EDIT: Interpreting Ensemble Models via Compact Soft Decision Trees

Jaemin Yoo
Seoul National University
Seoul, South Korea
jaeminyoo@snu.ac.kr

Lee Sael
Ajou University
Suwon, South Korea
sael@ajou.ac.kr

***Abstract*—Given feature-based data, how can we accurately classify individual input and interpret the result of it? Ensemble models are often the best choice in terms of accuracy when dealing with feature-based datasets. However, interpreting the decision made by the ensemble model for individual input seems intractable. On the other hand, decision trees, although being prone to overfit, are considered as the most interpretable in terms of being able to trace the decision process of individual input. In this work, we propose Ensemble to Distilled Tree (EDIT), a novel distilling method that generates compact soft decision trees from ensemble models. EDIT exploits the interpretability of a tree-based structure by removing redundant branches and learning sparse weights, while enhancing accuracy by distilling the knowledge of ensemble models such as random forests (RF). Our experiments on eight datasets show that EDIT reduces the number of parameters of an RF by 6.4 to 498.4 times with a minor loss of classification accuracy.**

***Index Terms*—interpretable learning, soft decision trees, random forests, knowledge distillation, weight pruning**

I. Introduction

Enabling interpretability, in addition to requirements for high accuracy, is currently one of the essential tasks in the machine learning and data mining community. Interpretable machine learning methods enable humans to understand the cause of a decision, allowing them to consistently predict the model's result [1, 2]. Interpretability is especially important for fields that require transparent and explainable predictions, such as medical domains and policy-making domains.

However, there exists a tension between accuracy and interpretability. Complex models such as deep neural networks and ensemble models are highly accurate while being difficult to interpret. On the other hand, intrinsically interpretable models are often simpler models that generally have low accuracies, such as linear models and decision trees.

How can we obtain predictions that are both accurate and interpretable? Deep neural networks (DNN) are one of the most studied models due to their high accuracy. There are several works that target making DNN models more interpretable, as surveyed in Montavon et al. [3]. However, DNN models are not always applicable as they require a large amount of training data and high computational power. Ensemble methods, on the other hand, often show the highest performances without such requirements. In fact, in an extensive study [4] of machine learning methods on small to midsize datasets, random forests (RF), which are representative ensemble models, had the best performances in 50 out of 121 datasets.

TABLE I: Comparison of tree-based models.

Method	DT	RF	SDT	C-SDT
Accuracy	Low	**High**	**High**	**High**
Model complexity	**Low**	High	Mid	**Low**
Interpretability	**High**	Low	Mid	**High**

Ensemble models are inherently complex and their interpretations are infeasible. Due to the limitation of DNNs and increasing interests in the requirements for interpretability, there have been several works on enhancing interpretability of ensemble models [5]–[7]. However, many of these models are limited to tree ensemble models, utilizing internal structures and similarities of the ensemble of trees to extract, prune, select, and summarize rules [7, 8].

In this work, we propose Ensemble to Distilled Tree (EDIT), an algorithm to distill the knowledge of an ensemble model to generate interpretable compact soft decision trees (C-SDT), our novel variants of soft decision trees (SDT) [9]. We focus on the problem of interpreting a model's decision process for a given input and show how an ensemble model can be distilled for better generalization accuracy of C-SDTs. The proposed C-SDT improves the interpretability of SDTs by allowing only a subset of input features to be involved in the decision process at each node and by simplifying the model structure by removing redundant branches.

Table I compares different tree-based models, including C-SDTs that are generated by our approach. Decision trees (DT) are one of the most interpretable models, but are easily overfit and produce high generalization errors. RFs solve such a problem and improve the accuracy significantly, but lose the interpretability of DTs due to their ensemble structures: all of their independent trees participate in every decision. An SDT is between the DT and RF; it is based on a single tree and thus more interpretable than the RF, but each internal node uses all features in each decision. Our proposed C-SDT is based on the structure of an SDT but learns a sparse weight vector for each node and removes redundant nodes in the tree structure. As a result, the complexity of a C-SDT is similar to that of a DT, but the classification accuracy is much higher.

II. Related Works

We first describe related works of EDIT, which are categorized into knowledge distillation and soft decision trees.

A. Knowledge Distillation

Knowledge distillation [10, 11] is a class of mimic learning approaches, which transfers the generalization ability of a cumbersome teacher model into a small student model by feeding the teacher's input and output to the student. Given a teacher network T and a student network S, one feeds training data to T and uses its predictions instead of the true labels to train S. As a result, S is trained by soft distributions rather than one-hot vectors, learning latent relationships between the labels that T has already learned. Knowledge distillation has been used for reducing the size of a model or training a model with insufficient data [12, 13]. It has also been applied for generating interpretable models from neural networks [9].

The idea of knowledge distillation is not restricted to neural networks. Any classifier can transfer its learned knowledge to a student model by giving its predictions as labels to train the student. In our work, we demonstrate how knowledge distillation can be applied to an ensemble model as a teacher model to generate a tree-structured student model.

B. Soft Decision Trees

Soft decision trees (SDT) [9,14] are tree-structured machine learning architectures that are based on soft decisions. The internal nodes of an SDT are linear classifiers of input features, and the leaves are values in bins where each bin is mapped to a class. Given a depth d as a hyperparameter, an SDT initializes a full binary tree of 2^d leaf nodes, where every internal and leaf node contains learnable parameters.

Each internal node computes the probability of passing an input feature $\mathbf{x}$ to the right or the left branch. This is different from a DT where a node chooses just a single branch. More specifically, given a feature vector $\mathbf{x}$, each internal node i computes the probability $p_i(\mathbf{x})$ of taking the right branch:

$$p_i(\mathbf{x}) = \sigma(\mathbf{x}^\top \mathbf{w}_i + b_i), \tag{1}$$

where $\mathbf{w}_i$ and b_i are learnable parameters of node i and σ is the logistic sigmoid function. Since an SDT is a binary tree, the probability of taking the left branch is $1-p_i(\mathbf{x})$. The input feature vector $\mathbf{x}$ is passed to the children in proportion to the probabilities computed at the current node.

Each leaf node i produces a probability distribution $\mathbf{q}_i$ that is learned over the training set:

$$\mathbf{q}_i = \mathrm{softmax}(\phi_i), \tag{2}$$

where ϕ_i is a parameter representing an unnormalized probability vector. This is also different from DTs because a leaf node can produce a soft distribution for multiple classes.

The prediction of $\mathbf{x}$ is performed differently for training and evaluation. For training, one calculates the weighted average of predictions of all leaves by the arrival probabilities:

$$\mathrm{sdt}(\mathbf{x}) = \sum_{n \in \mathcal{N}_d} r_n(\mathbf{x}) \cdot \mathbf{q}_n, \tag{3}$$

where $\mathcal{N}_d$ is the set of all leaf nodes, and r_n is the arrival probability for node n which sums to one if added for $\mathcal{N}_d$.

The arrival probability $r_i(\mathbf{x})$ of node i is computed as the multiplication of all path probabilities from the root:

$$r_i(\mathbf{x}) = \prod_{n \in \mathcal{P}_i} \mathbb{I}(\text{is\_left})(1 - p_n(\mathbf{x})) + \mathbb{I}(\text{is\_right}) p_n(\mathbf{x}), \tag{4}$$

where $\mathcal{P}_i$ is the set of nodes in the path between the root and node i, and $\mathbb{I}$ is an indicator function that produces one if the condition holds and zero otherwise.

The parameters of an SDT in all internal and leaf nodes are updated by a gradient-based approach. It aims to minimize the following loss function that is based on Equation (3):

$$l(\mathbf{x}) = \sum_{n \in \mathcal{N}_d} r_n(\mathbf{x}) \sum_{k \in \mathcal{Y}} y_k \log q_{nk}, \tag{5}$$

where $\mathcal{Y}$ is the set of possible labels, k is the index of a label, and y_k is the observed probability of $\mathbf{x}$ being categorized as k, which is either 1 or 0. This is a weighted cross-entropy that treats each leaf node as an independent classifier.

For the evaluation, only the most probable path is taken instead of all possible paths. In other words, an SDT picks the leaf node i with the maximum arrival probability $r_i(\mathbf{x})$ given a feature vector $\mathbf{x}$, and returns its softmax distribution $\mathbf{q}_i$ as the prediction for $\mathbf{x}$. The interpretation of the single most probable path is more straightforward than interpreting all possible paths for the prediction of each instance.

However, an SDT has limited interpretability even with its linear structure when the tree depth or the number of features is large. For instance, given a feature vector $\mathbf{x}$ of length $|\mathbf{x}|$, the explanation of a decision from an SDT of depth d involves $d(|\mathbf{x}|+1)+|\mathbf{y}|$ parameters, because it has passed through d linear classifiers with an additional leaf node. In this work, we resolve this issue by EDIT, our proposed approach that learns a compact SDT of pruned nodes and sparse weights.

III. Proposed Approach

We propose Ensemble to Distilled Tree (EDIT), a novel distilling method that learns compact soft decision trees (C-SDT) from a trained ensemble model. Our C-SDT is designed to enhance the interpretability of a vanilla soft decision tree (SDT) without decreasing its accuracy.

A. Overall Algorithm

Algorithm 1 describes EDIT, our proposed approach for training an interpretable C-SDT. EDIT first generates a new dataset $\mathcal{D}_d$ in line 1 by distilling the knowledge of a trained ensemble model M, which is used in the following training instead of the given dataset $\mathcal{D}_t$. EDIT initializes an SDT S in line 2 and applies whether the *weight masking* in line 3 or the *weight pruning* in line 11 to generate sparse weight vectors. EDIT updates S iteratively in lines 4 to 10, pruning redundant nodes in line 9 after consuming all training data at each epoch. EDIT returns the resulting C-SDT when the training is done, which is *compact* but produces high accuracy.

Although Algorithm 1 describes that S is updated at every instance, the actual implementation of EDIT is based on the batch learning: it divides the training data into small batches

Algorithm 1 Ensemble to Distilled Tree (EDIT)

Input: a pre-trained ensemble model M
Input: a training dataset $\mathcal{D}_t$ and a validation dataset $\mathcal{D}_v$
Input: a weight sparsity ratio γ and a pruning threshold ϵ
Output: a compact soft decision tree (C-SDT) S

1: $\mathcal{D}_d \leftarrow \{(\mathbf{x}, (M(\mathbf{x}) + \mathbf{y})/2) \mid (\mathbf{x}, \mathbf{y}) \in \mathcal{D}_t\}$ *// Sec. III-B*
2: $S \leftarrow$ initialize a soft decision tree (SDT)
3: $S \leftarrow$ weight masking with ratio γ *// optional; Sec. III-C*
4: **while** the convergence criterion is not met **do**
5: **for** $\mathbf{x}$, $\mathbf{y}$ in $\mathcal{D}_d$ **do**
6: $l(\mathbf{x}) \leftarrow -\log\left(\sum_{n \in \mathcal{N}_d} r_n(\mathbf{x}) \sum_{k \in \mathcal{Y}} y_k \log q_{nk}\right)$
7: $S \leftarrow$ update the parameters for decreasing $l(\mathbf{x})$
8: **end for**
9: $S \leftarrow$ prune nodes with threshold ϵ *// Sec. III-D*
10: **end while**
11: $S \leftarrow$ weight pruning with ratio γ *// optional; Sec. III-C*
12: **return** S

and updates the model for each mini-batch. We also use the Adam optimizer [15] for updating the parameters because it has been shown to work well with various machine learning models. We use the early stopping method as the convergence criterion in line 4; we stop the learning if the validation loss does not decrease for m consecutive epochs. We set m to 40 in our experiments, but its value can be different.

B. Knowledge Distillation

Given a pre-trained ensemble model M which performs well on given data, we use its predictions instead of the true labels in $\mathcal{D}_t$ when training our C-SDT model. We train the model by minimizing the following loss function:

$$l(\mathbf{x}) = -\log\left(\sum_{n \in \mathcal{N}_d} r_n(\mathbf{x}) \sum_{k \in \mathcal{Y}} M(\mathbf{x})_k \log q_{nk}\right), \quad (6)$$

where $M(\mathbf{x})$ is the output of M when $\mathbf{x}$ is given.

Although knowledge distillation improves the generalizability of the student model, it can be problematic if the teacher gives incorrect predictions. In other words, the student model may learn the wrong labeling. To alleviate the problem of the teacher feeding wrong information to the student, we use the average of a true label (one-hot vector) and a teacher-generated label (soft distribution vector). This is done by replacing the prediction $M(\mathbf{x})$ in Equation (6) as $(\mathbf{y} + M(\mathbf{x}))/2$, where $\mathbf{y}$ is the observed label for the feature $\mathbf{x}$. This approach is adopted form the original suggestion by Hinton et al. [11] for distilling the knowledge of a neural network.

C. Weight Sparsification

The vanilla SDT learns a linear classifier at each internal node that considers all input features, i.e., a vanilla SDT is a hierarchy of dense linear systems. It is well agreed that sparse linear systems are more interpretable compared with dense systems. Likewise, it is important to sparsify weights learned in each internal node for achieving high interpretability. We investigate three approaches for learning sparse weights.

1) L1 Regularization: A widely used approach for learning sparse weights is to add a sparse regularizer to the loss function by calculating the L1 norm of each weight. The loss function of a C-SDT with L1 regularization is as follows:

$$l_{\mathrm{L1}}(\cdot) = l(\cdot) + \lambda \sum_{i=0}^{d-1} \sum_{n \in \mathcal{N}_i} (\|\mathbf{w}_n\|_1 + |b_n|), \quad (7)$$

where i is the index of a layer, $\mathcal{N}_i$ is the set of nodes at the i-th layer, and $\mathbf{w}_n$ and b_n are the weight and bias of node n, respectively. Note that $i = 0$ represents the root node, and we do not include the leaves as it restricts the output space of the C-SDT, severely decreasing its learning capacity.

A benefit of this approach is that sparsity can be obtained naturally in a one-step process while optimizing for the loss function. However, the ratio of sparsification and the accuracy is highly dependent on the hyperparameter λ, which must be carefully tuned for each dataset. A wrong selection of λ can lead the model to either 1) little sparsification or 2) failure of minimizing the target loss at all. Moreover, the sparsity ratio cannot be controlled manually.

2) Weight Pruning: The second approach is weight pruning, which is to remove redundant weights from a trained model whose *significance scores* are smaller than a threshold [16]–[18]. The score can be calculated based on the contribution of the weight on the target loss [18] or simply be the absolute value of the weight [16, 17]. We use the absolute value of the weight as its score for simplicity.

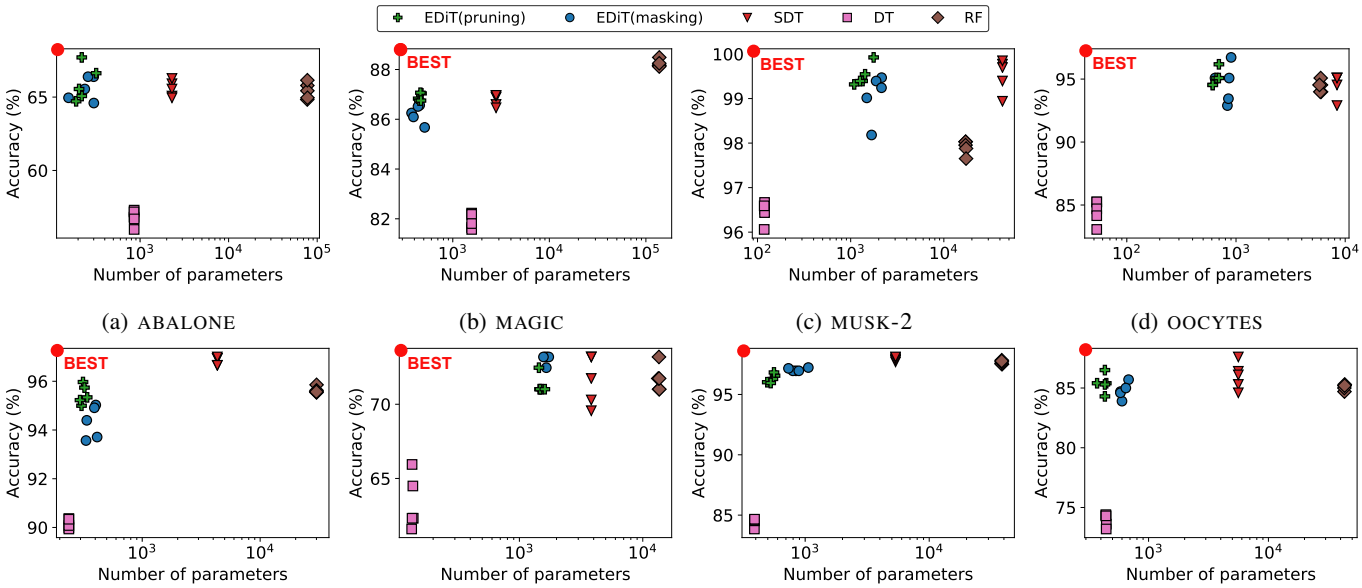
More specifically, we take a three-step approach to weight pruning. First, we train a model with dense weights. Second, we prune the weights by sorting the elements of each weight vector by their absolute values given a sparsity ratio γ. Thus, we maintain the top $\gamma|\mathbf{w}|$ elements of each weight vector $\mathbf{w}$. Finally, we fine-tune the pruned model with training data. A benefit of this approach is that the sparsity and accuracy can be balanced well by the value of γ, which we can control directly unlike the L1 regularization. However, the added fine-tuning step increases the complexity of training.

3) Weight Masking: The third approach is weight masking. We multiply the whole weight matrix of all internal nodes with a boolean mask generated randomly before the training begins. Thus, only the activated elements are used in further training and evaluation. This limits the performance of each node by preventing it from using all features but leads eventually to a robust tree that works well with fewer parameters.

Specifically, we generate the boolean mask by the element-wise Bernoulli distribution of probability γ, which represents the sparsity ratio as in the weight pruning. We do not mask the weights of leaf nodes because they represent final predictions. A benefit of this approach is that it is applied before the training begins and requires no separate fine-tuning. However, because the features are inactivated at random, the model is not able to learn a consistent decision process.

D. Tree Pruning

Even though tree structures are considered interpretable, in reality, it is difficult to interpret a large tree. For example, it



(a) ABALONE (b) MAGIC (c) MUSK-2 (d) OOCYTES

(e) PENDIGITS (f) STATLOG (g) TWONORM (h) WAVEFORM

Fig. 1: Classification accuracy of EDiT and the baselines with respect to the number of parameters. For each algorithm, we plot five results using different random seeds. EDiT(pruning) and EDiT(masking) are both located close to the best points at the upper left corners in all datasets, showing the best balance between the accuracy and complexity.

requires looking at eight different binary classifiers following the most probable path to interpret a prediction from an SDT of depth 8. Thus, it is desirable to decrease the number of nodes without affecting the classification accuracy.

Our *tree pruning* is to prune the branches at which only a few instances arrive during the training of S. We first initialize the weights of all internal nodes to zero, making S pass all instances equally to all leaf nodes. We then update the weights iteratively, increasing the bias for arrival probabilities to the leaf nodes as well as internal nodes. Given a pruning threshold ϵ, we iteratively prune the nodes where the average of arrival probabilities for the training data is less than ϵ, decreasing the number of active nodes at each iteration.

IV. Experimental Settings

We introduce our experimental settings including datasets, competitors, hyperparameters, and evaluation metrics. All our experiments were done at a workstation with Intel Xeon E5-2630 v4 and GTX 1080 Ti. Our compact soft decision trees (C-SDT) are implemented by the PyTorch [19] framework.

A. Datasets

We use eight datasets in the UCI Machine Learning Repository [20], which have various numbers of features, instances, and labels. The datasets are first downloaded form the USC website[1] that hosts preprocessed datasets from the repository [4, 21]. After that, we divide each dataset randomly into the 7:1:2 ratios for training, validation, and testing. Then, each dataset is z-transformed and scaled into zero-mean and unit-variance. Table II summarizes the datasets.

[1]http://persoal.citius.usc.es/manuel.fernandez.delgado/papers/jmlr/

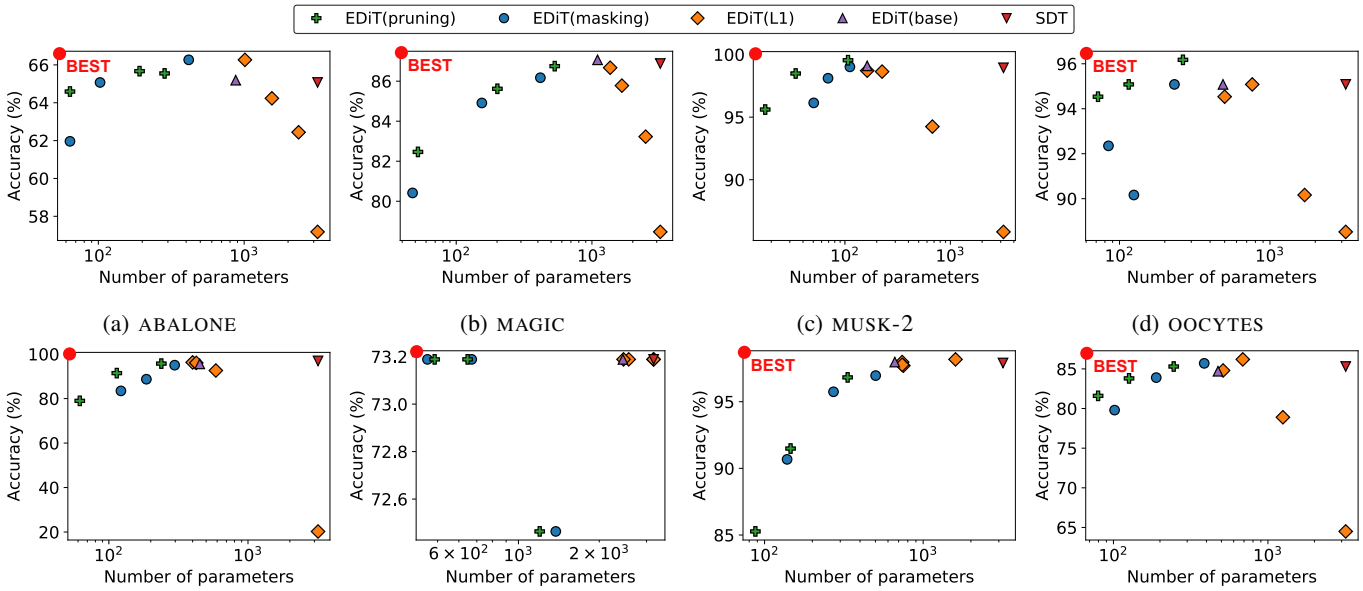
TABLE II: Summary of datasets.[1]

Dataset	Instances	Features	Labels
ABALONE	4,177	8	3
MAGIC	19,020	10	2
MUSK-2	6,598	166	2
OOCYTES	912	32	3
PENDIGITS	10,992	16	10
STATLOG	690	14	2
TWONORM	7,400	20	2
WAVEFORM	5,000	21	3

B. Competitors

We compare EDiT with various tree-based algorithms that have been used widely for classification problems. The first is a decision tree (DT) which supports high interpretability. However, a DT is known to overfit easily to training data and thus have a high generalization error. We use a random forest (RF) as a strong competitor in terms of accuracy, which is a representative ensemble approach whose performance is better than that of a DT but with loss of interpretability. Lastly, we use a soft decision tree (SDT) as our baseline method.

We use the implementations of *scikit-learn* [22] for DTs and RFs with the default hyperparameters; we have checked that changing them does not make a meaningful difference. One exception is the number of trees in an RF which is crucial for its classification performance and complexity. We set it to 100, because using more trees does not increase the accuracy of RFs in our datasets, while using fewer trees decreases it significantly. For SDTs, our PyTorch [19] implementation is used with the same hyperparameters as in C-SDTs.



(a) ABALONE (b) MAGIC (c) MUSK-2 (d) OOCYTES

(e) PENDIGITS (f) STATLOG (g) TWONORM (h) WAVEFORM

Fig. 2: Classification accuracy of EDIT with various approaches to generate sparse weight matrices, with respect to the number of parameters. EDIT(base) represents using only tree pruning and knowledge distillation without weight sparsification. EDIT with weight pruning and masking perform generally well in most cases, while the L1 approach performs the worst.

C. Experimental Setup

We use an RF as our teacher model for knowledge distillation, because it shows a good performance in many datasets. We set the depth d of all SDT-based models to 8, which was also used in [9]. We set the threshold ϵ for tree pruning to 10^{-3} and the weight sparsity ratio γ to 0.5 if not mentioned otherwise. For training, we use the Adam optimizer [15] and apply the early stopping; we stop the training if the validation loss does not decrease for 40 consecutive epochs.

Since we focus on developing an interpretable model, it is important to quantify the amount of interpretability. We count the numbers of parameters of all internal nodes and add them as an interpretability score of each model. We do not consider the leaf nodes because they represent the output of a model. For instance, each node in a DT represents one parameter as it considers a single feature element. The number of parameters of an RF is the sum of parameters for all its trees. Since an SDT-based model considers multiple parameters at each node, the number of parameters is likely to be larger than that of a DT if they have the same depth.

V. Experimental Results

In this section, we show experimental results of EDIT to answer the following questions:

- Q1) Does EDIT outperform the baselines in terms of classification accuracy and model complexity?
- Q2) Which sparsification approach is the most effective for generating sparse weight matrices?
- Q3) How many nodes does tree pruning remove? How is the accuracy affected by the tree pruning?

A. Accuracy and Complexity (Q1)

Figure 1 shows the classification accuracy of EDIT and the baseline approaches with respect to the number of parameters. We plot each method five times with different random seeds. The points from EDIT are closest to the best points at the upper left corners, especially when weight pruning is adopted, representing that they produce high accuracy even with few parameters. EDIT reduces the parameters of an SDT from 2.2 to 39.0 times, resulting in an efficient structure that contains up to 498.4 times fewer parameters than in an RF.

Interestingly, in two of the eight datasets, EDIT produces smaller models than even DTs. This is because DTs do not bound the depth of a tree: its depth is 27 and 38 in ABALONE and MAGIC, respectively, although the number of features in both datasets is at most 10. On the other hand, our C-SDT has a bounded depth, which is 8 in our experiments, and even removes most of those nodes by tree pruning. Thus, C-SDTs can be more lightweight than DTs even multiple features are considered at each internal node.

The number of parameters in an SDT increases significantly with the number of features since each internal node considers all features. SDTs are more complex than RFs in MUSK-2, where the number of features is very large. Weight sparsification is effective especially in this case, since most of those features are not used in the learned trees of a C-SDT.

B. Weight Sparsification (Q2)

We compare different approaches for weight sparsification in Figure 2. The first three methods represent EDIT with three different sparsification algorithms, and EDIT(base) represents

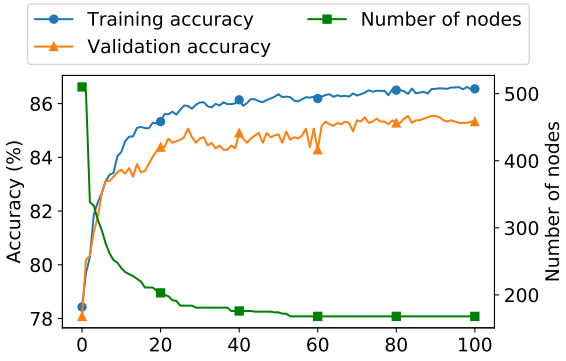


Fig. 3: Training process of EDIT for the MAGIC dataset. The accuracy for both training and validation data improve, while the number of active nodes is minimized.

using only tree pruning and knowledge distillation. We vary the sparsity ratio γ in $\{0.1, 0.2, 0.5\}$ for the weight tying and pruning approaches, and the regularization parameter λ in $\{10^{-5}, 10^{-4}, 10^{-3}, 10^{-2}\}$ for the L1-based approach.

As a result, weight masking and pruning generally perform well in most datasets, showing the smallest numbers of parameters in all cases. The accuracy drop is negligible when γ is set to 0.5, but it increases when smaller γ is adopted. It is shown that weight pruning reduces more parameters than weight masking does when $\gamma = 0.1$. This is because when weight pruning is adopted, we first train a model only with tree pruning and knowledge distillation and then sparsity the weights; the weight matrices are dense in the initial training and thus more branches can be removed by tree pruning.

Unlike the other approaches, the L1 approach has failed to reduce enough parameters even when large λ is adopted. This is because large λ prevents the model from searching enough space and limits its capacity for solving the classification, due to the regularizer added to the loss. On the other hand, weight masking and pruning have no such problems. Weight pruning gives the model a full chance of learning with dense weights before the sparsification, and weight masking does not restrict its search space once the boolean mask is determined.

C. Tree Pruning (Q3)

Since the tree pruning approach of EDIT removes active nodes during the training, it is necessary to check whether it affects the stability of the training or not. Figure 3 shows the training process of EDIT for the MAGIC dataset when weight masking is adopted for weight sparsification. The accuracy for both training and validation data improve while we iterate the training epochs, even though we remove many nodes; the half of all nodes are removed after epoch 8, and the $2/3$ after epoch 52. This is because we initialize all weights to zero; once a node has a small arrival probability, it is safe to prune it since it is redundant in terms of predictions.

VI. Conclusion

In this work, we propose Ensemble to Distilled Tree (EDIT), an accurate and interpretable distilling method for ensemble models that generates a compact soft decision tree (C-SDT). With EDIT, a vanilla soft decision tree (SDT) is improved in the perspective of interpretability and generalizability by our three techniques of a) knowledge distillation, b) weight sparsification, and c) tree pruning. Our experiments show that the resulting C-SDT trained by the knowledge of random forests (RF) shows comparable accuracy to both the RF and SDT, providing improved interpretability.

Acknowledgement

This work was supported by the National Research Foundation of Korea funded by the Ministry of Science, ICT and Future Planning (2018R1A5A1060031, 2018R1A1A3A0407953). Lee Sael is the corresponding author.

References

[1] B. Kim, R. Khanna, and O. Koyejo, "Examples are not enough, learn to criticize! criticism for interpretability," in *NIPS*, 2016, pp. 2288–2296.
[2] T. Miller, "Explanation in artificial intelligence: Insights from the social sciences," *Artificial Intelligence*, vol. 267, 2018.
[3] G. Montavon, W. Samek, and K.-r. Müller, "Methods for interpreting and understanding deep neural networks," *Digital Signal Processing*, vol. 73, pp. 1–15, 2018.
[4] M. F. Delgado, E. Cernadas, S. Barro, and D. G. Amorim, "Do we need hundreds of classifiers to solve real world classification problems?" *JMLR*, vol. 15, no. 1, pp. 3133–3181, 2014.
[5] G. Tolomei, F. Silvestri, A. Haines, and M. Lalmas, "Interpretable predictions of tree-based ensembles via actionable feature tweaking," in *KDD*, 2017.
[6] Y. Zhou, Z. Zhou, and G. Hooker, "Approximation Trees: Statistical Stability in Model Distillation," no. 2012, pp. 1–30, 2018.
[7] S. Hara and K. Hayashi, "Making tree ensembles interpretable: A bayesian model selection approach," in *AISTATS*, 2018.
[8] H. Deng, "Interpreting Tree Ensembles with inTrees," *International Journal of Data Science and Analytics*, vol. 7, no. 4, pp. 277–287, 2014.
[9] N. Frosst and G. E. Hinton, "Distilling a neural network into a soft decision tree," in *AI*IA 2017*, 2017.
[10] C. Bucila, R. Caruana, and A. Niculescu-Mizil, "Model compression," in *KDD*, 2006.
[11] G. E. Hinton, O. Vinyals, and J. Dean, "Distilling the knowledge in a neural network," *CoRR*, vol. abs/1503.02531, 2015.
[12] Y. Kim and A. M. Rush, "Sequence-level knowledge distillation," in *EMNLP*, 2016, pp. 1317–1327.
[13] A. Polino, R. Pascanu, and D. Alistarh, "Model compression via distillation and quantization," in *ICLR*, 2018.
[14] O. Irsoy, O. T. Yıldız, and E. Alpaydın, "A Soft Decision Tree," in *ICPR 2012*, 2012, pp. 1819–1822.
[15] D. P. Kingma and J. Ba, "Adam: A method for stochastic optimization," in *ICLR*, 2015.
[16] M. Zhu and S. Gupta, "To prune, or not to prune: Exploring the efficacy of pruning for model compression," in *ICLR Workshop*, 2018.
[17] T. Zhang, S. Ye, K. Zhang, J. Tang, W. Wen, M. Fardad, and Y. Wang, "A systematic DNN weight pruning framework using alternating direction method of multipliers," in *ECCV*, 2018, pp. 191–207.
[18] L. S. Moonjeong Park, Jun-Gi Jang, "VeST: Very Sparse Tucker Factorization of Large-Scale Tensors," in *arXiv:1904.02603*, 2019.
[19] A. Paszke, S. Gross, S. Chintala, G. Chanan, E. Yang, Z. DeVito, Z. Lin, A. Desmaison, L. Antiga, and A. Lerer, "Automatic differentiation in pytorch," in *NIPS (Workshop)*, 2017.
[20] D. Dua and C. Graff, "UCI machine learning repository," 2017. [Online]. Available: http://archive.ics.uci.edu/ml
[21] M. Olson, A. J. Wyner, and R. Berk, "Modern neural networks generalize on small data sets," in *NeurIPS*, 2018, pp. 3623–3632.
[22] F. Pedregosa, G. Varoquaux, A. Gramfort, V. Michel, B. Thirion, O. Grisel, M. Blondel, P. Prettenhofer, R. Weiss, V. Dubourg, J. Vanderplas, A. Passos, D. Cournapeau, M. Brucher, M. Perrot, and E. Duchesnay, "Scikit-learn: Machine learning in python," *Journal of Machine Learning Research*, vol. 12, pp. 2825–2830, 2011.