Extracting Optimal Explanations for Ensemble Trees via Automated Reasoning

Gelin Zhang[1], Zhé Hóu[2], Yanhong Huang[1*], Jianqi Shi[1], Hadrien Bride[2], Jin Song Dong[2,3] and Yongsheng Gao[2]

[1*]National Trusted Embedded Software Engineering Technology Research Center, East China Normal University, Shanghai, China.
[2]Griffith University, Brisbane, Australia.
[3]National University of Singapore, Singapore.

*Corresponding author(s). E-mail(s): yhhuang@sei.ecnu.edu.cn;

Abstract

Ensemble trees are a popular machine learning model which often yields high prediction performance when analysing structured data. Although individual small decision trees are deemed explainable by nature, an ensemble of large trees is often difficult to understand. In this work, we propose an approach called optimised explanation (OptExplain) that faithfully extracts global explanations of ensemble trees using a combination of logical reasoning, sampling and nature-inspired optimisation. OptExplain is an interpretable surrogate model that is as close as possible to the prediction ability of the original model. Building on top of this, we propose a method called the profile of equivalent classes (ProClass), which simplify the explanation even further by solving the maximum satisfiability problem (MAX-SAT). ProClass gives the profile of the classes and features from the perspective of the model[*]. Experiment on several datasets shows that our approach can provide high-quality explanations to large ensemble trees models, and it betters recent top-performers.

Keywords: Explainable Artificial Intelligence (XAI), Random Forest, Classification, Decision Rule Extraction

[*]The code is available at https://github.com/GreeenZhang/OptExplain.

1 Introduction

Background.

Ensemble trees are a family of machine learning techniques that combine individual decision trees to form a better prediction model. Examples include random forest [1, 2], which combines strong learners (e.g., large trees) to reduce variance and avoid overfitting. Boosting [3, 4], on the other hand, combines weak learners (e.g., small trees) to reduce bias. Ensemble trees are very successful in today's data analytics competitions and applications; they are especially suited to analyse *structured data* such as databases and spreadsheets, where they sometimes outperform deep learning [5].

Although decision trees are often deemed an explainable, or even a "white-box" model, such an impression usually refers to a single, short decision tree. In the context of ensemble trees such as the models generated by random forest or boosting, there can be a large number of trees and each tree can be gigantic. For example, to achieve a 0.76+ area-under-the-curve (AUC) for the 1 million flight dataset [5], Silas [6, 7] trains a model of 100 trees, and each tree has more than 32,000 branches. Such a model certainly does not manifest itself in an explainable manner to the general user. The main goal of this work is to extract faithful explanations for such large-scale models.

There are several existing methods for analysing and interpreting machine learning models. For example, the LIME tool [8] and the SHAP values [9] are both promising techniques for solving this problem. We will discuss more details of related work in Section 2. However, most existing work is done from a statistics perspective. Such methods use a prediction model as a black-box and attempt to find statistical (e.g., linear) approximations of the model. By contrast, our philosophy is that we should analyse the internal working of the model and obtain an understanding of how it works logically. Further, many existing techniques are focused on *local explanations*, that is, how the model predicts for a particular data instance. This work is primarily about *global explanations*, which explains how the model behaves generally.

Our approach.

In our previous work [7], we have used sampling and maximum satisfiable subset to extract the decision logic of the model. However, it is non-trivial to manually adjust the sampling parameters, which may lead to vastly different explanations. Default parameters often lead to very simple explanations that diverge from the original model. In this paper, we propose an integrated and automated framework for providing global explanations. Moreover, our goal is not just to give *an* explanation as is done in the literature, but to give *the optimal* explanation in terms of *simplicity* and *faithfulness*.

The utilities of this work are manifold. First, our approach can provide human-understandable explanations that are very close to the original ensemble trees model in predictive behaviour. Second, such an explanation can also be used as an approximation of the original model that can be used for other

purposes such as verification and testing. Finally, this work can serve as a stepping stone towards explaining deep learning by combining existing work that transforms neural networks to decision trees [10].

Contributions.

The main contributions of this paper are as follows:

1. We formalise ensemble trees into logical formulae and develop simplification and abstraction algorithms that are specialised for machine learning.
2. We propose an automated explanation extraction method called *OptExplain*, which combines logical reasoning, sampling, and bio-inspired optimisation.
3. We also develop a method called *ProClass* that computes the abstractions of each (equivalent) class using MAX-SAT.
4. Through case studies and experiment, we show that our method is practical and useful on different datasets. It also outperforms similar tools.

The remainder of this paper is organised as follows: Section 2 discusses related work, Section 3 details the proposed approach, Section 4 gives case studies and experiment, and Section 5 concludes the paper.

2 Related Work

There are a variety of approaches for tackling machine learning interpretability. Some recent and popular ones are focused on *local explanations*, that is, how the model predicts for a particular data instance. LIME [8] is such a tool that finds linear approximations of the prediction model and gives importance weights for certain predicates used in the prediction. Anchors [11] generates "if-then" style explanations for predictions. Such explanations have similar forms as our decision rules and are considered intuitive and easy to understand by the user. Shapley (SHAP) Values [9] are often used to extract importance scores and impacts on features. Like LIME, SHAP also provides user-friendly graphical presentations (e.g., bar charts) for explaining predictions. It should be noted that SHAP can also be used to obtain global feature importance. The above three methods are *model-agnostic*, which means that in the process of providing explanations and making machine learning more "white-box", they take prediction models as a "black-box" and attempt to find patterns of *features* when the model makes predictions. An advantage is that they can be applied to different machine learning techniques, including ensemble trees and neural networks. CHIRPS [12] is another technique for local explanations. In contrast to the above techniques, CHIRPS looks into decision trees and uses frequent pattern mining on decision nodes to obtain decision rules as the explanation.

Global explanations that explains how the model behaves generally are more closely related to this work. Recent examples include Hara and Hayashi's approach [13] that uses Bayesian model selection to extract decision rules.

Deng's inTrees [14] extracts, selects and prunes rules from a set of decision trees and uses frequent pattern analysis to summarise rules into a smaller prediction model.

Most relevant techniques come from a statistics perspective. Their simplification process often consists of selection, pruning and frequency analysis. By contrast, our work comes from a logician's point of view and, on top of the usual selection and pruning operations, uses automated reasoning to simplify logical formulae and find abstractions of equivalent classes. We see many related methods as complementary ones rather than competitors because they output in different forms. For example, one can combine SHAP values and our work to form a more comprehensive explanation.

3 The Proposed Method: OptExplain and ProClass

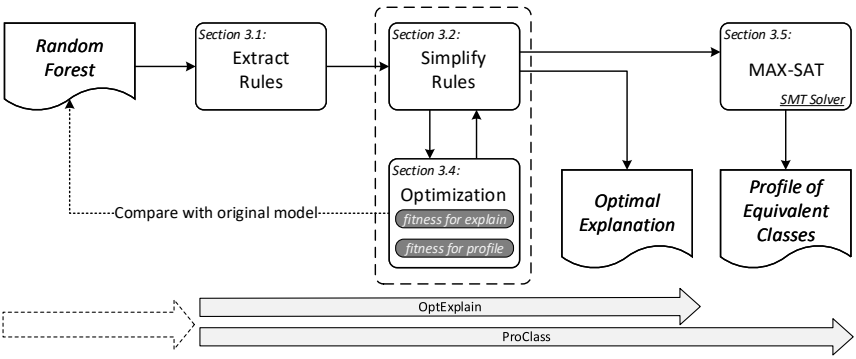

Fig. 1 An overview of the proposed method

An outline of our approach follows: we extract logical formulae from a set of trees where each branch forms a "decision rule". We then reduce the size of the model by filtering out low-quality nodes (i.e., sub-formulae) and branches. We also devise a customised formulae simplification algorithm to obtain logically equivalent smaller models. In case there are still too many decision rules, we group the rules into "equivalent classes" to further abstract the model. The parameters in the above process are optimised using particle swarm towards a sweet spot of simplicity and faithfulness. As an extra step, we can simplify the explanation using MAX-SAT to obtain even more abstract representations of each equivalent class, which we call the "profiles of classes". Such profiles can provide straightforward and even visual explanations of the model.

3.1 From Decision Trees to Decision Rules

Decision Tree.

Let $t(x) = y$, where t is the decision tree [15], x is an input instance, y is the output class. A decision tree is composed of internal nodes and terminal nodes called leaves. Each internal node is associated with a logical formula over a feature. Each leaf node contains a set of instances, which yield a *vote distribution* of the form $(n_1, \cdots, n_m)$ where m is the number of classes and n_i $(1 \leq i \leq m)$ is the number of instances of the corresponding class.

It is straightforward to obtain the formula that is associated with each internal node. From there, we can obtain the "decision logic" of a branch via the *branch formula* of the form $(\bigwedge F) \rightarrow D$, where $\bigwedge F$ is the conjunction of all the (possibly negated) internal node formula on the branch, and D is the vote distribution at the leaf node. That is, if the branch goes through a node F to the right branch, then we include F in the conjunction, otherwise we include $\neg F$. We refer to a formula of the above form as a *decision rule*.

The *tree conversion* algorithm is given in Algorithm 1.

Algorithm 1 The tree conversion algorithm

Input: a decision tree t
Output: a set R_t of branch formulae

$R_t \Leftarrow \{\}$
for branch b in t **do**
  $C \leftarrow \top$
  for node n on b with an associated formula F **do**
    if b goes through the right child of n **then**
      $C \leftarrow C \wedge F$
    else
      $C \leftarrow C \wedge (\neg F)$
    end if
  end for
  $D \leftarrow \bigwedge(c_i = n_i), 1 \leq i \leq m \triangleright c_i$ is the variable for the ith class and n_i is the number of instances of that class in the leaf of b.
  $R_t \leftarrow R_t \cup \{C \rightarrow D\}$
end for

A decision tree t can be converted into a set R_t of mutually exclusive decision rules. R_t can be used in classification tasks by finding the decision rule that satisfies an instance and outputting the class of the largest number of votes.

Ensemble Trees.

We adopt the definitions of Cui et al. [16]. Let an ensemble be a set of decision trees of size T. It gives the weighted sum of the trees as follows:

$E(x) = \sum_{i=1}^{T} w_i \cdot t_i(x)$, where E is the function for the ensemble, w_i and t_i are respectively the weight and function for each tree. The summation aggregates the weighted votes from each tree and obtains the final votes for each class. The above method can be extended to handle an ensemble of trees produced by random forest [2] or boosting [3, 17]. The *ensemble trees conversion* algorithm is given in Algorithm 2. In this work, we evaluate our approach using random forest, but our approach can also be adapted to handle boosting models. In such cases, we need to consider a set E of trees, and we need to multiply the vote distribution D of each tree by its weight in the ensemble. The result, which we refer to as R_E, is the union of the set of decision rules from each tree.

Algorithm 2 The ensemble trees conversion algorithm

Input: a set E of weighted decision trees
Output: a set R_E of decision rules

```
R_E ← {}
for tree t_i ∈ E which has weight w_i do
    obtain R_{t_i} via Algorithm 1
    for vote distribution D in R_{t_i} do
        multiply the vote count of each class by w_i
    end for
    R_E ← R_E ∪ R_{t_i}
end for
```

For an ensemble E of n trees and for any data instance, there should be exactly n decision rules in the set R_E that are satisfied by the instance — one from each tree. To use R_E in classification tasks, one can find the subset of decision rules that are satisfied by an instance x and aggregate the weighted vote counts for each class from those rules. The class with the highest weighted count is the output, which we refer to as $R_E(x)$. The following result is straightforward:

Proposition 1 *For any ensemble trees model E and any data instance x, suppose R_E is the set of decision rules of E derived by the method above, then $E(x) = R_E(x)$.*

In the sequel, we denote the original ensemble trees model as E and the converted set of decision rules as R_E.

3.2 Simplification of Decision Rules

As discussed in Section 1, some ensemble trees models used in real-life applications are enormous and complex. Consequently, the converted set of decision rules for such a model consists of a huge number of rules and each rule may be a very long formula. To reduce the complexity of the explanation, we consider

simplifying the set of decision rules in two dimensions: the length of the rules and the number of the rules.

Continuing from the output R_E of Section 3.1, each formula in R_E is a branch formula, which we can simplify using a node filter as step one.

Parameter 1 (Node Filter θ) We measure the "quality" of a node (NQ) by information gain [18] (IG):

$$NQ(n) = IG(n) \tag{1}$$

where n is the target node. We scan the nodes of each branch in each tree, and remove the nodes with NQ below θ, which is a real positive number.

The second step is to simplify each branch formula by merging the nodes.

Lemma 1 For any branch formula $(\bigwedge F_i) \rightarrow D$, where each F_i is the formula associated with a node on the branch, the left-hand side can be simplified to conjunction normal form (CNF) with at most n conjuncts, where n is the number of features of the dataset.

Proof of Lemma 1 By the construction of decision trees, for any two conjuncts $v \geq l$ and $v \geq l'$ over a numeric feature v that appears in $\bigwedge F_i$, we can simplify them into one conjunct $v \geq l''$ where $l'' = max(l, l')$.

For any two conjuncts $v \geq l$ and $\neg(v \geq l')$ over v, if $l < l'$, then they can be simplified to $l \leq v < l'$. Otherwise, we have $l \geq l'$. In this case, the two conjuncts do not have an intersection, and there would be no instances at the leaf node and the training algorithm should not let this case happen.

For any two conjuncts $\neg(v \geq l)$ and $\neg(v \geq l')$ over v, they can be simplified to $v < l''$ where $l'' = min(l, l')$.

For any two conjuncts $v' \in C$ and $v' \in C'$ over a nominal feature v', they can be simplified into one conjunct $v' \in C''$ where $C'' = C \cap C'$. If any of the two conjuncts is negated, e.g., $\neg(v' \in C)$, we can simply take the complement set $C''' = C_v \setminus C$, where C_v is the full set of permitted discrete values of the feature v, and convert the negated conjunct into $v' \in C'''$. The remainder of the proof is analogous.

Thus, the left-hand side of the branch formula can be simplified to one conjunct per feature. □

Note that the simplification of Lemma 1 preserves logical equivalence of the set of decision rules while the other steps of this section do not. The reason why we use node filter in combination with Lemma 1 is that using all the nodes in a branch may result in very narrow and focused explanations, so we use θ as a parameter to adjust the scope.

The above two steps aim to shorten the decision rules. The third step is to reduce the number of rules by filtering out those of low quality.

Parameter 2 (Rule Filter ϕ) We measure the "quality" of a decision rule (RQ) by the following formula:

$$RQ(r) = \frac{(log_2(m) - H(l_r))}{log_2(m)} \times Acc \tag{2}$$

where r is the target rule, m is the number of classes, $H(l_r)$ is the entropy [19] of the leaf of the rule, and Acc is the accuracy of the corresponding tree on the OOB dataset. We remove a rule if its RQ is less than ϕ, which is a real number between 0 and 1.

The fourth step merges decision rules into *groups of the same class signature*.

Parameter 3 (Leaf Merger ψ) Given a vote distribution $(n_1, \cdots, n_m)$, where m is the number of classes, we convert the distribution into ratios $(\xi_1, \cdots, \xi_m)$ where each $\xi_i, 1 \leq i \leq m$, is the ratio of class i in the leaf node. The class signature of this leaf node is defined as the tuple $(\lceil \xi_i/\psi \rceil, \cdots, \lceil \xi_m/\psi \rceil)$, where ψ is a real number between 0 and 1.

Using the above definition, we divide the set of decision rules into a set of sets $\{G_1, \cdots, G_j\}$. Each G_i, $1 \leq i \leq j$, contains the set of rules of the same class signature. Intuitively, a larger ψ yields fewer distinct equivalent classes/groups and vice versa. We use this parameter to control the number of groups in the final explanation.

For a large-scale random forest, the filtered rules are still a large-scale formula set. In the last step, we control the number of decision rules we get.

Parameter 4 (Size Filter k) In each G_i, we take the number of instances in the leaf node of each rule as the **weight of the rule**, and we select k rules in a weight-first manner.

Associating each rule with the weight is crucial because now the vote distribution has been converted into ratios, and we have lost the information of the number of votes in the distribution. The weight retains this information. For example, we would prefer a ratio of $(0.7, 0.3)$ with 100 votes to overwhelm a ratio of $(0.1, 0.9)$ with 10 votes.

We denote the composition of the above steps as $Simp$, which produces a set R'_E of weighted decision rules, and give the procedure in Algorithm 3.

3.3 Prediction

To use R'_E in a classification task, let x be a data instance, we first find all the decision rules in R'_E that are satisfied by x, then multiply the class signature of each rule by the corresponding weight, and finally add up to get a tuple. The class with the largest value is the output. This procedure is denoted as

Algorithm 3 $Simp(R_E, \theta, \phi, \psi, k)$

Input: $R_E, \theta, \phi, \psi, k$
Output: R'_E

$R'_E \leftarrow \{\}$
for rule r in R_E **do** ▷ The object *rule* has three attributes: F(branch formula), D(vote distribution) and W(weight).
  if $RQ(r) < \phi$ **then** ▷ See Eq2
    Continue
  end if
  for node n in $F[r]$ **do**
    if $NQ(n) < \theta$ **then** ▷ See Eq1
      Delete n from r
    end if
  end for
  Simplify r according to Lemma 1
  $D[r] \leftarrow \lceil D[r]/\psi \rceil$
  if $\exists$ group $g \in R'_E$ and $Sig[g] \Leftrightarrow D[r]$ **then** ▷ The object *group* has two attributes: R(a set of rules), Sig(class signature).
    Insert r into g and keep top k weighted rules in $R[g]$
  else
    $Sig[newGroup] \leftarrow D[r]$
    Insert r in $R[newGroup]$
    Insert $newGroup$ in R'_E
  end if
end for

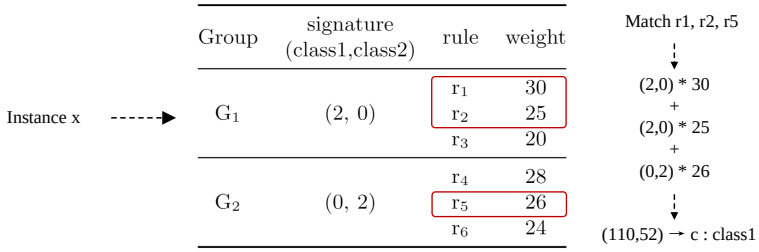

Fig. 2 An example of the prediction procedure using the simplified rules R'_E

$R'_E(x) = c$ where c is a class. We give an example in Fig. 2 where we put the satisfied rules in red boxes.

3.4 Optimal Explanations

Now we consider a step further: how to evaluate explanations and find the optimal one? Intuitively, a good explanation should meet the following two criteria:

- The classification behaviour of the explanation R'_E should be similar to the original model E.
- The explanation should be concise and small.

We use *fidelity* to measure the first criterion. Fidelity is defined as the degree of similarity between the predictions of R'_E and E on unseen data [20]. First, we take the test set without labels as X^*. Then we use the classification results on X^* from the original model E as the "ground truth". Lastly, we evaluate the classification accuracy of the explanation R'_E on X^* as fidelity. The fidelity component is denoted as $Fd(R'_E, E, X^*)$.

The second criterion is *scale* which is measured by the total number of conjuncts in the rules of R'_E and is denoted as $Sc(R'_E)$.

The *score* of an explanation R'_E is defined as

$$S_{opt}(R'_E) = Fd(R'_E, E, X^*) \times \epsilon + \frac{1-\epsilon}{1 + e^{5\times(\frac{Sc(R'_E)}{m\times n} - 1)}} \tag{3}$$

where ϵ is a real number between 0 and 1, n is the number of features, and m is the number of classes. The intuition is that the score grows linearly with fidelity, but it drops significantly when the explanation is too large. The second component is a sigmoid-shaped function. Also, a large ϵ puts more importance on fidelity, and a small ϵ puts more importance on scale.

To obtain an optimal explanation, we use the *linearly decreasing inertia weigh particle swarm optimization* algorithm (LDIW-PSO) [21] to optimise the parameters mentioned in section 3.2: $(\theta_{opt}, \phi_{opt}, \psi_{opt}, k_{opt}) = argmax(S_{opt})$, where S_{opt} is the fitness function to be optimised. Then we apply the optimal parameters to *Simp* and obtain the optimal explanation R_{opt}. We refer to the above procedure as *OptExplain*, and the algorithm is given in Algorithm 4, in which we denote the application of the LDIW-PSO algorithm by $PSO(E, S)$, where E is the original model and S is the fitness function. Note that the LDIW-PSO application iteratively calls $Simp()$ to obtain intermediate results.

Algorithm 4 $OptExplain(E, X^*, \epsilon)$

Input: E, X^*, ϵ
Output: R_{opt}

1: Optimal parameters $(\theta_{opt}, \phi_{opt}, \psi_{opt}, k_{opt}) \leftarrow PSO(E, S_{opt})$ ▷ X^* and ϵ are parameters of S_{opt}. See Eq 3.
2: $R_E \leftarrow$ Extract decision rules from E via Algorithm 2
3: $R_{opt} \leftarrow Simp(R_E, \theta_{opt}, \phi_{opt}, \psi_{opt}, k_{opt})$

3.5 Profile of Equivalent Classes

An explanation with high fidelity is usually large, while a concise explanation can allow users to quickly understand the predictive behavior of the model at

the sacrifice of fidelity. Sometimes the latter is preferred to draw a high-level conclusion of the classification behavior. We propose a new method called the *Profile of Equivalent Classes (ProClass)*, which describe the distribution of features that best match each class from the perspective of model.

The *ProClass* procedure is based on a explanation obtained by Algorithm 3. Different from Section 3.4, a explanation used for *ProClass* requires more rules, and the number of groups is equal to the number of classes. We denote the explanation as R''_E, and the number of groups in R''_E is denoted as $M(R''_E)$. For the above needs, we defined another *score* for the optimization in *ProClass*:

$$S_{pro}(R''_E) = (m - M(R''_E) + 1) \times Sc(R''_E) \tag{4}$$

where m is the number of classes. Using S_{pro} as fitness, we obtain the optimized result is denoted as R'_{opt}. An ideal R'_{opt} has m groups $\{G_1, \cdots, G_m\}$. For each group G_i $(1 \leq i \leq m)$, some rules in G_i may be mutually exclusive. To solve this problem, we associate the weight of each rule to each of its conjuncts, and send all the weighed conjuncts in the group to a SMT solver such as Z3 [22]. The solver will return a subset of satisfied conjuncts that maximise the total weight by solving a weighted maximum satisfiable (MAX-SAT) problem [23]. Then we simplify the conjuncts into one rule r'_i using Lemma 1. Performing the above steps on all groups, we get the profile of equivalent classes denoted as R_{pro}, which has the form $R_{pro} = \{r'_1 \rightarrow S_1, \cdots, r'_m \rightarrow S_m\}$, where r'_i is the logic for predicting the class S_i. The $ProClass$ algorithm is given in Algorithm 5.

Algorithm 5 $ProClass(E)$

Input: E
Output: R_{pro}

Optimized parameters $(\theta_{pro}, \phi_{pro}, \psi_{pro}, k_{pro}) \leftarrow PSO(E, S_{pro})$
$R_E \leftarrow$ Extract decision rules from E via Algorithm 2
$R'_{opt} \leftarrow Simp(R_E, \theta_{pro}, \phi_{pro}, \psi_{pro}, k_{pro})$
$R_{pro} \leftarrow \{\}$
for group g in R'_{opt} **do**
    $R[g] \leftarrow MAXSAT(R[g])$
    $r \leftarrow \top$
    for conjunct c in $R[g]$ **do**
        $r \leftarrow (r \wedge c)$
    end for
    $r \leftarrow (r \rightarrow Sig[g])$
    Insert r into R_{pro}
end for

4 Case Studies and Experiment

In this section, we demonstrate our method through case studies. We used scikit-learn to train random forest models. We implemented our method in Python and evaluated it on multiple datasets: adult, credit, diabetes, german, mnist, spambase, all of which are available on OpenML [24]. There are two important parameters in LDIW-PSO algorithm: particle size and iteration period. In our experiments, both particle size and iteration period are set to 20 by default. Experiments were conducted on a machine with an Intel Core i9-7960X CPU and 32GB RAM.

4.1 Case Study 1: Diabetes Prediction

We first evaluate *OptExplain* on diabetes dataset [25], which has 8 features, 2 classes, and 768 samples. The eight features are the number of times pregnant (preg), plasma glucose concentration (plas), diastolic blood pressure (pres), 2-hour serum insulin (insu), triceps skinfold thickness (skin), body mass index (mass), diabetes pedigree function (pedi) and age. We randomly select 100 samples as the testing set, and the rest as the training set. Then we train a random forest with 100 trees and unlimited depth.

Table 1 Optimized parameters and predictive performance of the explanation

(ϕ, θ, ψ, k)	R_E		R_{opt}		
	scale	accuracy	scale	accuracy	fidelity
(0.55, 0.45, 0.83, 3.0)	102584	80%	6	80%	92%

Table 2 The optimal explanation of a random forest model for diabetes

Groups	Class Signature (negative, positive)	Rules	Weight
Group1	(2.0, 0.0)	$pedi \leq 0.7$	30
		$plas \leq 130.0$	23
		$plas \leq 157.5$	21
Group2	(0.0, 2.0)	$mass > 28.7$	30
		$age > 27.5$	22
		$plas > 122.5$	20

We set ϵ to 0.9, and the explanation R_{opt} produced by *OptExplain* is shown in Table 1 and Table 2. Recall that R_E is equivalent to the original model E (Proposition 1). Both R_E and R_{opt} have 80% accuracy on the test set. R_E has 11106 rules with 102584 conjuncts, while R_{opt} has 6 rules with 6 conjuncts. The fidelity of R_{opt} is 92%, which means it is very similar to the original model.

By observing the decision rules, users can analyze what role each feature plays in the prediction process. In this explanation, if an instance has $plas > 157.5$, then the chance of prediction positive diabetes is high.

Table 3 gives the profile R_{pro} derived from *ProClass*. The profile is divided into two groups corresponding to equivalent class labels: negative diabetes and positive diabetes. Medical practitioners can quickly obtain the salient feature values corresponding to each class according to the profile.

Table 3 The profile of equivalent classes of a random forest model for diabetes

negative	**positive**
$2.0 < preg \leq 2.5$	$7.5 < preg \leq 8$
$90 < plas \leq 91.5$	$173.5 < plas \leq 175$
$74 < pres \leq 74.5$	$70 < pres \leq 71$
$28.5 < skin \leq 29$	$23.5 < skin \leq 24$
$61.5 < insu \leq 63$	$126.5 < insu \leq 127.5$
$25.9 < mass \leq 26$	$32.9 < mass \leq 33$
$pedi = 0.2$	$pedi = 0.5$
$23.5 < age \leq 24$	$33 < age \leq 33.5$

4.2 Case Study 2: Digit Recognition (MNIST)

In order to visually demonstrate the profile, we use the MNIST dataset [24] to illustrate *ProClass*. The MNIST dataset contains 70,000 images of size 28 $\times$ 28 pixels. In order to reduce the complexity, we randomly picked 10,000 samples from the dataset as the training set, and train a random forest with 100 trees and 15 max-depth. *ProClass* produces a profile R_{pro} which gives the range of some pixels. Then we take the median of the range as the pixel value. The remaining pixels that do not appear in the profile are set to light blue. Fig. 3 shows "how the prediction model see digits." The profile is not concerned with the peripheral pixels that are not significant in the prediction. It can be observed that the profile indeed shows human-readable visualisation of each class. The user can observe the visualisation and assess whether the ML model has potential issues when predicting certain classes. For example, by inspecting Figure 3, the user can see that the classification for digits 4, 5, 6, and 9 looks quite vague, which means that these cases are potential weaknesses of the model and retraining may be required. It can also be observed that there might be adversarial samples when trying to distinguish 4 and 9, as well as 5, 6 and 8 because these cases largely overlap.

We note that *OptExplain* is designed to extract the *logical decision-making* of the ML model by analysing features. This procedure is (though possible) not suitable for image datasets because the features are simply pixels, and logical formulae of pixels are hard to understand, which defeats the purpose of the explanation. This is part of the reason why we developed *ProClass*, which can visualise the explanation for image datasets from a different angle.

Consequently, we do not include the MNIST dataset in the experiment of *OptExplain*.

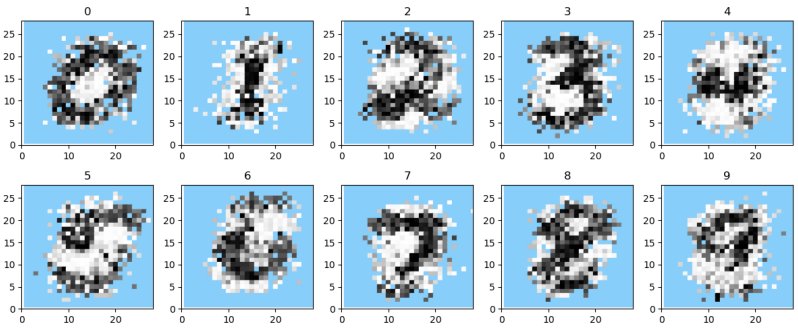

Fig. 3 As visualisation of classes profile (R_{pro}) on MNIST

4.3 Experiment and Comparison

We compared the proposed method to recent methods — Hara and Hayashi's approach named degragTrees [13], and Deng's approach named inTrees [14]. In Hara's work, defragTrees has been compared with BATrees [26], inTrees [14] and Node Harvest [27]. Their result suggests that defragTrees generated smaller set of rules with higher fidelity than the other methods. We choose the following datasets: adult, credit, diabetes, german, spambase [25]. We split the datasets into two subsets at random: a 70% training set, and a 30% testing set. Then we train the ensemble trees with 100 trees and 10 max-depth for *OptExplain* and defragTrees. inTrees is implemented in R language, we use its default setting of 100 trees. For *OptExplain*, we set two experimental groups and set ϵ as 0.5 and 0.9 respectively.

We conducted the experiment over ten random data realizations for each dataset. Table 4 shows the average values of the ten tests. The number in bold is the best scale/fidelity value for each dataset. Table 4 shows that the scale of explanations generated by *OptExplain* with 0.5 ϵ is the smallest except on the adult and german datasets, and the fidelity is generally better than defragTrees and inTrees. The small-scale explanation on german generated by defragTrees has much lower fidelity. *OptExplain* with 0.9 ϵ can generate the highest fidelity explanation with similar scale than defragTrees. In both settings of ϵ, *OptExplain* generates superior explanations in most cases. In addition, our approach supports user-customized generation explanations by modifying parameters ϵ.

Finally, we visually compare the computation time in Fig. 4, and show the mean time of 10 tests. We run *OptExplain* on models of 100 trees with depth of 10 and 30. The method inTrees and defragTrees on 100 is also used as a reference. The results show that inTrees has the best computational performance; however, this method often produces explanations with lower fidelity

Table 4 Comparison of *OptExplain* with inTrees and defragTrees in terms of scale and fidelity of the explanation on various datasets.

Data	inTrees		defragTrees		OptExplain $\epsilon = 0.5$		$\epsilon = 0.9$	
	scale	fidelity%	scale	fidelity%	scale	fidelity%	scale	fidelity%
adult	**23.4**	80.7	65.5	81.9	37.2	87.0	43.2	**88.4**
credit	24	76	8.7	85	**2.8**	94.2	9.8	**94.6**
diabetes	152	72.3	14.2	74.8	**4**	85.8	8.3	**88.3**
german	14.2	70.2	**5.4**	69.8	19	87.3	50.2	**88.7**
spambase	51.3	82.7	82	91.7	**22**	91.6	40.6	**92.9**

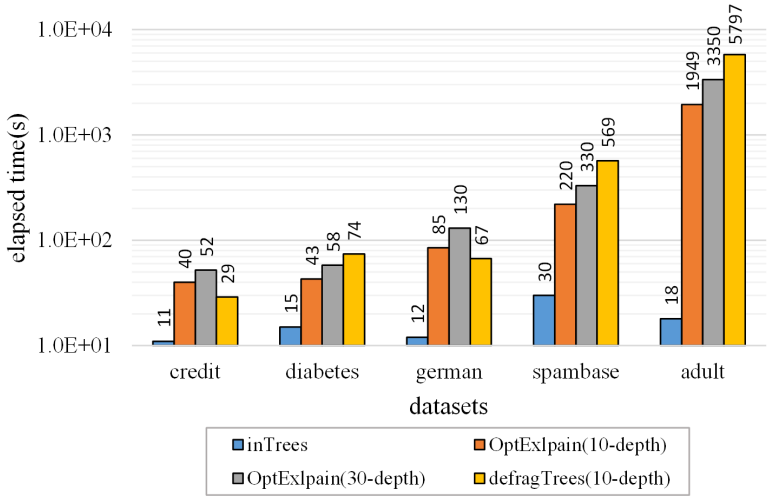

Fig. 4 Comparison of OptExplain with inTrees and defragTrees in terms of computational speed. The y-axis is in logarithmic scale.

compared to the other two methods, as can be observed in Table 4. *OptExplain* has better computational performance than defragTrees on adult, diabetes and spambase. For the above three datasets, *OptExplain* can generate explanations for trees of depth 30 faster than defragTrees can for depth 10. For a concrete example, the mean computation time of *OptExplain* on the adult dataset is 1,949 s, while the time of defragTrees is 5,797 s (both depth 10). Overall, OptExplain on average provides better quality explanations in a shorter time than defragTrees.

4.4 Discussion

The first novel feature of this work is *OptExplain*, which optimizes the extraction process according to the objective function Eq 3 to obtain not just *an* explanation but the optimal explanation. The optimal explanation is a trade-off between scale and fidelity. Most other methods simply give an explanation

without examining whether the explanation is good or not. If the explanation is too large, it will be hard or time-consuming for users to understand (e.g., [28] Figure 14), whereas if it is too small, it may be of poor fidelity, so the trade-off is particularly important. Table 1 and Table 2 show that our method can obtain small size and high fidelity explanations, and Figure 4 shows that we do compute them in a reasonable time, often faster than comparable methods such as defragTrees, though not as fast as inTrees, which often yields worse explanations.

The other new feature we provide is *ProClass*, which is useful for showing the global decisions of ML models. Complementing *OptExplain*, *ProClass* can be used to visualise global explanations as "heatmaps" for image datasets.

Users can adjust the objective function parameter ϵ of *OptExplain* according to domain knowledge and application scenarios so as to adjust whether scale or fidelity is more important in a specific scenario. With *ProClass*, domain knowledge could help to improve the original model by inspecting the explanations for each class and assessing the strengths and weaknesses of the ML model, as discussed in Section 4.2.

5 Conclusion and Future Work

This paper presents a streamlined procedure for extracting optimal *logical* explanations from ensemble trees models. As an additional feature, our method can also output the "profile" of each class so that the user can *see* how the model predicts in different cases. We give two case studies to illustrate how our method works and show that our method outperforms state-of-the-art through experimental results.

In future work, we plan to use this work to perform efficient verification tasks. A general *sound and complete* verification algorithm for ensemble trees is impractical [29]. As a step back, we can look at the software testing perspective: since the explanation mimics the behaviour of the original model, if it violates a property, then it is likely that the original model would fail the verification, too. In such cases, we can use the explanation to constrain the search space when finding counterexamples. A prototype built on top of this paper and the above idea has shown potential and is published [30]. Another important future direction is to convert deep neural networks to ensemble trees and extend this work to explain deep learning. Existing methods only induce a single decision tree whose predictive performance is incomparable to the deep learning model. We plan to develop a new approach to convert neural networks to a set of decision trees instead, and then this work can be directly applied to obtain explanations.

We note that different aggregation strategies of voting in random forest or boosting algorithms may affect the classification performance in some applications, as discussed and evaluated in-depth in [31]. However, voting mechanisms are non-trivial to be integrated into explanations and may add unnecessary complexity to the outcome, so our methods focus on the decision

rules extracted from the internal nodes of decision trees. The integration of voting in explanation is left as future work.

Declarations

Funding

Yanhong Huang's work is partially supported by National Key Research and Development Program (2019YFB2102602).

Conflict of interest

The authors have no conflicts of interest to declare that are relevant to the content of this article.

Ethics approval

Not applicable

Consent to participate

Not applicable

Consent for publication

Not applicable

Availability of data and materials

All data generated or analysed during this study are included in this published article (and its supplementary information files).

Code availability

The code is available at https://github.com/GreeenZhang/OptExplain.

Authors' contributions

Gelin zhang: Software, Investigation, data curation, Writing - review and editing; **Zhé Hóu**: Conceptualization, Methodology, Writing – original draft; **Yanhong Huang**: Resources, Validation; **Jianqi Shi**: Supervision, Conceptualization, Validation; **Hadrien Bride**: Methodology, Software; **Jin Song Dong**: Conceptualization, Validation; **Yongsheng Gao**: Investigation, Writing - review and editing. All authors read and approved the final manuscript.

References

[1] Ho, T.K.: Random decision forests. In: Proceedings of 3rd International Conference on Document Analysis and Recognition, vol. 1, pp. 278–282 (1995). IEEE

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

[3] Freund, Y., E Schapire, R.: A short introduction to boosting. Journal of Japanese Society for Artificial Intelligence **14**, 771–780 (1999)

[4] Friedman, J.H.: Stochastic gradient boosting. Computational statistics & data analysis **38**(4), 367–378 (2002)

[5] Pafka, S.: A minimal benchmark for scalability, speed and accuracy of commonly used open source implementations of the top machine learning algorithms for binary classification. https://github.com/szilard/benchm-ml (2018)

[6] Bride, H., Dong, J., Dong, J.S., Hóu, Z.: Towards dependable and explainable machine learning using automated reasoning. In: Formal Methods and Software Engineering - 20th International Conference on Formal Engineering Methods, ICFEM 2018, Gold Coast, QLD, Australia, November 12-16, 2018, Proceedings, pp. 412–416 (2018)

[7] Ltd, D.I.P.: Silas. https://depintel.com/silas/ (2018)

[8] Ribeiro, M.T., Singh, S., Guestrin, C.: "why should I trust you?": Explaining the predictions of any classifier. In: Proceedings of the 22nd ACM SIGKDD International Conference on Knowledge Discovery and Data Mining, San Francisco, CA, USA, August 13-17, 2016, pp. 1135–1144 (2016)

[9] Lundberg, S.M., Lee, S.-I.: A unified approach to interpreting model predictions. In: NIPS (2017)

[10] Hinton, G., Frosst, N.: Distilling a neural network into a soft decision tree. (2017). https://arxiv.org/pdf/1711.09784.pdf

[11] Ribeiro, M.T., Singh, S., Guestrin, C.: Anchors: High-precision model-agnostic explanations. In: Proceedings of the AAAI Conference on Artificial Intelligence, vol. 32 (2018)

[12] Hatwell, J., Gaber, M.M., Azad, R.M.A.: Chirps: Explaining random forest classification. Artificial Intelligence Review **53**, 5747–5788 (2020)

[13] Hara, S., Hayashi, K.: Making tree ensembles interpretable: A bayesian model selection approach. In: International Conference on Artificial

Intelligence and Statistics, pp. 77–85 (2018). PMLR

[14] Deng, H.: Interpreting tree ensembles with intrees. International Journal of Data Science and Analytics **7**(4), 277–287 (2019)

[15] Quinlan, J.R.: Induction of decision trees. Mach. Learn. **1**(1), 81–106 (1986). https://doi.org/10.1023/A:1022643204877

[16] Cui, Z., Chen, W., He, Y., Chen, Y.: Optimal action extraction for random forests and boosted trees. In: Proceedings of the 21th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining. KDD '15, pp. 179–188. ACM, New York, NY, USA (2015). https://doi.org/10.1145/2783258.2783281. http://doi.acm.org/10.1145/2783258.2783281

[17] Friedman, J.H.: Greedy function approximation: A gradient boosting machine. Annals of Statistics **29**, 1189–1232 (2000)

[18] Kullback, S.: Information theory and statistics. (1959)

[19] Shannon, C.: A mathematical theory of communication. Bell Syst. Tech. J. **27**, 379–423 (1948)

[20] Papenmeier, A., Englebienne, G., Seifert, C.: How model accuracy and explanation fidelity influence user trust. arXiv preprint arXiv:1907.12652 (2019)

[21] Shi, Y., Eberhart, R.C.: Empirical study of particle swarm optimization. In: Proceedings of the 1999 Congress on Evolutionary Computation-CEC99 (Cat. No. 99TH8406), vol. 3, pp. 1945–19503 (1999). https://doi.org/10.1109/CEC.1999.785511

[22] de Moura, L., Bjørner, N.: Z3: An efficient smt solver. In: Ramakrishnan, C.R., Rehof, J. (eds.) Tools and Algorithms for the Construction and Analysis of Systems, pp. 337–340. Springer, Berlin, Heidelberg (2008)

[23] Du, D., Gu, J., Pardalos, P.: Satisfiability problem: Theory and applications. (1997)

[24] OpenML: openml.org. https://www.openml.org (Accessed 2019)

[25] Dua, D., Graff, C.: UCI Machine Learning Repository (2017). http://archive.ics.uci.edu/ml

[26] Breiman, L., Shang, N.: Born again trees. (1996)

[27] Meinshausen, N.: Node harvest. (2009)

[28] Wan, A., Dunlap, L., Ho, D., Yin, J., Lee, S., Jin, H., Petryk, S., Bargal, S.A., Gonzalez, J.E.: NBDT: Neural-Backed Decision Trees. arXiv (2020). https://doi.org/10.48550/ARXIV.2004.00221. https://arxiv.org/abs/2004.00221

[29] Törnblom, J., Nadjm-Tehrani, S.: Formal Verification of Random Forests in Safety-Critical Applications: 6th International Workshop, FTSCS 2018, Gold Coast, Australia, November 16, 2018, Revised Selected Papers, pp. 55–71 (2019)

[30] Wang, B., Hóu, Z., Zhang, G., Shi, J., Huang, Y.: Tree ensemble property verification from a testing perspective. In: Accepted by the 33rd International Conference on Software Engineering and Knowledge Engineering (SEKE), Pittsburgh, USA (2021)

[31] Bride, H., Cai, C., Dong, J., Dong, J.S., Hóu, Z., Mirjalili, S., Sun, J.: Silas: A high-performance machine learning foundation for logical reasoning and verification. Expert Syst. Appl. **176**, 114806 (2021). https://doi.org/10.1016/j.eswa.2021.114806