.8                    | 10.6                             | 39.2                            |
| HIGGS         | 28       | 2       | 300   | 8         | 107.1      | 13.5                    | 2.1                              | 24.0                            |

Table 7: Parameters and statistics for datasets and the standard (natural) RFs.

| Dataset       | train size | test size | trees | depth $l$ | subsampling | test acc. |
|---------------|------------|-----------|-------|-----------|-------------|-----------|
| breast-cancer | 546        | 137       | 4     | 6         | .8          | .974      |
| diabetes      | 614        | 154       | 25    | 8         | .8          | .775      |
| MNIST2-6      | 11,876     | 1,990     | 1000  | 4         | .8          | .963      |
| icnn          | 49,990     | 91,701    | 100   | 8         | .8          | .919      |
| MNIST         | 60,000     | 10,000    | 400   | 8         | .8          | .907      |
| F-MNIST       | 60,000     | 10,000    | 400   | 8         | .8          | .823      |
| webspam       | 300,000    | 50,000    | 100   | 8         | .8          | .948      |
| covtype       | 400,000    | 180,000   | 160   | 8         | .8          | .745      |
| HIGGS         | 10,500,000 | 500,000   | 300   | 8         | .8          | .702      |

B Supplementary Experiments

B.1 Additional Experimental Results on Random Forests

Table 8: Average $\ell_{\infty}$ perturbation over 100 test examples on the standard (natural) random forests (RF) models. Datasets are ordered by training data size. Bold and blue highlight the best and the second best entries respectively (not including MILP).

| $\ell_{\infty}$ Perturbation | Standard RF | Cube | LT-Attack (Ours) | MILP | Ours vs. MILP |
|-----------------------------|-------------|------|-----------------|------|--------------|
|                             | $\bar{r}$  | time | $r_{\text{cut}}$ | time | $r^{*}$ | $r_{\text{cut}}/r^{*}$ | Speedup |
| breast-cancer               | .797        | .208s | .340           | .01s | .332       | .008s         | 1.02     | 8X          |
| diabetes                    | .159        | .271s | .111           | .002s| .103       | .054s         | 1.08     | 27X         |
| MNIST2-6                    | .135        | .185s | .130           | .041s| .121       | .335s         | 1.07     | 8.2X        |
| icnn                        | .032        | .340s | .026           | .003s| .026       | .338s         | 1.00     | 112.7X      |
| MNIST                       | .017        | .198s | .010           | .056s| .009       | 21.4s         | 1.11     | 382.1X      |
| F-MNIST                     | .036        | 2.57s | .036           | .084s| .032       | 34.1s         | 1.13     | 406X        |
| webspam                     | .004        | .652s | .002           | .023s| .002       | 2.63s         | 1.00     | 114.3X      |
| covtype                     | .050        | .684s | .048           | .037s| .048       | 72.2s         | 1.00     | 1951.4X     |
| HIGGS                       | .008        | .389s | .007           | .011s| .006       | 20.9s         | 1.17     | 1900X       |

B.2 Attack Success Rate

We present attack success rate in Fig. 3 which is calculated as the ratio of constructed adversarial examples that have smaller perturbation than the thresholds. We use 50 test examples for MILP due to long running time, and 500 test examples for other methods.

\(^1\)https://xgboost.readthedocs.io/en/release_1.0.0/tutorials/rf.html
B.3 $\ell_\infty$ Perturbation Using Different Number of Initial Examples

Fig. 4 presents the average $\ell_\infty$ perturbation of 50 test examples vs. runtime per test example in log scale. We plot the results for SignOPT, HSJA, Cube, and LT-Attack on \{1, 2, 4, 6, 10, 20, 40\} initial examples, using 2 threads per task. Initial examples are not applicable to RBA-Appr and MILP thus we only plot a single point for each method.

B.4 Experimental Results for $\ell_1$ Norm and Verification

Table 9 presents the experimental results on $\ell_1$ norm perturbation. Cube doesn’t support $\ell_1$ objective by default and we report the $\ell_1$ perturbation of the constructed adversarial examples from $\ell_\infty$ objective attacks. For completeness we include verification results [Chen et al. 2019b, Wang et al. 2020] in Table 9 and Table 10 which output lower bounds of the minimal adversarial perturbation denoted as $\hat{\ell}$ (in contrast to adversarial attacks that aim to output an upper bound $\bar{r}$).
Table 9: Average $\ell_1$ perturbation over 50 test examples on the standard (natural) GBDT models and robustly trained GBDT models. Datasets are ordered by training data size. Bold and blue highlight the best and the second best entries respectively (not including MILP and Verification).

| Standard GBDT | SignOPT | RBA-Appr | Cube | LT-Attack (Ours) | MILP | Verification | Ours vs. MILP |
|---------------|---------|----------|------|----------------|------|--------------|--------------|
| Perturbation  | $r$     | time     | $r$  | time           | $r_{\text{avg}}$ | time | $r_{\text{avg}}$ | Speedup |
| breast-cancer | 413     | .68s     | 535  | .0002s         | .372 | .002s        | .372 | .002s | 1.00 | 6.X |
| diabetes      | .183    | 1.04s    | .294 | .0005s         | .131 | .006s        | .126 | .005s | 1.13 | 26.7X |
| MNIST-2       | 35.4    | 6.81s    | 25.9 | .109s          | .783 | .230s        | .568 | .251s | .571 10.9X |
| icnn          | .075    | .899s    | .286 | .016s          | .064 | .008s        | .060 | .032s | .044 1.68s | 1.07 410.3X |
| MNIST         | 22.8    | 8.77s    | 46.5 | 3.13s          | .341 | .267s        | .207 | .568s | 1.36 212.7X |
| F-MNIST       | 13.8    | 9.55s    | 42.7 | 4.57s          | .600 | .479s        | .181 | 70.3s | .136 14.68X |
| webspm        | 287     | 3.43s    | 614  | .630s          | .066 | .462s       | .004 | 14.73s | .003 6.40s | 1.50 67.3X |
| covtype       | .074    | 1.74s    | 223  | 2.92s          | .057 | .039s        | .052 | 31.4s | .024 2.26s | 1.10 8051.3X |
| HIGGS         | .050    | 1.26s    | 611  | 4.47s          | .001 | .049s       | .013 | 25min | .002 7.36s | 1.23 30183.7X |

Table 10: Average $\ell_\infty$ and $\ell_2$ perturbation over 500 test examples on the standard (natural) GBDT models and robustly trained GBDT models. (**): Average of 50 examples due to long running time.

| Standard GBDT | LT-Attack (Ours) | MILP | Verification | Ours vs. MILP |
|---------------|-----------------|------|--------------|--------------|
| Perturbation  | $r_{\text{avg}}$ | time | $r_{\text{avg}}$ | Speedup |
| breast-cancer | .253 | .001s | .222 | .013s | .220 | .002s | 1.00 4.X |
| diabetes      | .059 | .002s | .056 | .004s | .047 | .010s | 1.00 4.X |
| MNIST-2       | .097 | .007s | .097 | .011s | .093 | .012s | 1.00 4.X |
| icnn          | .029 | .007s | .031 | .011s | .027 | .012s | 1.00 4.X |
| MNIST         | .282 | .007s | .140 | .011s | .135 | .012s | 1.00 4.X |
| F-MNIST       | .098 | .003s | .081 | .005s | .075 | .014s | 1.00 4.X |
| covtype       | .038 | .008s | .036 | .011s | .033 | .013s | 1.00 4.X |
| HIGGS         | .004 | .006s | .004 | .010s | .003 | .013s | 1.00 4.X |

| Standard GBDT | LT-Attack (Ours) | MILP | Verification | Ours vs. MILP |
|---------------|-----------------|------|--------------|--------------|
| Perturbation  | $r_{\text{avg}}$ | time | $r_{\text{avg}}$ | Speedup |
| breast-cancer | .283 | .001s | .280 | .011s | .277 | .002s | 1.00 4.X |
| diabetes      | .077 | .003s | .073 | .005s | .058 | .048s | 1.00 4.X |
| MNIST-2       | .245 | .235s | .183 | .252s | .180 | .076s | 1.00 4.X |
| icnn          | .044 | .010s | .043 | .218s | .030 | 4.62s | 1.00 4.X |
| MNIST         | .072 | .243s | .043 | .325s | .013 | 6.81s | 1.00 4.X |
| F-MNIST       | .073 | .006s | .070 | .257s | .023 | 2.96s | 1.00 4.X |
| webspm        | .002 | .053s | .002 | 3.17s | .002 | 8.95s | 1.00 4.X |
| covtype       | .045 | .039s | .042 | .257s | .023 | 2.96s | 1.00 4.X |
| HIGGS         | .008 | .007s | .007 | .131s | .002 | 10.3s | 1.00 4.X |

C Proofs

C.1 Proof of Theorem 2

**Proof.** By contradiction. Given adversarial tuple $C'$ and victim example $x_0$, assume

$$\exists C_1 \in \text{Neighbor}^+_1(C') \quad \text{s.t.} \quad C_1 \notin \text{Neighbor}^\text{Bound}(C').$$

Assume $p \in \{1, 2, \infty\}$. Let $x_1 = \text{argmin}_{x \in B(C_1)} \|x - x_0\|_p$, $x' = \text{argmin}_{x \in B(C')} \|x - x_0\|_p$, and let $J$ be the set of dimensions that $x_1$ is closer to $x_0$:

$$J = \{ j \mid \|x_{1,j} - x_{0,j}\| < \|x'_{j} - x_{0,j}\| \}.$$  

According to the definition of $\text{Neighbor}^+_1(C')$ we have $\text{dist}_p(C_1, x_0) < \text{dist}_p(C', x_0)$, and consequently $J \neq \emptyset$. We choose any $j \in J$ and for cleanliness we use $\langle l', r' \rangle$ to denote the interval from $B(C')$ on $j_{th}$ dimension, and let $d_0 = x_{0,j}$, $d_1 = x_{1,j}$, $d' = x'_{j}$.  

15
Observe \( d_0 \not\in (l', r'] \), otherwise we have \( |d' - d_0| = 0 \) and \( |d_1 - d_0| \) cannot be smaller. W.l.o.g. assume \( d_0 \) is on the right side of the interval, i.e., \( r' < d_0 \), then according to the argmin property of \( x' \) we have \( d' = r' \).

Recall that \( B(C') \) is the intersection of \( K \) bounding boxes from the ensemble, then

\[
\exists t' \in [K] \quad \text{s.t.} \quad B_{C'}^{\min} = r' = r'.
\]

Observe \( r' = d' < d_1 \) since \( d_0 \) is on the right side of \( d' \) and \( d_1 \) has smaller distance to \( d_0 \), which means \( C_1 \) has different leaf than \( C' \) on tree \( t \). Also according to the above equation \( t' \in T_{B, \text{Bound}}(C') \), and \( C_1 \in \text{Neighbor}_1^{(t)}(C') \), thus \( C_1 \in \cup_{t \in T_{\text{Bound}}(C')} \text{Neighbor}_1^{(t)}(C') = \text{Neighbor}_{\text{Bound}}(C') \), contradiction.

\[ \square \]

C.2 Proof of Corollary 2

**Proof.** According to [Theorem 1] \( B^{(-t)} \) is the intersection of \( K - 1 \) bounding boxes and can be written as the Cartesian product of \( d \) intervals, we call it a box. Each tree \( t \) of depth \( l \) splits the \( \mathbb{R}^d \) space into up to \( 2^l \) non-overlapping axis-aligned boxes, and \( |\text{Neighbor}_{1}^{(t)}(C')| + 1 \) (plus current box) corresponds to the number of boxes that has non-empty intersection with \( B^{(-t)} \), thus \( |\text{Neighbor}_{1}^{(t)}(C')| \leq 2^l - 1 \).

Observe that \( k^{(t)} \) axis-aligned feature split thresholds can split \( \mathbb{R}^d \) into at most \( 2^{k^{(t)}} \) non-overlapping boxes, assuming \( d \geq k^{(t)} \), and the maximum can be reached by having at most 1 split threshold on each dimension. In conclusion there are at most \( 2^\min(k^{(t)}, l) \) boxes that has non-empty intersection with \( B^{(-t)} \), thus \( |\text{Neighbor}_{1}^{(t)}(C')| \leq 2^\min(k^{(t)}, l) - 1 \) (minus the current box).

\[ \square \]

C.3 Proof of Theorem 3

**Proof.** By contradiction. Let \( x_0, y_0 \) be the victim example and assume

\[
\exists C^* \in (V^+)^K \cap C \quad \text{s.t.} \quad f(C^*) \neq y_0 \land \text{dist}_p(C^*, x_0) < \text{dist}_p(C', x_0).
\]

Assume \( p \in \{1, 2, \infty\} \). Recall \( f(C) = \text{sign}(\sum_{t \in C} v^i) \), we compute the tree-wise prediction difference

\[
v^{(t)} = v^{C^*} - v^{C'}, \quad t \in [K].
\]

Let \( t_{\min} \) be the tree with the smallest functional margin difference

\[
t_{\min} = \arg\min_{t \in [K]} y_0 \cdot v^{(t)}.
\]

We construct a tuple \( C_1 \) which is the same as \( C' \) except on \( t_{\min} \), where \( C_1^{(t_{\min})} = C^*(t_{\min}) \), consequently we have

\[
C_1 \in C \land \text{dist}_p(C_1, x_0) < \text{dist}_p(C', x_0).
\]

Now we show \( f(C_1) \neq y_0 \), or \( y_0 \sum_{t \in C_1} v^i < 0 \):

i. Case \( y_0 \cdot v^{(t_{\min})} \leq 0 \). Then

\[
y_0 \sum_{t \in C_1} v^i = y_0 \sum_{t \in C'} v^i + y_0 \cdot v^{(t_{\min})} \leq y_0 \sum_{t \in C'} v^i < 0.
\]

ii. Case \( y_0 \cdot v^{(t_{\min})} > 0 \). Then

\[
y_0 \sum_{t \in C_1} v^i = y_0 \sum_{t \in C'} v^i + y_0 \cdot v^{(t_{\min})} < y_0 \sum_{t \in C'} v^i + Ky_0 \cdot v^{(t_{\min})}
\]

\[
\leq y_0 \sum_{t \in C'} v^i + y_0 \sum_{t \in [K]} v^{(t)} = y_0 \sum_{t \in C^*} v^i < 0.
\]

In conclusion \( C_1 \) is a valid adversarial tuple within \( \text{Neighbor}_1(C') \) and has smaller perturbation than \( C' \), thus the algorithm won’t stop.

\[ \square \]
D Supplementary Algorithms

D.1 Generating Initial Adversarial Examples for LT-Attack

Algorithm 2: Generating Initial Adversarial Examples for LT-Attack

**Data:** Target white-box model $f$, victim example $x_0$.

1. $y_0 \leftarrow f(x_0)$;
2. $r', C' \leftarrow \text{MAX, None}$;
3. $\text{num\_attack} \leftarrow 20$
4. for $i \leftarrow 1, \ldots, \text{num\_attack}$ do
5.   do
6.     $x' \leftarrow x_0 + \text{Normal}(0, 1)^d$;
7.     while $f(x') = y_0$;
8.     $x' \leftarrow \text{BinarySearch}(x', x_0, f)$;
9.     $r^*, C^* \leftarrow \text{LT-Attack}(f, x_0, x')$;
10.    if $r^* < r'$ then
11.       $r', C' \leftarrow r^*, C^*$;
12.   end
13. end
14. return $r', C'$

D.2 Algorithm for NaiveLeaf

Algorithm 3: Compute NaiveLeaf

**Data:** Target white-box model $f$, current adversarial example $x'$, victim example $x_0$.

**Result:** The NaiveLeaf neighborhood of $C(x')$.

1. $(i^{(1)}, \ldots, i^{(K)}) \leftarrow C(x')$;
2. $N \leftarrow \emptyset$
3. for $t \leftarrow 1 \ldots K$ do
4.   for $i \in S^{(t)}, i \neq i^{(t)}$ do
5.     $x_{\text{new}} \leftarrow x'$;
6.     for $j, (l, r) \in B^t$ do
7.       $\triangleright S^{(t)}$ denotes the leaves of tree $t$.
8.       $x_{\text{new}} \leftarrow x'$;
9.     end
10.    end
11.   end
12. $N \leftarrow N \cup \{C(x_{\text{new}})\}$;
13. return $N$

D.3 A Greedy Algorithm to Estimate the Minimum Neighborhood Distance

To understand the quality of constructed adversarial examples we use an empirical greedy algorithm to estimate the minimum neighborhood distance $h$ such that $\text{Neighbor}_h(\cdot)$ can reach the exact solution. Assume our method converged at $C'$ and the optimum solution is $C^*$, let $T_{\text{diff}} = \{t \mid C'^{(t)} \neq C^{*(t)}\}$ be the set of trees with different prediction, then the Hamming distance $\bar{h} = |T_{\text{diff}}|$ is a trivial upper bound where $\text{Neighbor}_{\bar{h}}(C')$ can reach $C^*$ with a single addition iteration. To estimate a realistic $h$~— we want to find the disjoint split $T_{\text{diff}} = \bigcup_{t \in \{1, \ldots, K\}} T_t$ such that we can mutate $C'$ into $C^*$ by changing trees in $T_t$ to match $C^*$ at $t_{\text{th}}$ iteration. We make sure all intermediate tuples are valid and has strictly
decreasing perturbation as required by Eq. (3) and report \( h^* = \max_{t \in [k]} |T_t| \). \( h^* \) is an estimation of the minimum \( h \) because we cannot guarantee the \( \arg\min \) constrain due to the large complexity. As shown in Table 11 we have median \( h \) = 23 and median \( h^* \) = 8 on ensemble with 300 trees (HIGGS), which suggests that our method is likely to reach the exact optimum on half of the test examples through \( \sim 3 \) additional iterations on \( \text{Neighborhood} \). In this experiment we disabled the random noise optimization discussed in §3.5 to provide a cleaner picture of Algorithm 1.

Algorithm 4: A greedy algorithm to estimate the minimum neighborhood distance \( h^* \).

Data: The model \( f \), our adversarial point \( x_{out} \), exact MILP solution \( x^* \).

Result: An estimation of neighborhood distance \( h^* \).

begin
1. \( C_{out}, C^* \leftarrow C(x_{out}), C(x^*) \);
2. \( r^* \leftarrow \text{dist}_p(C^*, x_0) \);
3. \( y^* \leftarrow f(x^*) \);
4. \( h_{\text{min}} \leftarrow D(C_{out}, C^*) ; \quad \triangleright \text{Hamming distance is the upper bound.} \)
5. \( \mathcal{I}_{\text{diff}} \leftarrow \{ (v^{C^*(t)} - v^{C_{out}(t)}, C^{(t)}_{out}, C^{(t)}_{out}) \mid C^{(t)}_{out} \neq C^{(t)}_{out}, t \in [K] \} ; \quad \triangleright \mathcal{I}_{\text{diff}} \text{ is the list of tuples in the form of (label diff, our leaf, MILP leaf).} \)
6. \( \text{num} \text{ trial} \leftarrow 200 ; \)
7. for \( i \leftarrow 1, \ldots, \text{num trial} \) do
8. \( h \leftarrow 0 ; \)
9. \( \mathcal{I} \leftarrow \text{shuffle}(\mathcal{I}_{\text{diff}}) ; \)
10. \( C_{\text{tmp}} \leftarrow C_{\text{out}} ; \)
11. while \( \mathcal{I} \neq \emptyset \) do
12. \( r_{\text{last}} \leftarrow \text{dist}_p(C_{\text{tmp}}, x_0) ; \)
13. \( C_{\text{tmp}} \leftarrow \text{pop the first tuple from } \mathcal{I} \text{ with positive label diff and replace with MILP leaf} ; \)
14. \( h \leftarrow h + 1 ; \)
15. while \( C_{\text{tmp}} \notin C \text{ or dist}_p(C_{\text{tmp}}, x_0) \notin [r^*, r_{\text{last}} \text{ or } f(C_{\text{tmp}}) \neq y^* \) do
16. \( \triangleright \text{Making sure } C_{\text{tmp}} \text{ satisfies Equation except the argmin. We cannot guarantee argmin due to the high complexity. The while loop is guaranteed to exit since we can pop all tuples in } \mathcal{I} \text{ to become } C^* . \)
17. \( C_{\text{tmp}} \leftarrow \text{pop the first tuple from } \mathcal{I} \text{ and replace with MILP leaf} ; \)
18. \( h \leftarrow h + 1 ; \)
19. end
20. end
21. \( h_{\text{min}} \leftarrow \min(h_{\text{min}}, h) ; \)
22. end
23. return \( h_{\text{min}} \)
end

Table 11: Convergence statistics for the standard (natural) GBDT models between our solution and the optimum MILP solution. We collect the data after the fine-grained binary search but before applying LT-Attack (Initial), and the data after LT-Attack (Converged). We disabled the random noise optimization discussed in §3.5.

| Dataset    | Model | HammingDist \( h^* \) | NeighborDist \( h^* \) |
|------------|-------|------------------------|------------------------|
|            |       | Initial | Converged | Initial | Converged |
| # of trees | max   | median  | max   | median  | max   | median  |
|------------|-------|---------|-------|---------|-------|---------|
| breast-cancer | 4     | 2       | 0     | 0       | 2     | 0       |
| diabetes   | 20    | 10      | 3     | 6       | 0     | 5       |
| MNIST2-6   | 1000  | 676     | 490   | 347     | 172   | 46      |
| ijcnn      | 60    | 27      | 10    | 16      | 3     | 18      |
| MNIST      | 400   | 266     | 115   | 96      | 33    | 173     |
| F-MNIST    | 400   | 237     | 174   | 100     | 56    | 82      |
| webspam    | 100   | 88      | 56    | 36      | 16    | 75      |
| covtype    | 160   | 84      | 23    | 67      | 9     | 82      |
| HIGGS      | 300   | 190     | 62    | 125     | 23    | 181     |

18