Making Tree Ensembles Interpretable: A Bayesian Model Selection Approach

Satoshi Hara
Osaka University, Japan
JST, ERATO, Kawarabayashi Large Graph Project
satohara@ar.sanken.osaka-u.ac.jp

Kohei Hayashi
National Institute of Advanced
Industrial Science and Technology, Japan
hayashi.kohei@gmail.com

Abstract

Tree ensembles, such as random forests, are renowned for their high prediction performance. However, their interpretability is critically limited due to the enormous complexity. In this study, we propose a method to make a complex tree ensemble interpretable by simplifying the model. Specifically, we formalize the simplification of tree ensembles as a model selection problem. Given a complex tree ensemble, we aim at obtaining the simplest representation that is essentially equivalent to the original one. To this end, we derive a Bayesian model selection algorithm that optimizes the simplified model while maintaining the prediction performance. Our numerical experiments on several datasets showed that complicated tree ensembles were approximated interpretably.

1 Introduction

Tree ensembles such as random forests [1] and boosted trees [2] are popular machine learning models, particularly for prediction tasks. A tree ensemble builds numerous decision trees that divide an input space into a ton of tiny *regions*, places their own outputs for all the regions, and makes predictions by averaging all the outputs. Owing to the power of model averaging, their prediction performance is considerably high, and they are one of the must-try methods when dealing with real problems. Indeed, XGBoost [3], the state-of-the-art tree ensemble method, is one of the most popular methods in machine learning competitions [4].

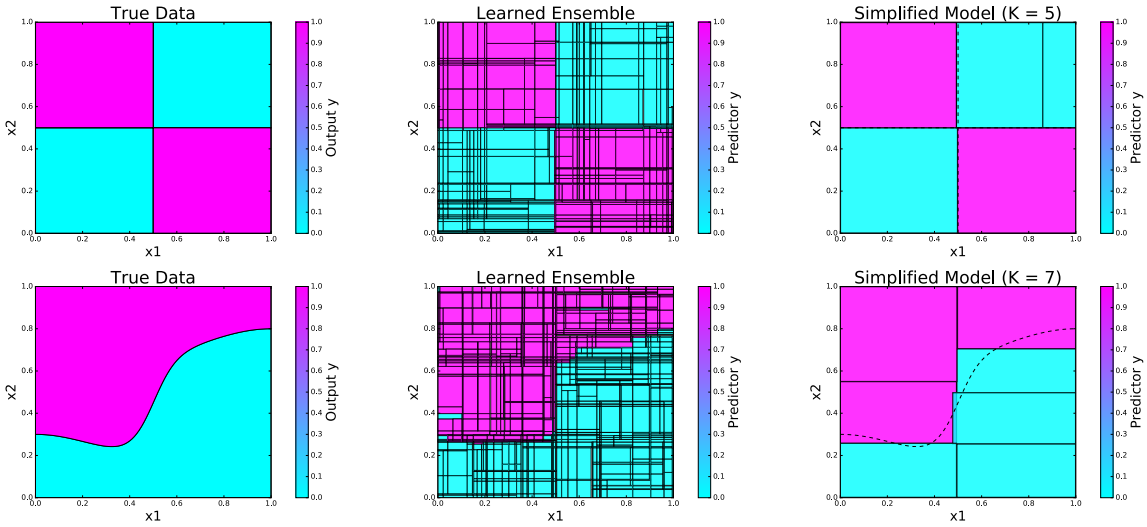
Proceedings of the 21^{st} International Conference on Artificial Intelligence and Statistics (AISTATS) 2018, Lanzarote, Spain. PMLR: Volume 84.

However, this high prediction performance of the tree ensemble makes large sacrifices of interpretability. Because every tree generates different regions, the resulting prediction model is inevitably *fragmented*, i.e., it has a lot of redundancy and becomes considerably complicated (Figure 1(b)), even if the original data are simply structured (Figure 1(a)). The total number of regions is usually over a thousand, which roughly means that thousands of different rules are involved in the prediction. Such a large number of rules are nearly impossible for humans to interpret.

How can we make a tree ensemble more interpretable? Clearly, reducing the number of regions, or equivalently, reducing the number of rules, simplifies the model and improves its interpretability. However, if the model is too simplified, we may overlook important rules behind the data. Also, oversimplification of the model possibly degrades its prediction performance. These observations imply that there is a trade-off between the number of regions and the prediction performance when simplifying the model.

In statistics, similar trade-offs have comprehensively been addressed as the model selection problem. Given multiple models, model selection methods typically aim to choose the model that achieves the best generalization performance, i.e., it can predict well for new data [5, 6]. Since too complex models cause overfitting, simple models tend to be selected. One of the most popular model selection is Bayesian model selection [7]. Bayesian model selection uses the marginal likelihood as a criterion, which eliminates redundant models as the Occam's razor principle [6]. This is a desirable property for tree ensemble simplification—using Bayesian model selection, we can find a simplified expression of the tree ensemble with smaller number of regions that is essentially equivalent to the original one.

Though the Bayesian model selection is a promising approach, there are two difficulties. First, popular tree ensembles such as XGBoost are not probabilistic



(a) Original Data (b) Learned Tree Ensembles (c) Simplified Models

Figure 1: The original data (a) are learned by tree ensembles with number of regions (b). In this example, the first five trees in the ensembles generated around 1,000 regions. The complicated ensembles (b) are *defragged* into a few regions using the proposed method (c). Each rectangle shows each input region specified by the model.

models, and their marginal likelihoods are not defined. The Bayesian model selection is therefore not directly applicable to them. Second, the model simplification problem potentially incurs computational intractability. For good model simplification, we have to make large regions that approximate the original tree ensemble as in Figure 1(c). However, because the possible configurations of input regions, e.g., the shape and the location of the regions, can be infinitely many, the full search of all possible candidates is infeasible.

In this study, we propose a simplification method for tree ensembles, both for classification and regression tasks, in a Bayesian manner. Suppose a tree ensemble that is already learnt with a number of regions (Figure 1(b)) is given. Our objective is to *defrag* the tree ensemble into a simple model using a smaller number of regions, as shown in Figure 1(c). In the proposed method, we tackle the two difficulties by using the following approaches:

1. We adopt a probabilistic model representation of the tree ensemble so that Bayesian model selection to be applicable.
2. We search for a simple yet accurate model by optimizing the number of regions, their shapes, and locations through estimating parameters of the probabilistic model.

With these approaches, the model simplification problem then reduces to the Bayesian model selection problem that optimizes input regions for simplification while maintaining the prediction performance as much as possible. For efficient Bayesian model selection, we further adopt the next approach:

3. We use an algorithm called *factorized asymptotic Bayesian (FAB) inference* [8, 9].

FAB inference provides an asymptotic approximation of the marginal likelihood in a tractable form. We also propose an acceleration technique for FAB inference so that the proposed method to be scalable to large datasets. Our numerical experiments on several datasets showed that complicated tree ensembles were approximated adequately while maintaining prediction performance.

Notation: For $N \in \mathbb{N}$, $[N] = \{1, \ldots, N\}$ denotes the set of integers. For a proposition a, $\mathbb{I}(a)$ denotes the indicator of a, i.e., $\mathbb{I}(a) = 1$ if a is true, and $\mathbb{I}(a) = 0$ if a is false. Let $\boldsymbol{x} = (x_1, x_2, ..., x_D) \in \mathbb{R}^D$ be a D-dimensional input and $y \in \mathcal{Y}$ be an output. Here, for regression problems, the output y is numeric, i.e., $\mathcal{Y} = \mathbb{R}$. For classification problems with C categories, the output y is one-hot vector, i.e., for category $c \in [C]$, $y_c = 1$ and $y_{c'} = 0$ for $c \neq c'$.

2 Preliminaries

Decision Tree The decision tree makes the predication depending on the leaf node to which the input $\boldsymbol{x}$ belongs. The corresponding leaf node is determined by traversing the tree from the root. In each internal node j of the tree, the input $\boldsymbol{x}$ is directed to one of two child nodes depending on whether the statement $x_{d_j} > b_j$ is true or not, where $d_j \in [D]$ is a feature index checked at the node j and $b_j \in \mathbb{R}$ is a threshold. For example, suppose the case that $D = 3$ and the leaf node i is described by four internal nodes as

$x_1 > b_1$, $x_2 \leq b_2$, $x_3 > b_3$, $x_3 \leq b'_3$, and let $z_i \in \mathcal{Y}$ be the predictive value of the leaf node i. Then, if the input $\boldsymbol{x}$ arrives at the leaf node i by traversing these internal nodes, the prediction mechanism is described as a *rule*:

$$\underbrace{x_1 > b_1}_{statement} \wedge \underbrace{x_2 \leq b_2}_{statement} \wedge \underbrace{x_3 > b_3}_{statement} \wedge \underbrace{x_3 \leq b'_3}_{statement} \Longrightarrow y = z_i.$$

We refer to each component as a *statement* hereafter.

A list of statements can also be represented as a *region*. The above list of statements (the left hand side of the rule) can be written as $\boldsymbol{x} \in R_i := (b_1, \infty) \times (-\infty, b_2] \times (b_3, b'_3]$. Note that the regions are mutually disjoint, i.e., $R_i \cap R_{i'} = \emptyset$ if $i \neq i'$. Letting $\mathcal{Z} = \{z_i\}_{i=1}^I$ and $\mathcal{R} = \{R_i\}_{i=1}^I$, the decision tree with I leaf nodes can be expressed as follows:

$$f(\boldsymbol{x}; \mathcal{Z}, \mathcal{R}, I) := \textstyle\sum_{i=1}^I z_i \mathbb{I}(\boldsymbol{x} \in R_i). \tag{1}$$

Tree Ensemble The tree ensemble makes a prediction by combining the outputs from T decision trees. Suppose that the t-th decision tree has I_t leaf nodes with predictive values $\mathcal{Z}_t$ and regions $\mathcal{R}_t$. With weights $w_t \in \mathbb{R}$ on each decision tree $t \in [T]$, the output of the tree ensemble y is determined by the weighted average $y = \sum_{t=1}^T w_t f(\boldsymbol{x}; \mathcal{Z}_t, \mathcal{R}_t, I_t)$ for regression, or the weighted voting $y = \mathrm{argmax}_c \sum_{t=1}^T w_t \mathbb{I}(f(\boldsymbol{x}; \mathcal{Z}_t, \mathcal{R}_t, I_t) = c)$ for classification. This formulation includes the ordinary random forests [1], boosted trees [2], and XGBoost [3].

Extracting Rules from Tree Ensemble To interpret the tree ensemble, we need to extract rules from it. This corresponds to finding input regions and corresponding predictive values as in the single tree case (1). This can be achieved by considering multiple regions assigned by each tree to the input $\boldsymbol{x}$. Suppose that the region $R_{i_t}^t$ is assigned to the input $\boldsymbol{x}$ in the t-th tree for each $t \in [T]$. This means that the input $\boldsymbol{x}$ belongs to the intersection of those regions, namely $R_g = \cap_{t=1}^T R_{i_t}^t$. The predictive value corresponding to R_g can be expressed as $z_g = \sum_{t=1}^T w_t z_{i_t}^t$ for regression, and $z_g = \mathrm{argmax}_c \sum_{t=1}^T w_t \mathbb{I}(z_{i_t}^t = c)$ for classification. As the result, the tree ensemble can be expressed as $f(\boldsymbol{x}; \mathcal{Z}, \mathcal{R}, G) = \sum_{g=1}^G z_g \mathbb{I}(\boldsymbol{x} \in R_g)$, where G is the number of total regions determined by the tree ensemble, $\mathcal{Z} = \{z_g\}_{g=1}^G$, and $\mathcal{R} = \{R_g\}_{g=1}^G$. Because each region and predictive value corresponds to a rule, now we obtain the rules of the tree ensemble as $\mathcal{Z}$ and $\mathcal{R}$.

3 Problem Formulation

The aforementioned expression indicates that the tree ensemble can be expressed using G regions. Because $\mathcal{R}$ is generated from all the possible combination of the regions of the individual trees $\mathcal{R}_1, \ldots, \mathcal{R}_T$, the number of regions G can grow exponentially in the number of trees T, which makes the interpretation of the tree ensemble almost impossible. For example, Figure 1(b) shows that even five trees can generate more than a thousand regions.

To make the tree ensemble with large G interpretable, we approximate it using a smaller number of regions. Once the tree ensemble is approximated using a few regions as in Figure 1(c), it is easy to interpret the underlying rules in the model. This idea is formulated as the following problem.

Problem 1 *Given $G \in \mathbb{N}$ predictive values $\mathcal{Z} = \{z_g\}_{g=1}^G$ and regions $\mathcal{R} = \{R_g\}_{g=1}^G$, find $K \ll G$ predictive values $\mathcal{Z}' = \{z'_k\}_{k=1}^K$ and regions $\mathcal{R}' = \{R'_k\}_{k=1}^K$ such that, for any $\boldsymbol{x} \in \mathbb{R}^D$,*

$$f(\boldsymbol{x}; \mathcal{Z}, \mathcal{R}, G) \approx f(\boldsymbol{x}; \mathcal{Z}', \mathcal{R}', K). \tag{2}$$

In the following sections, we describe the proposed method for solving Problem 1. The proposed method is applicable to any tree ensemble defined in Section 2.

4 Probabilistic Model Expression

To solve Problem 1, we need to optimize the number of regions K, the predictors $\mathcal{Z}'$, and the regions $\mathcal{R}'$. Here, we introduce a probabilistic model that expresses the predictive values and the regions.

4.1 Binary Vector Expression of the Regions

First, we modify the representation of the input $\boldsymbol{x}$ and the regions $\mathcal{R}$ for later convenience. Suppose that the tree ensemble consists of L statements in total, i.e., the decision trees in the ensemble have L internal nodes in total. By definition, each input region $R_g \in \mathcal{R}$ is uniquely characterized by the combination of L statements, meaning that R_g is represented by the binary vector $\tilde{\boldsymbol{\eta}}_g \in \{0,1\}^L$, where $\tilde{\eta}_{g\ell} = 1$ if $x_{d_\ell} > b_\ell$ for all $\boldsymbol{x} \in R_g$, and $\tilde{\eta}_{g\ell} = 0$ otherwise. Figure 2 illustrates an example. By using this binary vector $\tilde{\boldsymbol{\eta}}_g$, we have $\mathbb{I}(\boldsymbol{x} \in R_g) = \mathbb{I}(\boldsymbol{s}(\boldsymbol{x}) = \tilde{\boldsymbol{\eta}}_g)$, where the ℓ-th element of $\boldsymbol{s}(\boldsymbol{x}) \in \{0,1\}^L$ is defined as $s_\ell(\boldsymbol{x}) = \mathbb{I}(x_{d_\ell} > b_\ell)$. To simplify the notation, we use $\boldsymbol{s}$ to denote $\boldsymbol{s}(\boldsymbol{x})$, and we refer to $\boldsymbol{s}$ as a *binary feature* of the input $\boldsymbol{x}$.

4.2 Probabilistic Model Expression of the Regions

As shown in Figure 1(b), the tree ensemble splits the input region into small fragments. To derive a simplified model as in Figure 1(c), we need to merge the

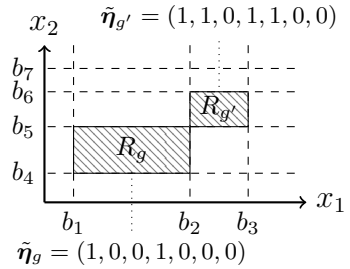


Figure 2: The region $R_g = \{(x_1, x_2) \mid b_1 < x_1 \leq b_2, b_4 < x_2 \leq b_5\}$ is expressed by the binary vector $\tilde{\boldsymbol{\eta}}_g = (1,0,0,1,0,0,0)$: $\tilde{\eta}_{g1}$ is 1 because R_g satisfies $x_1 > b_1$ while $\tilde{\eta}_{g2}$ is 0 because R_g does not satisfy $x_1 > b_2$. The region $R_{g'}$ is also expressed by $\tilde{\boldsymbol{\eta}}_{g'}$ in the similar manner.

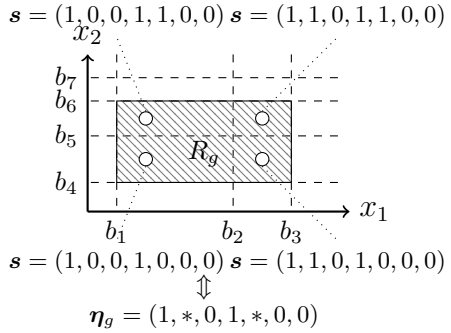


Figure 3: The concatenated region $R_g = \{(x_1, x_2) \mid b_1 < x_1 \leq b_3, b_4 < x_2 \leq b_6\}$ is indicated by the vector $\boldsymbol{\eta}_g = (1,*,0,1,*,0,0)$ where $*$ denotes the value between 0 and 1.

small fragments to make a large region. We achieve this by interpreting the region R_g as a generative model of the binary feature $\boldsymbol{s}$.

From the definition, the next equation holds:

$$\mathbb{I}(\boldsymbol{s} = \tilde{\boldsymbol{\eta}}_g) = \prod_{\ell=1}^{L} \tilde{\eta}_{g\ell}^{s_\ell} (1 - \tilde{\eta}_{g\ell})^{1-s_\ell}. \tag{3}$$

The equation can be "soften" by extending $\tilde{\boldsymbol{\eta}}_g$ from a binary vector to a $[0,1]$-continuous vector $\boldsymbol{\eta}_g \in [0,1]^L$. Namely, now the right-hand-side of (3) is the Bernoulli distribution on $\boldsymbol{s}$ with a model parameter $\boldsymbol{\eta}_g$ where $\eta_{g\ell}$ indicates a probability $\eta_{g\ell} = p(s_\ell = 1) = p(x_{d_\ell} > b_\ell)$.

With the extended vector $\boldsymbol{\eta}_g$, we can now express the concatenated region using $\boldsymbol{\eta}_g$ as shown in Figure 3. Here, $\eta_{g\ell} = 1$ means that the region R_g satisfies $x_{d_\ell} > b_\ell$ while $\eta_{g\ell} = 0$ means that R_g satisfies $x_{d_\ell} \leq b_\ell$. Moreover, with the extended vector $\boldsymbol{\eta}_g$, we have a third case when $\eta_{g\ell} \in (0,1)$: this corresponds to the case when some of $\boldsymbol{x} \in R_g$ satisfies $x_{d_\ell} > b_\ell$ while some other $\boldsymbol{x}' \in R_g$ satisfies $x'_{d_\ell} \leq b_\ell$, i.e., the boundary $x_{d_\ell} = b_\ell$ is inside R_g, and hence does not affect the definition of the region R_g.

The probabilistic version of (3) is then given as the following generative model:

$$p(\boldsymbol{s}|g) := \prod_{\ell=1}^{L} \eta_{g\ell}^{s_\ell} (1 - \eta_{g\ell})^{1-s_\ell}, \tag{4}$$

where $\eta_{g\ell} = p(s_\ell = 1) = p(x_{d_\ell} > b_\ell)$. Note that $\boldsymbol{\eta}_g$ is now a parameter of the model such that its zero–one pattern represents the shape of the concatenated region R_g. Hence, by optimizing $\boldsymbol{\eta}_g$, we can optimize the shape of the region R_g. The resulting parameter $\boldsymbol{\eta}_g$ is then translated to the corresponding statements describing the region R_g from its zero–one pattern.

4.3 Probabilistic Model Expression of the Tree Ensemble

We finally extend the tree ensemble into a probabilistic model. For this purpose, we introduce an indicator $\boldsymbol{u} \in \{0,1\}^G$ that describes which region the input $\boldsymbol{x}$ belongs to, i.e., if $\boldsymbol{x}$ belongs to the region R_g, $u_g = 1$ and $u_{g'} = 0$ for $g' \neq g$. We then model the probability of the pair $(y, \boldsymbol{s})$ given the indicator $\boldsymbol{u}$ as

$$p(y, \boldsymbol{s}|\boldsymbol{u}, G) = \prod_{g=1}^{G} \left(p(y|g) p(\boldsymbol{s}|g)\right)^{u_g}, \tag{5}$$

where $p(y|g)$ is the probability that y is output from the region R_g and $p(\boldsymbol{s}|g)$ is defined in (4). Specifically, we adopt the next output model for $p(y|g)$: $p(y|g) := \mathcal{N}(y|\mu_g, \lambda_g^{-1})$ for regression, and $p(y|g) := \prod_{c=1}^{C} \gamma_{gc}^{y_c}$ for classification. We denote the parameter of $p(y|g)$ by ϕ which is given by $\phi = \{\mu_g, \lambda_g\}_{g=1}^{G}$ for regression and $\phi = \{\{\gamma_{gc}\}_{c=1}^{C}\}_{g=1}^{G}$ with $\gamma_{gc} \geq 0$ and $\sum_{c=1}^{C} \gamma_{gc} = 1$ for classification. We also model the probability of $\boldsymbol{u}$ by $p(\boldsymbol{u}) = \prod_{g=1}^{G} \alpha_g^{u_g}$ where $\alpha_g \geq 0$ and $\sum_{g=1}^{G} \alpha_g = 1$. Here, α_g represents the probability of an event $u_g = 1$, i.e., the probability that input $\boldsymbol{x}$ belongs to the region R_g. Therefore we write as $\alpha_g = p(g|\alpha)$. Hence, the overall probabilistic expression of the tree ensemble can be expressed as follows:

$$p(y, \boldsymbol{s}, \boldsymbol{u}|\Pi, G) = \prod_{g=1}^{G} \left(p(y|g, \phi) p(\boldsymbol{s}|g, \eta) p(g|\alpha)\right)^{u_g} \tag{6}$$

where we explicitly written down the model parameters ϕ, η, and α for each component, and Π is the set of all parameters $\Pi = \{\phi, \eta, \alpha\}$.

4.4 Prediction

From (6), we can naturally derive the posterior distribution of y given the binary feature $\boldsymbol{s}$ using Bayes' rule. In the prediction stage, we want to derive the output y with the maximum posterior. However, searching for the maximum posterior is computationally demanding, and we therefore propose using the next two-step MAP estimate. We first find the region $\hat{g}$ with the maximum posterior $\hat{g} := \operatorname{argmax}_g p(g|\boldsymbol{s}, \Pi, G)$. Then, we output $\hat{y}$ with the maximum posterior given $\hat{g}$ which is $\hat{y} := \operatorname{argmax}_y p(y|\hat{g}, \phi)$.

5 Bayesian Model Selection Algorithm

Using the probabilistic model (6), Problem 1 is solved by estimating the model parameter Π and the number of regions $K \ll G$ so that the model in (6) is adequately simplified. If the number of regions K is known and fixed, we can derive the optimal model parameter Π of the simplified model using the maximum-likelihood estimation with the EM algorithm (see Appendix A). For an unknown K, we need to optimize it so that we can derive a simplified model with appropriate complexity. From standard Bayesian theory [7], this model selection problem can be formulated as the maximization of marginal log-likelihood. To solve the problem, we employ factorized asymptotic Bayesian (FAB) inference [8, 9], a Bayesian model selection algorithm that determines Π and K simultaneously. Because the number of regions K is automatically determined using FAB inference, we can avoid searching several possible values of K. Hence, we can solve the model selection problem efficiently.

5.1 FAB Inference

With observations $\mathcal{D} = \{(y^{(n)}, \boldsymbol{s}^{(n)})\}_{n=1}^N$, we aim to determine the optimal number of regions K by maximizing the marginal log-likelihood given by $\log p(\mathcal{D}|K) = \log \int p(\mathcal{D}|\Pi, K)p(\Pi)d\Pi$. Here, the likelihood is given as $p(\mathcal{D}|\Pi, K) = \prod_n p(y^{(n)}, \boldsymbol{s}^{(n)}|\Pi, K) = \prod_n \sum_{\boldsymbol{u}^{(n)}} p(y^{(n)}, \boldsymbol{s}^{(n)}, \boldsymbol{u}^{(n)}|\Pi, K)$.

Because the maximization of the marginal log-likelihood is intractable, we instead maximize the lower bound which is given by

$$\sum_{n=1}^N \sum_{k=1}^K \mathbb{E}_{q(U)}[u_k^{(n)}] \log p(y^{(n)}|k,\phi)p(\boldsymbol{s}^{(n)}|k,\eta)p(k|\alpha) - \omega \sum_{k=1}^K \log\left(\sum_{n=1}^N \mathbb{E}_{q(U)}[u_k^{(n)}] + 1\right) + H(q(U)), \tag{7}$$

where $\omega = (\dim\phi/K + L + 1)/2$, $q(U)$ is the distribution of U, and $H(q(U))$ is the entropy of $q(U)$. The derivation of this lower bound can be observed in the supplementary material (Appendix B). The EM-like FAB inference algorithm is then formulated as an alternating maximization of the lower bound with respect to q (E-step) and the parameter Π (M-step). See Algorithm 1 for the pseudo code, and Appendix C for the detailed algorithm derivation.

E-Step In E-Step, we update $q(U)$ so that the lower bound in (7) is maximized. Let $\beta_k^{(n)} = \mathbb{E}_{q(U)}[u_k^{(n)}] = q(u_k^{(n)})$. The optimal $\beta_k^{(n)}$ can be derived by iterating the next update until convergence:

$$\beta_k^{(n)} \propto f_k^{(n)} \exp\left(-\omega/(\textstyle\sum_{n=1}^N \beta_k^{(n)} + 1)\right), \tag{8}$$

Algorithm 1 FAB Inference

Input: Training data $\mathcal{D} = \{(y^{(n)}, \boldsymbol{s}^{(n)})\}_{n=1}^N$, maximum number of regions $K_{\max}$, tolerance δ
Output: # of regions K, Parameter $\Pi = \{\phi, \eta, \alpha\}$

```
Initialize parameter Π and {{q(u_k^(n))}_{k=1}^{K_max}}_{n=1}^N randomly
K ← K_max
while lower bound not converged do
  while not converged do
    Update {{q(u_k^(n))}_{k=1}^K}_{n=1}^N by (8)
  end while
  Remove k-th region when (1/N) Σ_{n=1}^N q(u_k^(n)) < δ
  K ← # of active regions
  Update Π by (9), (10), and (11)
end while
```

where $f_k^{(n)} = p(y^{(n)}|k,\phi)p(\boldsymbol{s}^{(n)}|k,\eta)p(k|\alpha)$.

M-Step In M-Step, we update Π so that the lower bound in (7) is maximized. Let $\beta_k^{(n)} = q(u_k^{(n)})$. The parameter $\Pi = \{\phi, \eta, \alpha\}$ is then updated as

$$\text{(regression):} \begin{cases} \mu_k = \frac{\sum_{n=1}^N \beta_k^{(n)} y^{(n)}}{\sum_{n=1}^N \beta_k^{(n)}}, \\ \lambda_k = \frac{\sum_{n=1}^N \beta_k^{(n)}}{\sum_{n=1}^N \beta_k^{(n)} (y^{(n)} - \mu_k)^2}, \end{cases} \tag{9}$$

$$\text{(classification):} \quad \gamma_{kc} = \frac{\sum_{n=1}^N \beta_k^{(n)} y_c^{(n)}}{\sum_{n=1}^N \beta_k^{(n)}}, \tag{10}$$

$$\eta_{k\ell} = \frac{\sum_{n=1}^N \beta_k^{(n)} s_\ell^{(n)}}{\sum_{n=1}^N \beta_k^{(n)}}, \qquad \alpha_k = \frac{1}{N}\sum_{n=1}^N \beta_k^{(n)}. \tag{11}$$

Region Truncation The iterative update of E-Step in (8) induces truncation of the region [8, 9]. For instance, when $\sum_{n=1}^N \beta_k^{(n)} = \epsilon \ll N$, in (8), the updated value $\beta_k^{(n)}$ is multiplied by $\exp(-\omega/(\epsilon+1)) \ll 1$ for all $n \in [N]$. The iterative multiplication of this small value results in $q(u_k^{(n)}) \approx 0$ for all $n \in [N]$, which means that the k-th region can be removed without affecting the marginal log-likelihood. With this region truncation, FAB inference automatically decides the number of regions K within the iterative optimization. Hence, we only need to specify a sufficiently large $K_{\max}$ as the initial value of K. We note that we can leave $K_{\max}$ as a constant (say, $K_{\max} = 10$) rather than the tuning parameter.

5.2 Scalable Computation of FAB Inference

The time complexity of the proposed FAB inference is dominated by E-Step which is $O(K_{\max}LN + \zeta K_{\max}N)$, where ζ is the number of iterations in E-Step. In E-Step, we first need to compute $f_k^{(n)}$ for all $k \in [K_{\max}]$ and $n \in [N]$ which requires $O(K_{\max}LN)$ time complexity. We then iteratively update the value of

$\beta_k^{(n)}$ based on (8). The one update step (8) for all $k \in [K_{\max}]$ and $n \in [N]$ requires $O(K_{\max}N)$ time complexity. The overall time complexity of E-Step is therefore $O(K_{\max}LN + \zeta K_{\max}N)$. In M-Step, the update of ϕ, η, and α require $O(K_{\max}N)$, $O(K_{\max}LN)$, and $O(K_{\max}N)$ time complexities, respectively. These complexities are dominated by the complexity of E-Step, and thus can be ignored.

We note that the time complexity $O(K_{\max}LN)$ sometimes gets prohibitive when L is large, i.e., when the tree ensemble is huge. To scale up the proposed FAB inference to large L, we propose a simple heuristic, a sampling FAB inference, based on the random sampling of the statements: we do not use all the statements in the tree ensemble but the randomly sampled subset. If the number of statements is reduced and L gets very small, FAB inference gets highly scalable.

The sampling idea follows from the intuition that most of the statements in the tree ensemble are redundant when we make a simplified expression. This is because similar statements such as $x_1 < 0.500$ and $x_1 < 0.501$ appear in the tree ensemble. When this subtle change on the threshold is ignorable, removing one of these two statements will have almost no impact to the resulting simplified expression. Random sampling of the statements can remove these redundancies effectively.

We now turn to the proposed heuristic, a sampling FAB inference. Suppose that we randomly sampled $\tilde{L}$ statements out of L statements in the tree ensemble. Then, the binary feature defined on the sampled statements is given by $\tilde{\boldsymbol{s}} \in \{0,1\}^{\tilde{L}}$ where $\tilde{s}_{\ell'} = \mathbb{I}(x_{\tilde{d}_{\ell'}} > \tilde{b}_{\ell'})$, and $\tilde{d}_{\ell'}$ and $\tilde{b}_{\ell'}$ denote the feature index and the threshold of each of the sampled statement, respectively. The generative model of $\tilde{\boldsymbol{s}}$ is given by $p(\tilde{\boldsymbol{s}}|g) := \prod_{\ell'=1}^{\tilde{L}} {\eta'}_{g\ell'}^{\tilde{s}_\ell}(1-\eta'_{g\ell'})^{1-\tilde{s}_{\ell'}}$, where $\eta'_{g\ell'} = p(\tilde{s}_{\ell'}=1) = p(x_{\tilde{d}_{\ell'}} > \tilde{b}_{\ell'})$. Here, we assume that the sampled version of this generative model becomes a good approximation of the original generative model (3) when we virtually increase the number of statements from $\tilde{L}$ to L, i.e., $p(\tilde{\boldsymbol{s}}|g)^{L/\tilde{L}} \approx p(\boldsymbol{s}|g)$.

With the above approximation, we can derive the approximate FAB lower bound by slightly modifying the original lower bound (7). The sampling FAB inference algorithm can then be derived by a slight modification of the original E-step (32): we replace $f_k^{(n)} = p(y^{(n)}|k,\phi)p(\boldsymbol{s}^{(n)}|k,\eta)p(k|\alpha)$ with $\tilde{f}_k^{(n)} = p(y^{(n)}|k,\phi)p(\tilde{\boldsymbol{s}}^{(n)}|k,\eta)^{L/\tilde{L}}p(k|\alpha)$. The M-step remains the same as the original FAB inference except that η is replaced with the sampled version η'.

Because the number of statements appearing in the sampling FAB inference is $\tilde{L}$ rather than L, its time complexity is $O(K_{\max}\tilde{L}N + \zeta K_{\max}N)$, which can be significantly smaller than the original FAB inference. For example, when the number of original statements is $L = 10,000$, the sampling FAB inference can be nearly 100 times faster when we set $\tilde{L} = 100$.

6 Related Work

Interpretability of complex machine learning models are now in high demand [10, 11]. Interpreting learned models allows us to understand the data and predictions more deeply [12, 13], which may lead to effective usage of data and models. For instance, we may be able to design a better prediction model by fixing the *bug* [14] in the model [15], or we can make a better decision based on the insights on the model [16, 13].

There are a few seminal studies on interpreting tree ensembles: *Born Again Trees* (BATrees) [17]; *interpretable Trees* (inTrees) [18]; and *Node Harvest* [19].

In BATrees, a single decision tree that mimics the tree ensemble is build. The tree ensemble is used to generate additional samples that are used to find the best split in the tree node. The single decision tree is then built to perform in a manner similar to that of the original tree ensemble. An important note regarding BATrees is that it tends to generate a deep tree, i.e., a tree with several complicated prediction rules which may be difficult to interpret. We also note that a similar method as BATrees was proposed independently by Zhou and Jiang [20].

The inTrees framework extracts rules from tree ensembles by treating tradeoffs among the frequency of the rules appearing in the trees, the errors made by the predictions, and the length of the rules. The fundamental difficulty with inTrees is that its target is limited to the classification tree ensembles. Regression tree ensembles are first transformed into the classification ones by discretizing the output, and then inTrees is applied to extract the rules. The number of discretization levels remains as a tuning parameter, which severely affects the resulting rules.

Node Harvest simplifies tree ensembles by using the shallow parts of the trees. In the first step, the shallow part of the trees (e.g., depth two trees) are extracted and the remaining parts are discarded. Node Harvest then combines the shallow trees so that they fit the training data well. The shortcoming of Node Harvest is that the derived simplified model is still an ensemble of the shallow trees. It is therefore still challenging to interpret the resulting simplified ensemble. It is also important to note that Node Harvest is designed for regression. Although it can handle binary classification as an extension of regression, it cannot handle classification with more than two categories.

Our proposed method overcomes the limitations of these existing methods: the resulting model tends to have only a few rules that are easy to interpret; it can handle both classification and regression tree ensembles; and there are no tuning parameters.

We also note that there are some works attempting to compress tree ensembles so that the model to fit in a small memory [21, 22]. They reduce the size of the model by pruning redundant leaves. Although they can make the model size small, the number of rules can remain as exponentially large as long as the model is retained as an ensemble. Hence, these methods does not fit our objective where we want a simple model with small number of rules for interpretation.

7 Experiments

We demonstrate the efficacy of the proposed method through synthetic and real-world data applications [1]. We used two synthetic data (Synthetic 1 and 2) and four real-world data [23] (Spambase, MiniBooNE, Higgs, and Energy). The used data are summarized in the supplementary material. The dataset Synthetic1 has a box-shaped class boundary (upper figure of Figure 1(a)). The dataset Synthetic2 has a more complicated class boundary (bottom figure of Figure 1(a)). Hence, it is more difficult to simplify the tree ensemble and derive a good approximate model.

Baseline Methods: We compared the proposed method to four baseline methods. The first three are BATrees [17], inTrees [18], and Node Harvest (NH) [19]. The last baseline is the depth-2 decision tree (DTree2). While the first three methods tend to generate tens or hundreds of rules, DTree2 generates only four rules. Hence, it is a good baseline method to compare with the proposed method that tends to generate only a few rules.

Implementations: In all experiments, we used `randomForest` package in R to train tree ensembles with 100 trees. The tree ensemble simplification methods are then applied to extract rules from the learned tree ensembles. The proposed method is implemented in Python. For the proposed method, we used the statement sampling heuristic with the sampling size $\tilde{L} = 100$. We set $K_{\max} = 10$ and ran FAB inference for 20 different random initial parameters, and derived learned 20 parameters $\{\Pi_m\}_{m=1}^{20}$. We then adopted the parameter with the smallest training error, i.e., $\tilde{\Pi} = \mathrm{argmin}_{\Pi_m} \mathrm{Error}(\mathcal{D}, \Pi_m)$ with $\mathrm{Error}(\mathcal{D}, \Pi) := \sum_{n=1}^{N} \left(y^{(n)} - \hat{y}^{(n)}\right)^2$ for regression, and $\mathrm{Error}(\mathcal{D}, \Pi) := \sum_{n=1}^{N} \mathbb{I}(y^{(n)} \neq \hat{y}^{(n)})$ for classification. The BATrees is implemented also in Python. The depth of BATrees is chosen from $\{2, 3, 4, 6, 8, 10\}$ using 5-fold cross validation. For inTrees and Node Harvest, we used their R implementations with their default settings. For DTree2, we used `DecisionTreeRegressor` and `DecisionTreeClassifier` of scikit-learn in Python while fixing their depth to two. All experiments were conducted on 64-bit CentOS 6.7 with an Intel Xeon E5-2670 2.6GHz CPU and 512GB RAM.

[1] Due to the page limitation, only a few important results are highlighted in this section. The full results are reported in Appendix E. The exhaustive experimental results include the results on the computational scalability of the proposed sampling heuristic. The codes are available at `https://github.com/sato9hara/defragTrees`

Table 1: Average runtimes in seconds for one restart on three datasets: the EM algorithm ran over $K = 1, \ldots, 10$, and its total time is reported.

	Spambase	MiniBooNE	Higgs
FAB	5.32 ± 1.98	93.0 ± 49.2	54.3 ± 13.7
EM	35.7 ± 3.47	282 ± 60.1	227 ± 63.6

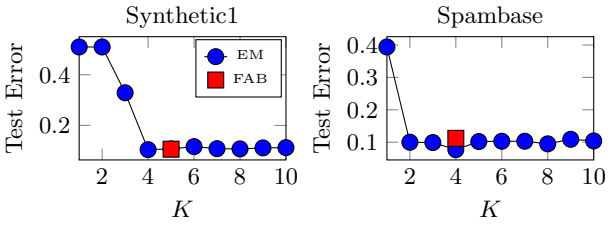


Table 2: Test Error of FAB inference and the EM algorithm on Synthetic1 and Spambase.

7.1 FAB Inference vs. EM Algorithm

We compared the runtimes of FAB inference and the EM algorithm. For the EM algorithm, we ran the algorithm by varying the value of K from 1 to 10, and reported the total runtime. For both methods, we used the sampling heuristic.

Table 1 shows that FAB inference was from three to nearly seven times faster than the EM algorithm. FAB inference attained smaller runtimes by avoiding searching over several possible number of rules K and deciding the number automatically. Figure 2 shows the comparison of the test errors of the found rules on Synthetic1 and Spambase: they show that FAB inference could find an appropriate number of rules K with small prediction errors. These results suggest the superiority of FAB inference over the EM algorithm as it could find an appropriate number of rules efficiently.

7.2 Comparison with the Baseline Methods

We compared the performance of the proposed method with the baseline methods with respect to the num-

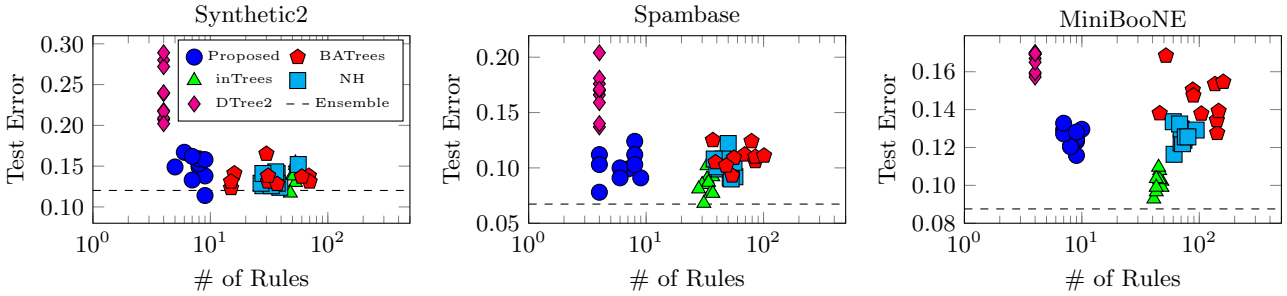


Figure 4: Comparison of the simplification methods: # of rules vs. test error. *Ensemble* denotes the average error of the tree ensemble over the ten random data realizations.

Table 3: Extracted rules using the proposed method on Energy data.

y	Rule
12.33	OverallHeight $<$ 5.25, WallArea $<$ 306.25
14.39	OverallHeight $<$ 5.25, WallArea $\geq$ 318.50
28.17	OverallHeight $\geq$ 5.25, WallArea $<$ 330.75
37.38	OverallHeight $\geq$ 5.25, WallArea $\geq$ 343.00

ber of found rules and the test errors. We conducted the experiment over ten random data realizations for each dataset. Figure 4 shows the trade-off between the number of found rules K and the test errors of each method on three datasets.

Number of Found Rules: Figure 4 shows that DTree2 tended to attain the smallest number of rules (i.e., four), and the proposed method was second (from four to ten). The number of rules found by inTrees and Node Harvest tended to be around 30 to 100, while the number of rules found by BATrees sometimes exceeded 100.

Test Errors: Figure 4 also shows inTrees and BATrees tended to attain the smallest test errors while DTree2 appeared to perform the worst on most of the datasets. The proposed method attained a good trade-off between the number of rules and the test errors: it tended to score smaller errors than DTree2 while using only a few rules, which is significantly smaller than the other baseline methods.

These results suggest that the proposed method is favorable for interpretation as it generates only a few rules with small errors. A smaller number of rules helps users to easily check the found rules. The small errors support that the found rules are reliable, i.e., the rules explain the original tree ensemble adequately.

7.3 Example of Found Rules

We show a rule example found in the energy efficiency dataset (Energy data). The energy efficiency dataset is a simulation dataset sampled from 12 different building shapes. The dataset comprises eight numeric features: Relative Compactness, Surface Area, Wall Area, Roof Area, Overall Height, Orientation, Glazing Area, and Glazing Area Distribution. The task is regression, which aims to predict the heating load of the building from these eight features.

Table 3 summarizes the four rules found by the proposed method, where the rules are characterized by the two features: Overall Height and Wall Area. The four rules are expressed as a direct product of the two statements; (i) Overall Height $\in \{\text{low}, \text{high}\}$, and (ii) Wall Area $\in \{\text{small}, \text{large}\}$. The resulting rules are intuitive such that the load is small when the building is small, while the load is large when the building is huge. Hence, from these simplified rules, we can infer that the tree ensemble is learned in accordance with our intuition about the data. In contrast to the simple rules found by the proposed method, the baseline methods found more rules: BATrees learned 66 rules, inTrees enumerated 23 rules, and Node Harvest found 10 rules, respectively.

8 Conclusion

We proposed a simplification method for tree ensembles to enable users to interpret the model. We formalized the simplification as a model selection problem to obtain the simplest representation that is essentially equivalent to the original one. To solve the problem, we derived a Bayesian model selection algorithm that automatically determines the model complexity. By using the proposed method, the complex ensemble is approximated with a simple model that is easy to interpret.

Acknowledgments

This work was supported by JST CREST Grant Number JPMJCR1666, Japan, JST ERATO Kawarabayashi Large Graph Project, Grant Number JPMJER1201, Japan and in part by NEDO.

References

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

[2] J. H. Friedman. Greedy function approximation: A gradient boosting machine. *Annals of statistics*, 29(5):1189–1232, 2001.

[3] T. Chen and C. Guestrin. Xgboost: A scalable tree boosting system. *Proceedings of the 22nd ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, pages 785–794, 2016.

[4] Kaggle. Your year on kaggle: Most memorable community stats from 2016, 2017.

[5] H. Akaike. A new look at the statistical model identification. *IEEE transactions on automatic control*, 19(6):716–723, 1974.

[6] G. Schwarz. Estimating the dimension of a model. *The annals of statistics*, 6(2):461–464, 1978.

[7] R. E. Kass and A. E. Raftery. Bayes factors. *Journal of the American Statistical Association*, 90:773–795, 1995.

[8] R. Fujimaki and S. Morinaga. Factorized asymptotic bayesian inference for mixture modeling. *Proceedings of the 15th International Conference on Artificial Intelligence and Statistics*, pages 400–408, 2012.

[9] K. Hayashi, S. Maeda, and R. Fujimaki. Rebuilding factorized information criterion: Asymptotically accurate marginal likelihood. *Proceedings of the 32nd International Conference on Machine Learning*, pages 1358–1366, 2015.

[10] B. Kim, D. M. Malioutov, and K. R. Varshney. Proceedings of the 2016 ICML workshop on human interpretability in machine learning. *arXiv preprint arXiv:1607.02531*, 2016.

[11] A. G. Wilson, B. Kim, and W. Herlands. Proceedings of NIPS 2016 workshop on interpretable machine learning for complex systems. *arXiv preprint arXiv:1611.09139*, 2016.

[12] M. T. Ribeiro, S. Singh, and C. Guestrin. Why Should I Trust You?: Explaining the predictions of any classifier. *Proceedings of the 22nd ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, pages 1135–1144, 2016.

[13] B. Kim, R. Khanna, and O. Koyejo. Examples are not enough, learn to criticize! Criticism for interpretability. *Advances In Neural Information Processing Systems*, pages 2280–2288, 2016.

[14] S. Kaufman, S. Rosset, C. Perlich, and O. Stitelman. Leakage in data mining: Formulation, detection, and avoidance. *ACM Transactions on Knowledge Discovery from Data*, 6(4):15, 2012.

[15] J. R. Lloyd and Z. Ghahramani. Statistical model criticism using kernel two sample tests. *Advances in Neural Information Processing Systems*, pages 829–837, 2015.

[16] B. Kim, J. Shah, and F. Doshi-Velez. Mind the gap: A generative approach to interpretable feature selection and extraction. *Advances in Neural Information Processing Systems*, pages 2260–2268, 2015.

[17] L. Breiman and N. Shang. Born again trees. *University of California, Berkeley, Berkeley, CA, Technical Report*, 1996.

[18] H. Deng. Interpreting tree ensembles with intrees. *arXiv preprint arXiv:1408.5456*, 2014.

[19] N. Meinshausen. Node harvest. *The Annals of Applied Statistics*, 4(4):2049–2072, 2010.

[20] Z.-H. Zhou and Y. Jiang. Nec4. 5: neural ensemble based c4. 5. *IEEE Transactions on Knowledge and Data Engineering*, 16(6):770–773, 2004.

[21] S. Ren, X. Cao, Y. Wei, and J. Sun. Global refinement of random forest. *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pages 723–730, 2015.

[22] F. Nan, J. Wang, and V. Saligrama. Pruning random forests for prediction on a budget. *Advances in Neural Information Processing Systems*, pages 2334–2342, 2016.

[23] M. Lichman. UCI machine learning repository, 2013.