arXiv:2001.11905v1 [cs.LG] 31 Jan 2020

Additive Tree Ensembles: Reasoning About Potential Instances

Laurens Devos Wannes Meert Jesse Davis

Department of Computer Science,
KU Leuven
{laurens.devos, wannes.meert, jesse.davis}@cs.kuleuven.be

Abstract

Imagine being able to ask questions to a black box model such as "Which adversarial examples exist?", "Does a specific attribute have a disproportionate effect on the model's prediction?" or "What kind of predictions are possible for a partially described example?" This last question is particularly important if your partial description does not correspond to any observed example in your data, as it provides insight into how the model will extrapolate to unseen data. These capabilities would be extremely helpful as it would allow a user to better understand the model's behavior, particularly as it relates to issues such as robustness, fairness, and bias. In this paper, we propose such an approach for an ensemble of trees. Since, in general, this task is intractable we present a strategy that (1) can prune part of the input space given the question asked to simplify the problem; and (2) follows a divide and conquer approach that is incremental and can always return some answers and indicates which parts of the input domains are still uncertain. The usefulness of our approach is shown on a diverse set of use cases.

1 Introduction

As machine learning sees wider and wider adoption, it is increasingly being deployed in more sensitive areas. Therefore, it is crucial that we understand how machine learned models reach a decision, even in circumstances that we have not encountered before. To better understand the model's behavior, having the ability to ask questions relating to issues such as robustness, fairness, and bias is crucial. These questions are particularly useful when it comes to reasoning about unseen situations, i.e., *potential instances* that do not necessarily appear in the training or test datasets. For example, given a black box model, it would be really useful to be able to get answers to the following questions:

- Given a correctly classified instance from the dataset, can slightly perturbing it cause its predicted label to change? Can we carefully choose the perturbation such that the model's predicted label corresponds to a specific class? These sorts of instances are often called *adversarial examples*.

- Can a particular attribute have a disproportionate or unwanted effect on the model's prediction? Can we find all attributes that have a disproportionate effect on the output of the model? These effects are often called an *unintended bias*.

- Given a partially described instance, can we find values for the unknown attributes such that a certain label is predicted? These instances can be linked to regions of the instance space where the model *extrapolates* because there are no training instances.

In this work, we focus on developing an approach that can answer such questions about *additive tree ensembles*, which includes random forests [3] and gradient boosting trees (e.g., [4, 15, 7]). This represents a powerful and widely-used family of machine learning algorithms. There has been prior work on answering questions about such models. However, this work has focused on specific types of questions such as finding adversarial examples [9], evasion [13], providing explanations [12], or testing robustness and stability [21, 17]. A standard approach is to encode the tree(s) and question into a logical theory, and then perform theorem proving. As this process is NP-complete, approaches exploit the fact that they target a specific question to design efficient procedures in practice.

This paper makes three contributions.[1] First, we focus on being able to provide answers to a broader class of questions about an additive tree ensemble. Namely, we consider any question that can be represented as a satisfiable modulo theory (SMT) formula. Second, as our approach relies on theorem proving, which is intractable in general, we develop two strategies for speeding up the process. Specifically, our approach (1) prunes large parts of the model's input space given the question, and (2) follows an incremental, divide-and-conquer strategy that can always return some answers, and indicate which parts of the input domains are still uncertain. The divide-and-conquer strategy naturally decomposes the problem into disjoint subproblems, and allows us to find multiple distinct satisfying instances. Additionally, this information can be used as feedback to the user to potentially refine the question or update the background knowledge to restrict the input space, making it an iterative procedure with a human in the loop. Third, we provide a diverse set of use cases that highlights our approach's ability to answer questions falling into each of the three aforementioned categories.

2 Preliminaries

The framework described in this paper reasons about **additive ensembles of binary trees**.[2] A binary tree T consists of nodes n_i and has a special first node n_0 called the root node. There are two types of nodes. An *internal node* n stores a split condition defined on an input attribute and references to two child nodes $\mathrm{left}(n)$ and $\mathrm{right}(n)$. The split condition is either a less-than split $X_k < \tau$ defined on the real variable $X_k \in \mathcal{V}$ corresponding to a real attribute A_k or a Boolean split defined on a variable

[1] Code is available at https://github.com/laudv/treeck

[2] Note that all trees can be represented as binary trees.

corresponding to a Boolean attribute. A *leaf node* is a node without children that stores an output value.

A tree is evaluated by recursively traversing it starting from the root node. For internal nodes, the node's split condition is tested; if the test succeeds, the procedure is recursively applied to the left child node, else, it is applied to the right child node. If a leaf node is encountered, the value stored in the leaf is returned and the procedure terminates.

An additive ensemble of trees is a sum of trees $\boldsymbol{T} = T_1 + \cdots + T_M$ and is evaluated by summing the evaluations of all trees. Examples of additive tree models are random forests with voting for classification or output averaging for regression, and gradient boosting trees. Both of these are powerful methods frequently used in practice.

We will be using **SMT** solvers (satisfiability modulo theories) to check logical theories. SMT extends the boolean satisfiability problem (often abbreviated SAT) with a number of additional concepts. For this work, the most important capability is the addition of real variables and linear constraints between them. As SMT is more expressive than SAT, many SMT problems are of course intractable. However, powerful solvers like Z3 [5] have been used to solve many real-world problems.

3 Problem Setting and Approach

The problem setting we consider is the following:

Given: An additive tree ensemble model, a question that can be represented as an SMT formula, and any available domain knowledge.

Do: Check if a (set of) instances exists that satisfies the requirements in the question.

Our approach is translate the given information into a logical formula. An SMT solver can then be applied to check if a satisfying assignment to the formula exists. The SMT solver will either return *yes*, and provide a concrete instance that answers the question, or *no*, which means that no instance exists that satisfies the provided information. Next we discuss encoding the ensemble, the question, and the optional background knowledge into a logical theorem as well as the computational complexity of this problem.

3.1 Encoding the Ensemble

Consider an additive tree ensemble trained on input data with K attributes A_k. We first need to define the decision variables that correspond to the different values used and stored in the trees:

- K real or boolean variables X_k corresponding to the K input attributes A_k,
- M real tree output values W_m corresponding to the outputs (or leafs) of the individual trees, and
- one real ensemble output variable F.

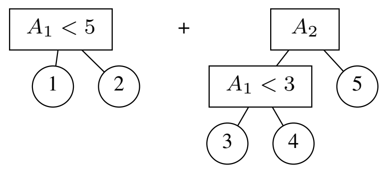

Figure 1: An additive tree model

The decision variables are collected in a set $\mathcal{V}$.

We define an encoding procedure enc that translates a tree into a logical theorem using the variables in $\mathcal{V}$. Each tree T_m is encoded by starting at its root node n_{m0} and recursively encoding all its descendants. A node's encoding depends on its type. A leaf node n is encoded as:

$$\mathrm{enc}(n, \mathcal{V}) \to (W_m = \mathrm{value}(n)), \tag{1}$$

where $\mathrm{value}(n)$ is the output value stored in the leaf node. An internal node n is encoded as:

$$\begin{aligned} \mathrm{enc}(n, \mathcal{V}) \to \quad & (\quad \mathrm{cond}(n) \quad \wedge \quad \mathrm{enc}(\mathrm{left}(n)) \quad) \\ \vee \; & (\quad \neg\mathrm{cond}(n) \quad \wedge \quad \mathrm{enc}(\mathrm{right}(n)) \quad), \end{aligned} \tag{2}$$

where $\mathrm{left}(n)$ and $\mathrm{right}(n)$ are the left and right child nodes of the internal node, and $\mathrm{cond}(n)$ refers to the node's split condition.

An additive ensemble $\boldsymbol{T} = \sum_m T_m$ is encoded as the conjunction of the encodings of the individual trees and an output constraint:

$$\mathrm{enc}(\boldsymbol{T}, \mathcal{V}) \to \left(\bigwedge_m \mathrm{enc}(T_m, \mathcal{V}) \right) \wedge \left(F = \sum_m W_m \right), \tag{3}$$

with W_m the tree output variables in $\mathcal{V}$, $m = 1, \ldots, M$.

The encoding of the ensemble in Figure 1 is:

$$\begin{aligned} & [(X_1 < 5 \wedge W_1 = 1) \vee (X_1 \geq 5 \wedge W_1 = 2)] \\ \wedge\, & (X_2 \wedge [(X_1 < 3 \wedge W_2 = 3) \vee (X_1 \geq 3 \wedge W_2 = 4)]) \\ \wedge\, & (\neg X_2 \wedge W_2 = 5) \;\wedge\; F = W_1 + W_2. \end{aligned}$$

3.2 Encoding the Question

The encoding $\mathrm{enc}(\boldsymbol{T})$ of an additive tree ensemble $\boldsymbol{T}$ is only useful when combined with the encoding of a question. Our approach works with any question that can be represented as a formula in the SMT-Lib [1] language using (1) (a subset of) the variables in $\mathcal{V}$, and possibly (2) extra decision variables to formulate constraints. We will refer to the encoding of a question by $\mathrm{question}(\mathcal{V}, S)$, with S the set of additional decision variables.

A simple question about the ensemble in Figure 1 is

Example 1. *Does an instance x exists for which attribute 1 has a value less than 2 and the output of the model is greater than 5?*

This can be encoded as the following SMT formula:

$$\mathrm{enc}(\boldsymbol{T}, \mathcal{V}) \ \wedge \ (X_1 < 2) \ \wedge \ F > 5.$$

However, SMT also gives the flexibility to ask for more complicated questions, including ones that require reasoning about sets of instances. A question of this type about the ensemble in Figure 1 is

Example 2. *Find two instances x and x^* that differ in only one attribute such that the model's prediction for x is 2 units greater than its prediction for x^*.*

Representing this question as an SMT formula requires introducing *new decision variables*. This question can be encoded as:

$$\begin{aligned}
&\mathrm{enc}(\boldsymbol{T}, \mathcal{V}) \ \wedge \ \mathrm{enc}(\boldsymbol{T}, \mathcal{V}') \\
&\quad \wedge [(\neg S_1 \wedge (X_1 = X_1')) \vee (S_1 \wedge (X_1 > X_1')] \\
&\quad \wedge [(\neg S_2 \wedge (X_2 = X_2')) \vee (S_2 \wedge (X_2 \wedge \neg X_2')] \\
&\quad \wedge [(S_1 \wedge \neg S_2) \vee (\neg S_1 \wedge S_2)] \ \wedge \ F = F' + 2.
\end{aligned}$$

Here, a positive S_1 or S_2 variable indicates that the attribute takes on a different value in each instance. When reasoning about multiple instances, we add multiple encodings of the ensemble to the theorem using different variable sets $\mathcal{V}$. Depending on the question, the variables in the different instances might be related. These relations must be encoded by constraints in $\mathrm{question}(\{\mathcal{V}\}, S)$, with $\{\mathcal{V}\}$ the set of all variable sets. There is no requirement for the ensemble to be the same for all instances. For example, it is possible to compare multiple 1-versus-all classifiers originating from a multi-class classifier.

3.3 Background Knowledge

The background knowledge BK represents the implicit rules and constraints present in the dataset. Two sorts of background knowledge can be distinguished: A first sort is knowledge about the problem domain. For example, in soccer, a cross must end in the opponent's penalty box. A second is feature engineering. For example, one-hot encoding usually requires an *exactly-one* constraint, i.e., exactly one option is true. When investigating the effects of extrapolation, one should avoid suggestions that violate the background knowledge, not only because these suggestions are not informative, but also because they might distort your results.

3.4 Complexity Analysis

Our setting means that answering a question about an ensemble boils down to checking if $\mathrm{enc}(\boldsymbol{T}, \mathcal{V}) \wedge \mathrm{question}(\mathcal{V}, S) \wedge BK$ is satisfiable. Without considering the additional and arbitrarily complex constraints in $\mathrm{question}(\mathcal{V}, S)$, verifying whether $F > 0$ for a given additive ensemble is already NP-complete as it can be reduced from 3-SAT [13].

4 Algorithm: Prune, Divide, and Conquer

There are two main requirements:

- *Requirement 1*: The algorithm must be able to handle general questions. We cannot rely on question-specific optimizations used in prior work.
- *Requirement 2*: The algorithm must provide insights even if the question is intractable. The insights should inform the user about potential useful refinements to the question or updates to the background knowledge.

Intuitively, the complexity of the problem can be seen as exponential in the number of leafs. We can understand this as follows: the ensemble output F is defined as the sum of all tree outputs. The tree output is the value in the reached leaf node. The number of possible leaf-value combinations making up the sum F is enormous. For example, for a small ensemble consisting of 10 trees each with 64 leafs would have $64^{10} \approx 10^{18}$ potential leaf-value combinations. This suggest we should somehow limit the number of leafs per tree. Guided by this intuitive insight, we designed two strategies that reduce the number of leafs without changing the prediction made by the ensemble.

4.1 Pruning Unreachable Branches

The idea underlying this strategy is simple: the constraints in a question can make certain branches in the trees inaccessible. For example, assume a question is about "*men aged 32 or older*". If you reach a node splitting on $\mathrm{Age} < 23$, its left branch is never followed, because it contradicts the age condition in the question.

We can formulate this more formally as follows. Consider the path from a node n in a tree T to the root node n_0. When moving from a child node to its parent node p, either a left or a right branch is followed. In the case of a left branch, the true condition $\mathrm{cond}(p)$ holds on the path from n to n_0. If a right branch is taken, the negated condition $\neg\mathrm{cond}(p)$ holds.

Let $\mathrm{path}(n, T)$ be the conjunction of the (negated) conditions on the path from node n to the root node of tree T. Given $\mathrm{question}(\mathcal{V}, S)$, if a tree branch rooted at n is impossible given that question, that is, $\mathrm{question}(\mathcal{V}, S) \wedge \mathrm{path}(n, T) \wedge BK$ does not have a solution, we can prune it from the model. Excluding the entire branch from the tree encoding $\mathrm{enc}(T, \mathcal{V})$ reduces the number of encoded leafs by the number of leafs below n. As long as $\mathrm{question}(\mathcal{V}, S)$ is reasonably simple, this subproblem is much easier to solve than the full problem.

4.2 Divide and Conquer: Split the Input Domain

This strategy splits the input domain into two sub-domains such that the number of still reachable leafs is maximally decreased. Consider Figure 1. Initially, the domains of the input attributes A_1 and A_2 are $\mathbb{R}$ and $\{\mathit{True}, \mathit{False}\}$. If we split the domain of A_1 into $(-\infty, 5)$ and $[5, \infty)$, then node ② is unreachable in the first sub-domain, and nodes ① and ③ are unreachable in the second sub-domain.

The procedure loops over all splits in the additive ensemble and counts the number of unreachable leafs in the first and second sub-domains. It proceeds by splitting the input domain using the split with the highest unreachable leaf count. This produces two subproblems of reduced size. Any satisfying solution of a subproblem is a satisfying solution of the original problem. If all subproblems are unsatisfiable, then the original problem is unsatisfiable.

4.3 Combining Strategies: Prune, Divide and Conquer

The two strategies above can be applied iteratively:

1. Prune the tree given the *divide constraint* (initially unconstrained, i.e., *True*) and the question.

2. Apply the SMT solver on the pruned encoding of the trees and the question. If an answer is obtained within the timeout, report the answer; else stop the solver and continue to Step 3.

3. Divide the input domain into two sub-domains using the best split C from the ensemble (e.g., $A_k < \tau$) and start two new problem instances at Step 1. The *divide constraint* is the current divide constraint appended by C and $\neg C$ respectively.

A major benefit from this approach is that the two subproblems produced by Step 3 can be solved independently in parallel, with only minor data moving and synchronization requirements. Our implementation can run the subproblems on a cluster of machines.

We can stop the algorithm at any point, even if we do not have an answer to our question. There are two ways in which we can interpret the partial results of the algorithm: (1) The solver may have returned a *yes* or *no* answer to some subproblems. Depending the requirements, a single *yes* answer with a generated instance might be enough. The *no* answers indicate that the question is impossible to answer in some sub-domains of the solution space. For example, when testing robustness, a *no* answer indicates that robustness holds in the sub-domain. (2) The subproblems that have not been solved yet might indicate that finding a solution in this subspace is difficult. The more difficult the subspace, the more domain splits will have been generated. This may be informative to a domain expert: are some sub-domains (physically) impossible or uninteresting? Can we add information to the background knowledge to avoid this area? Can we reformulate the question to bypass this issue?

5 Use Cases

All models are constructed by XGBoost [4]. We used XGBoost's early stopping functionality with a maximum number of trees of 50. The early stopping functionality stops the ensemble construction when no progress is made on the test set in the last 5 iterations.

5.1 Verifying Monotonicity

A model is monotone in an attribute A_{k^*} if an increase in the attribute's value results in an increase in the model's output. We can verify monotonicity as follows. Let X_k and X'_k be attribute variables and F and F' be ensemble outputs for a first and a second instance. Add the following constraints to the question encoding: $X_k = X'_k$ for all $k \neq k^*$, $X'_{k^*} < X'_{k^*}$, and $F > F'$. We can pass this question to our system.

If our system responds with a *yes*, then we have two instances for which the monotonicity constraint is violated. If the system reports *no*, then the monotonicity constraint holds.

We reproduced the simple synthetic dataset presented in the XGBoost documentation about the monotonicity feature.[3] We were able to verify that the models constructed using this dataset were indeed monotone.

5.2 Generating Adversarial Examples

This use case focuses on generating *adversarial examples*, which corresponds to the following question:

> Given a correctly classified instance from the dataset, can a minor perturbation change the predicted output label to a desired value?

This problem has been well studied for additive tree ensembles [13, 9] as well as for other algorithms (e.g. neural networks [20, 11], SVMs [2]). However, existing approaches have formulated solution strategies specifically for this question. We illustrate that our generic approach can solve the same question.

We used an XGBoost model trained on the MNIST dataset [16] that has 45 trees and results in a test set error rate of 2.76%. For a given training data $I = I_1, \ldots, I_K$, we check whether an adversarial example exists that satisfies the following three constraints: (1) each individual pixel can only deviate by at most δ from the original pixel: $|X_k - I_k| < \delta$, (2) the sum of all absolute changes can be at most Δ: $\sum_k |X_k - I_k| < \Delta$, and (3) the model must have a low confidence in the original label, and a high confidence in the desired false label for the perturbed instance. Figure 2 shows a representative digit from the training set (top row) and generated examples with $\delta = 75$ and a total budget $\Delta = 3000$ (bottom row). Apart from the generated eight, the model is 99% certain all digits in the bottom row should be classified as a nine.

We can use our system to analyze an *average robustness* of the model for each digit. To test this, we randomly selected an instance and a desired incorrect target label. Then we check if an adversarial example exists given the two constraints on perturbing the pixels. We repeated this 2000 times. Figure 3 shows two quantities for each digit. The left side shows how difficult it is to change the label of each digit. The right side shows how difficult it is to perturb an instance such that the model predicts a specific target digit. We can see that it is hard to change an *eight* into another digit, but it is easy to change any other digit into an *eight*. The opposite is true for the *one* digit, which is easy to change into any other digit, but it is hard to turn another digit into a *one*. The

[3] xgboost.readthedocs.io/en/latest/tutorials/monotonic.html

Figure 2: Original and adversarial examples for MNIST. The top row shows the original instances which are correctly classified with high confidence. The bottom row shows the generated perturbed instances. For each of the original instances for zero through seven, it is possible to generate an adversarial example that is incorrectly classified as the number nine with high confidence. However, for the last digit, no adversarial exists, i.e., there is no perturbation that satisfies the given constraints such that the generated instance is classified as a nine.

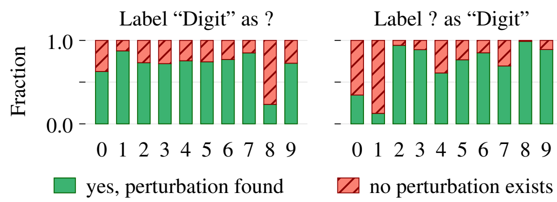

Figure 3: The robustness of each digit averaged over 2000 randomly sampled MNIST digits. The left column shows how frequently the classifier could be *fooled* into labeling the instance with label on the x-axis as another preselected digit. The right column shows the inverse: how frequently could the label of an instance be turned into the label on the x-axis.

solver always terminates in this experiment and our approach takes between 10 and 45 seconds on average to answer the question.

5.3 Challenging Fairness

Although our system cannot reason about *individual fairness* as it was original defined [8], we can generate individuals that are treated unfairly. We use the Adult[4] dataset to illustrate this idea. This task is to predict whether an individual has a salary greater than 50k using information like age, education, race, sex, etc. We use an XGBoost model with 30 trees that achieves a test set accuracy of 86.7%. We ask the following question:

> Can we find two individuals A and B that only differ on one protected attribute where the model makes a different prediction?

We use sex as the protected attribute and search for pairs of examples where the model's predicted probability of earning less than 50k is $\geq p$ for individual A and $\leq 1 - p$ for individual B. Figure 4 shows how varying p affects the results. It is relatively easy to generate pairs instances that are treated unfairly when $p \leq 0.8$. For $p > 0.85$, no such pairs exists. However, the solver does not terminate within 20 minutes when p is between 0.8 and 0.85. This is referred to as the *phase transition* of satisfiability problems [10].

[4]https://archive.ics.uci.edu/ml/datasets/adult

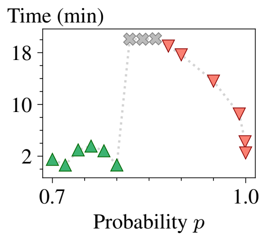

Figure 4: A SAT phase transition for generating pairs of instances that differ only in a protected attribute, yet receive an opposite label. The confidence of the prediction p is varied. The upwards and downwards triangles refer to *yes* and *no* responses. Crosses refer to timeouts.

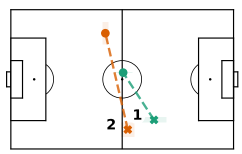

Figure 5: Two midfield *pass-dribble* sequences that have a probability greater than 10%. The pass starts at the cross and ends at the circle. The dribble starts at the circle.

5.4 Detecting Dominant Attributes

This use case attempts to predict the order of magnitude of YouTube[5] video view counts (log 10) using a binary bag-of-words representation of the words appearing in the video's title and description. The XGBoost ensemble has 50 trees and achieves a test set mean absolute error of 0.38. We address the following question:

> Is it possible to find two instances, i.e., sets of words, that differ only in a single word such that the single word difference results in a two order of magnitude difference in the predicted view count?

This questions contrasts with the previous two use cases as we neither *investigate a particular instance (MNIST)* nor *focus on one specific protected attribute (Adult)*. The task is to find *any* attribute that, when flipped, causes a significant change in the predicted value.

This produces some interesting results. We constrained the result to have less than 12 words. A bag-of-words representation does not enforce a word order, so we reordered the word to make a sentence. We also added filler words in non-bold. These words do not appear in the bag-of-words representation and therefore have no effect on the predicted view count.

The first example is constrained to contain the words *night*, *talk*, and *show*. This is one of the results:

> **Night talk show video**: **pop drama** about the **latest hot Christmas house album** (**remix**).

Without the word *remix*, this title is predicted to receive 200,000 views. However, if *remix* is included, the predicted view count rises to 30 million. A second example of a video title containing the words: *news*, *breaking*, and *channel*:

> **Breaking news** from the **money channel**: **no weird vlogs today challenge** (**full movie**).

[5] https://www.kaggle.com/datasnaek/youtube-new

Without the word *no*, this video is predicted to receive 100,000 views. With *no*, it is predicted to receive 100 million views. A final example is:

> The **12 avengers challenge Paul**, the **<u>Christmas</u> pop fashion king** in **DE**.

When the word *Christmas* is omitted, the video is predicted to get 1 million views. However, if *Christmas* is included, the prediction drops to only 1000 views.

While the stakes are not high for view count prediction, similar situations can arise in prediction tasks of insurance companies, law enforcement, the health care sector, etc. The robustness of the above boosted tree ensemble is clearly inadequate for such much more sensitive applications.

5.5 Querying the Model

Our final use case involves analyzing real-world event stream from professional soccer matches. The machine learning task is to estimate the probability of scoring a goal in the near future (e.g., within the next ten actions) from a particular game state.[6] This enables valuing on-the-ball actions, which is a crucial task for soccer analytics [6]. We trained an XGBoost model with 50 trees using 1.1 million actions over multiple games. The attributes are: action type (e.g. pass, shot, throw in, penalty, etc.), x and y coordinate of action, body part used (foot, head, other), current goal scores, and time remaining. We consider the following question:

> Can we find a two action sequence involving a backward pass in the middle of the field that results in a game state with a probability greater than 10% of scoring in the near future?[7]

This is a relevant question as the soccer analytics community is interested in understanding the usefulness of backwards passes far away from the goal. Our method proved that *pass-pass* sequences in the midfield cannot have a probability greater than 10%. When also allowing dribbles, the system generated several *pass-dribble* sequences, two of which are shown in Figure 5. Intuitively, these sequences could represent valuable situations because the backward pass could simultaneously switch the direction of play and get the ball to a player who has space where he can advance the ball.

6 Algorithm Analysis

To evaluate effectiveness of our prune plus divide and conquer algorithm, we compare three variants: (1) the full encoding is passed directly to the SMT solver, (2) applying the pruning step before passing the encoding to the SMT solver, and (3) applying both the pruning and divide-and-conquer steps. We consider two question types. On YouTube, we randomly picked eight fixed words and asked the same question as in the use case. On MNIST, we randomly picked an instance and an incorrect adversarial target label.

[6]https://github.com/ML-KULeuven/socceraction

[7]Goals are exceedingly rare in soccer, and very few game states would have such a high probability.

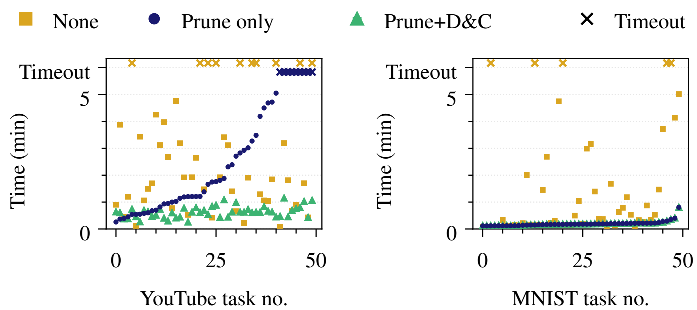

Figure 6: Timings of 50 YouTube tasks on the left, and 50 MNIST tasks on the right. Timings are shown for the following three cases: passing the full encoding directly to the SMT solver ("None"), using only the pruning step ("Prune only"), and using the pruning and divide-and-conquer algorithm ("Prune+D&C").

Figure 6 shows timings on 50 random task for each data set. For the YouTube tasks on the left, all timeouts are eliminated when the full algorithm is active. For the MNIST tasks on the right, the pruning step is the most effective. This makes sense: the adversarial example question is heavily constrained; each pixel can only deviate by δ, which makes many branches in the ensemble's trees unreachable. The number of leafs before pruning is about 5000 and the pruning eliminates 80% of them on average. The SMT solver can solve this reduced problem easily, so no divide-and-conquer is necessary.

7 Related Work

For additive tree ensembles, most related work has focused on finding [9] and evading [13] adversarial examples. The former also uses an SMT solver, while the latter uses mixed-integer linear programming. In contrast to the work in these papers, we have presented a method to perform targeted attacks: we do not just change the label, we change it to a specific value. More generally, we can handle a wider range of questions that do not necessarily reason about one specific instance from the dataset, a setting inherent to the *adversarial question*.

Other work has moved beyond individual adversarial examples and proposed methods to prove *stability* and *robustness* of additive tree ensembles. Ranzato and Zanella [17] propose a method that uses a similar prune and divide-and-conquer approach, but they do not use an SMT solver but an approach specifically tuned for the *stability* problem. Törnblom and Nadjm-Tehrani [21] use a technique they call *equivalence class partitioning* that enumerates all possible outputs of the model. This approach does not scale, however, for problems with more attributes like MNIST.

Ignatiev et al. [12] have used an SMT encoding of boosted trees in a method to provide global explanations and validate heuristic explanations. Their focus is, therefore, also limited to a single question.

Systems answering more general verification questions have been studied for other models. Shih et al. have used knowledge compilation to verify the behavior of Bayesian network classifiers [18] and binarized neural networks [19]. Katz et al. [14] have used SMT solvers for binarized neural networks. These systems can answer general questions like: "How many binary pixels do we have to flip before the label changes?", and "Can we get an output greater than τ given some constraints on the input features?" Our system can handle these questions, though it might require decomposing them into multiple subquestions.

8 Conclusion

We presented an approach that answers general questions about additive tree ensembles. Our approach applies theorem proving using an SMT solver, which is intractable in general. We propose a prune, divide and conquer algorithm that (1) speeds up the computation, and (2) provides partial results when the full computation takes too long. To illustrate the abilities of our approach, we provide a diverse set of use cases and experiments.

References

[1] C. Barrett, P. Fontaine, and C. Tinelli. The satisfiability modulo theories library (smt-lib). `www.SMT-LIB.org`, 2016.

[2] B. Biggio, I. Corona, B. Nelson, B. Rubinstein, D. Maiorca, G. Fumera, G. Giacinto, and F. Roli. *Security Evaluation of Support Vector Machines in Adversarial Environments*, pages 105–153. Springer International Publishing, 2014.

[3] L. Breiman. Random forests. *Machine learning*, 45(1):5–32, 2001.

[4] T. Chen and C. Guestrin. XGBoost: A scalable tree boosting system. In *Proceedings of the 22nd acm sigkdd international conference on knowledge discovery and data mining*, pages 785–794. ACM, 2016.

[5] L. De Moura and N. Bjørner. Z3: An efficient smt solver. In *International conference on Tools and Algorithms for the Construction and Analysis of Systems*, pages 337–340. Springer, 2008.

[6] T. Decroos, L. Bransen, J. Van Haaren, and J. Davis. Actions speak louder than goals: Valuing player actions in soccer. In *Proceedings of the 25th ACM SIGKDD International Conference on Knowledge Discovery & Data Mining*, pages 1851–1861, 2019.

[7] L. Devos, W. Meert, and J. Davis. Fast gradient boosting decision trees with bit-level data structures. In *Proceedings of ECML PKDD*. Springer, 2019.

[8] C. Dwork, M. Hardt, T. Pitassi, O. Reingold, and R. Zemel. Fairness through awareness. In *Proceedings of the 3rd innovations in theoretical computer science conference*, pages 214–226. ACM, 2012.

[9] G. Einziger, M. Goldstein, Y. Sa'ar, and I. Segall. Verifying robustness of gradient boosted models. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 33, pages 2446–2453, 2019.

[10] I. P. Gent and T. Walsh. The sat phase transition. In *ECAI*, volume 94, pages 105–109. PITMAN, 1994.

[11] I. J. Goodfellow, J. Shlens, and C. Szegedy. Explaining and harnessing adversarial examples, 2014.

[12] A. Ignatiev, N. Narodytska, and J. Marques-Silva. On validating, repairing and refining heuristic ml explanations, 2019.

[13] A. Kantchelian, J. D. Tygar, and A. Joseph. Evasion and hardening of tree ensemble classifiers. In *International Conference on Machine Learning*, pages 2387–2396, 2016.

[14] G. Katz, C. Barrett, D. L. Dill, K. Julian, and M. J. Kochenderfer. Reluplex: An efficient smt solver for verifying deep neural networks. In *International Conference on Computer Aided Verification*, pages 97–117. Springer, 2017.

[15] G. Ke, Q. Meng, T. Finley, T. Wang, W. Chen, W. Ma, Q. Ye, and T.-Y. Liu. LightGBM: A highly efficient gradient boosting decision tree. In *Advances in Neural Information Processing Systems*, pages 3146–3154, 2017.

[16] Y. LeCun. The mnist database of handwritten digits, 1998.

[17] F. Ranzato and M. Zanella. Formal stability verification of forest classifiers. unpublished, 2019.

[18] A. Shih, A. Choi, and A. Darwiche. Formal verification of bayesian network classifiers. In *International Conference on Probabilistic Graphical Models*, pages 427–438, 2018.

[19] A. Shih, A. Darwiche, and A. Choi. Verifying binarized neural networks by local automaton learning. In *AAAI Spring Symposium on Verification of Neural Networks (VNN)*, 2019.

[20] C. Szegedy, W. Zaremba, I. Sutskever, J. Bruna, D. Erhan, I. Goodfellow, and R. Fergus. Intriguing properties of neural networks, 2013.

[21] J. Törnblom and S. Nadjm-Tehrani. Formal verification of random forests in safety-critical applications. In *International Workshop on Formal Techniques for Safety-Critical Systems*, pages 55–71. Springer, 2018.