Received March 15, 2021, accepted March 30, 2021, date of publication April 2, 2021, date of current version April 16, 2021.

Digital Object Identifier 10.1109/ACCESS.2021.3070575

SDTR: Soft Decision Tree Regressor for Tabular Data

HAORAN LUO, FAN CHENG, (Member, IEEE),
HENG YU, (Graduate Student Member, IEEE), AND YUQI YI, (Student Member, IEEE)
MoE Key Laboratory of Artificial Intelligence, Department of Computer Science and Engineering, AI Institute, Shanghai Jiao Tong University, Shanghai 200240, China

Corresponding author: Fan Cheng (chengfan@sjtu.edu.cn)

This work was supported in part by NSFC under Project 61701304, in part by the Shanghai Sailing Program under Grant 17YF1410100, and in part by the Program for Professor of Special Appointment (Eastern Scholar) at Shanghai Institutions of Higher Learning.

ABSTRACT Deep neural networks have been proved a success in multiple fields. However, researchers still favor traditional approaches to obtain more interpretable models, such as Bayesian methods and decision trees when processing heterogeneous tabular data. Such models are hard to differentiate, thus inconvenient to be integrated into end-to-end settings. On the other hand, traditional neural networks are differentiable but perform poorly on tabular data. We propose a hierarchical differentiable neural regression model, Soft Decision Tree Regressor (SDTR). SDTR imitates a binary decision tree by a differentiable neural network and is plausible for ensemble schemes like bagging and boosting. The SDTR method was evaluated on multiple tabular-based regression tasks (YearPredictionMSD, MSLR-Web10K, Yahoo LETOR, SARCOS and Wine quality). Its performance is comparable with non-differentiable models (gradient boosting decision trees) and better than uninterpretable models (regular FCNN). On top of that, it can produce fair results with a restricted number of parameters, only using a small forest or even a single tree. We also propose an ''average entropy'' metric to evaluate the level of interpretability of a trained, soft decision tree neural network. This metric also helps to select proper structure and hyperparameters for such networks.

INDEX TERMS Decision trees, neural networks, machine learning algorithms.

I. INTRODUCTION

Regression refers to a category of supervised learning, whose output is continuous other than a limited set of values. In the past decades, many algorithms were proposed to perform regression tasks. Roughly speaking, we can divide them into two categories, and each category has its pros and cons.

- Parametric models. This category ranges from the simplest Linear Regression models [26] to the most complicated deep neural networks. Most of them can be optimized via backpropagation, thus can be easily integrated as a component in complex, end-to-end pipelines, and applicable for a wide field of problems. However, there is an implicit tradeoff between the model's accuracy and the ease of interpretation. A simple function, for example, linear regression or the Cox model [42] for survival analysis, is easy to be interpreted; while in real-world settings, a proper approximation function can be hard to find. In most cases, such a function might not exist in solution spaces of linear regressors or other straight-forward models. When dealing with a large dataset, as in many real-world regression tasks, a complicated neural network can be trained to obtain a close approximation of the ground-truth results [39]. However, it can hardly be interpreted due to its multi-layer nature and complex connections.
- Non-parametric models. This type of regression model tries to make assumptions about target distribution given the patterns observed from input samples, other than making assumptions beforehand. Kernel regression [27] and gradient boosting tree-based methods (which are quite popular these years) are members of this category. It is relatively easier to interpret non-parametric models: such a model generally tends to ''group'' similar examples into the same cluster or assign them the same branch on a tree node. This behavior naturally reveals how the model works. However, such non-parametric models are harder to combine with other gradient-based methods.

The associate editor coordinating the review of this manuscript and approving it for publication was F. K. Wang.

As is proposed by Frosst and Hinton [13], using deep neural nets to mimic the branching structure of decision trees helps us to explain the neural network model better, thus inheriting the merits of both parametric and non-parametric models. The proposed Soft Decision Tree (SDT) model performs slightly worse than traditional convolutional neural networks in MNIST hand-written digits classification [22] task. However, the model itself shows a clear relationship among different classes in a hierarchical fashion. It automatically groups some similar classes (such as digits 5 and 6) by assigning a common parent for those classes without explicit supervision to force the model to do so.

However, the potential of SDT on heterogeneous tabular data and regression tasks was left unnoticed, as [13] only examines the performance of SDT on homogeneous image classification datasets (e.g., MNIST). We adopted this idea and developed the Soft Decision Tree Regressor (SDTR) for regression tasks and heterogeneous tabular data.

In this paper, we enumerate our main contributions as the following:

- We proposed SDTR, a lightweight differentiable soft decision-tree based neural regression model.
- Based on observations of single-tree settings of SDTR, we developed several techniques for improving the accuracy of soft decision trees.
- We tested the performance of SDTR on different scales, including single-tree and ensembles on both bagging and boosting. We compared its performance against traditional FCNN, state-of-the-art decision tree-based NN methods NODE [34], non-hierarchical NN TabNet [2] and several non-differentiable gradient decision tree methods (XGBoost [9] and DeepForest [49]). Among those models, SDTR is competitive for regression tasks on tabular data, and it achieves comparable performance with NODE and TabNet using fewer parameters.
- We proposed the exponentially decaying L_1 regularization to encourage the sparsity of weight matrix in SDT and SDTR, making its interpretability level closer to GBDT. We also introduced the "average entropy" metric to evaluate the interpretability of SDTR and other soft decision tree models. This metric can be used to select proper structure and hyperparameters.

II. RELATED WORK

A. DECISION TREES

Decision Trees have been a common approach to regression problems. As proposed in [6], if one defines a node's predicting value as the constant prediction and the node's impurity as the sum of squared deviations about its samples' mean, the decision tree will become a regression model.

In the past decades, this field's primary research focus was based on the gradient boosting decision tree method (GBDT) proposed in [12]. There are multiple open-source packages that implement the GBDT algorithm (for both classification tasks and regression tasks), for example scikit-learn [33], gbm [36], XGBoost [9] LightGBM [19] and CatBoost [50]. While the core idea was left unchanged, these packages mainly focused on speed up, parallelization, large-scale datasets handling, and robust training.

B. SOFT DECISION TREES

Recently, Deep Neural Networks (DNNs) achieved great success in fields like Computer Graphics [16], Natural Language Processing [44], speech recognition [15], and reinforcement learning [25]. However, DNNs have met obstacles in processing heterogeneous tabular data, unable to outperform prevailing traditional models consistently.

However, traditional models are generally not differentiable, thus unable to be integrated as components in pipelines. Arik and Pfister (2019) [2] proposed a new canonical DNN architecture TabNet, which outperforms or is on par with traditional tabular learning models.

Another line of research tries to imitate the traditional learners by neural networks. Inspired by decision trees, which are proved to be capable of processing heterogeneous data, researchers have developed various sorts of differentiable, or "soft" decision trees/forests. Soft decision trees were first introduced by Suarez and Lutsko [40]. They performed a "fuzzification" process over a trained CART decision tree skeleton, replacing the hard threshold at each non-leaf node with sigmoid functions. The fuzzy model was treated as a feed-forward network and thus can be trained via backpropagation.

In recent years, soft decision trees (SDT) have been an active field of research again. Léon and Denoyer [23] proposed a computationally efficient backpropagation scheme to directly optimize the hard partitioning function at each node. Frosst and Hinton [13] proposed a regularization method to encourage a balanced split at each internal node, improving the robustness of SDT.

While traditional SDT uses a single feedforward layer with sigmoidal activation function as their decision function, modern SDTs may choose various decision functions to resolve different problems. Bulo and Kontschieder [37] proposed a tree-shaped neural network with randomized MLP decision function to solve semantic image labelling, and ensembled multiple such networks to make a "decision forest". Yang *et al.* [47] took soft binning function into account, using a neural network to faithfully imitate the "splitting" choice made at each node split. Popov *et al.* [34] uses a similar technique, but instead of a full binary tree, they tried to implement an oblivious decision tree for faster training/inference. Tanno *et al.* [41] took a more adaptive approach. In their work, each edge of the tree would further stand for a transformation function (often residual).

C. ENSEMBLE LEARNING AND TREE ENSEMBLE

In real tasks where the features are highly entangled, the performance of a single tree is limited. The conclusion holds for both traditional decision trees and their soft counterparts. A widely accepted workaround is to use more trees, then aggregate the results together. This method falls into the ensemble learning category.

Ensemble learning is a well-developed direction of machine learning research. It has been utilized in various fields and applications, e.g. stock returns prediction [43], credit scoring [1], sentiment analysis [31] and text classification [30]. In these works, decision tree often serves as a base learner, and its result is often aggregated with other base learners such as Naïve Bayes, nearest neighbor classifiers [29] and support vector machines.

Modern decision trees also use ensemble methods to increase the model's capacity. In this scope, all base learners are decision trees. Different decision trees would deal with different inputs or in charge of different positions in a pipeline. Breiman [5] proposed the random forest algorithm, organizing decision trees by bagging on random subsets of examples and features. Friedman [12] proposed the gradient-boosted trees (GBT) method which yielded state-of-the-art performance in many fields including research institutions ranking [38], recommender systems [46], bioinformatics [8] and medical applications [21]. Zhou and Feng (2017) [49] proposed the Deep Forest approach to generate a cascade ensemble of forests, which outperforms traditional tree ensembles in various domains.

Soft decision trees can also be ensembled via a similar scheme. Kumar *et al.* [20] proposed an ensemble of soft decision trees for robust classification; Yıldız *et al.* [48] tested the bagged soft decision trees on two-class classification datasets and regression datasets. Some previously mentioned neural SDTs also utilized ensemble learning. Tanno *et al.* [41] trained multiple models and took the average of outputs as the final result, and observed a performance gain; Popov *et al.* [34] stacked the oblivious trees, feeding the output of previous trees into following ones, and improved the performance at the cost of training time.

III. METHODOLOGY

In this section, we describe the algorithm and techniques used in SDTR.

A. THE HIERARCHICAL MIXTURE OF CONSTANT PREDICTIONS

The main idea of SDTR follows the "hierarchical mixture of bigots" setting proposed by Léon *et al.* [23] and Frosst *et al.* [13]. Unlike [47] and [34], each bigot draws its conclusion on all feature vectors through the decision function of its own, rather than selecting a splitting point for a particular feature.

In the binary decision tree model, decisions are made by a series of hierarchical nodes. Each node can be viewed as a binary classifier: for an incoming sample, the node would decide which branch should further handle the sample.

Similarly, the SDTR structure is a full binary tree. Each node i in this tree represents a binary classifier, with learnable parameters w_i, b_i. Given a certain input, the classifier's output is calculated by a sigmoid function representing the probability of choosing the left branch:

$$p_i(\mathbf{x}) = \sigma(\mathbf{w_i x} + b_i). \tag{1}$$

To prevent too soft decisions, we multiply a β_i on the term $\mathbf{w_i x} + b_i$, before calculate the sigmoid. Each β_i is initialized as a hyperparameter (in all our experiments, we choose 1.5), and different β_is are independently trainable. Thus the output would be soft again at some nodes when needed.

Note that the choice function $p_i(x)$ is not necessarily a linear function $\mathbf{w_i x} + b_i$. Any function can be used to make the decision, as long as its output scope is [0,1], and thus we can interpret the result as "probability". Multi-layered networks or convolutional networks (for image processing) are also feasible choices. Given the probability of choosing the left branch, the probability of choosing the right branch should be $1 - p_i(\mathbf{x})$.

All nodes naturally form a hierarchical mixture of experts (HME) [18]. Each leaf node corresponds to a **scalar** R_ℓ, which serves as the model's prediction. The label y is normalized by $y = \frac{y' - \bar{y'}}{\sigma}$, where y' is the original label and $(\bar{y'}, \sigma)$ are the mean and standard error of y' in the training set. R_ℓ is initialized via sampling from a standard normal distribution. Consider each leaf, the probability of choosing the leaf equals the multiplication of probabilities of a series of nodes that lie along the path from the root to leaf:

$$P_\ell(x) = \prod_{i \in \mathrm{Path}(\ell)} p_i(x)^{l_i} (1 - p_i(x))^{1 - l_i}, \tag{2}$$

where l_i stands for the routing (whether the next node is the left child of i) on $Path(\ell)$.

Unlike traditional neural networks where the output of previous layer serves as the input into following layer, all nodes in SDTR use the same input x from dataset. This structure itself mitigates vanishing gradient problem, making it possible to stack multiple sigmoid functions sequentially while preserving the strength of gradient descent.

We illustrate a 2-layer SDTR tree in figure 1.

The objective function seeks to minimize the MSE (mean squared error) between each leaf's prediction R_ℓ and the target ground-truth value y, or to minimize other reasonable differentiable objective functions. Take MSE as an example. The objective function at each leaf is weighted by its path probability:

$$L_{MSE}(x) = \sum_{\ell \in \mathrm{LeafNodes}} P_\ell(x)(R_\ell - y)^2. \tag{3}$$

There are two ways to interpret the model's output to obtain the prediction $\hat{y}$.

- Take the value on the leaf with the largest probability as the prediction.

$$\begin{aligned} \ell^* &= \arg\max_{l} P_\ell(x) \\ \hat{y} &= R^*_\ell \end{aligned}$$

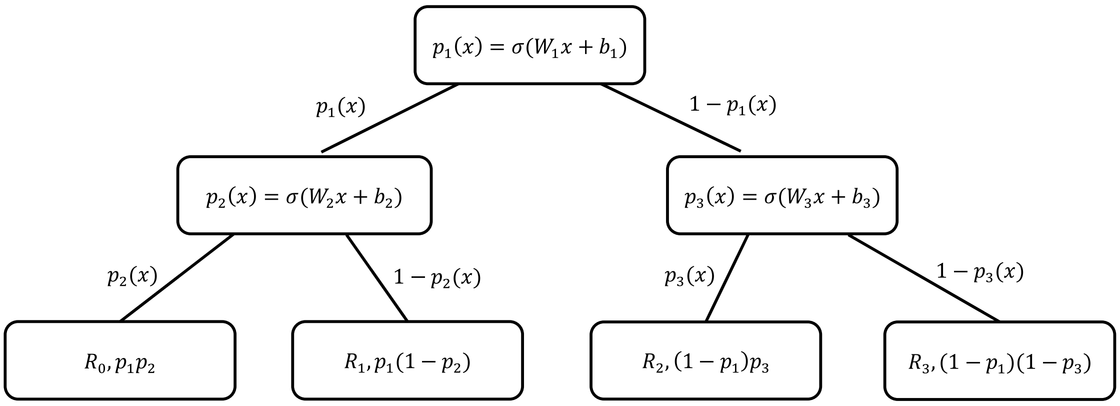

$p_1(x) = \sigma(W_1 x + b_1)$
$p_1(x)$
$1 - p_1(x)$
$p_2(x) = \sigma(W_2 x + b_2)$
$p_3(x) = \sigma(W_3 x + b_3)$
$p_2(x)$
$1 - p_2(x)$
$p_3(x)$
$1 - p_3(x)$
$R_0, p_1 p_2$
$R_1, p_1(1 - p_2)$
$R_2, (1 - p_1) p_3$
$R_3, (1 - p_1)(1 - p_3)$

FIGURE 1. A shallow SDTR with depth $d = 2$. x stands for the normalized input feature all nodes use the same x from data), and we use a linear function followed by a sigmoid to generate the decision on each non-leaf node. Note that at each decisional node, we can use an arbitrarily differentiable function, as long as the output vector sums up to 1.

- Sum up the predictions on all leaves and weight them by corresponding path probabilities.

$$\hat{y} = \sum_{\ell \in \text{LeafNodes}} P_\ell(x) R_\ell$$

In our experiments, the second interpretation performs better in most settings, which means the conclusion of [13] still holds for regression tasks. In the experiment section, we only use the second interpretation.

B. REGULARIZATION

As in [13], we used a "path penalty" term that encourages a balanced tree classifier. This penalty causes each internal node to equally use both left and right sub-trees, thus avoiding getting stuck on too biased weights (the case that classifier assigns most examples to one leaf node).

We compute the penalty term for each node. Mathematically speaking, this penalty term is the cross-entropy between the ideal "balanced" distribution [0.5,0.5] and the actual distribution. The latter is sampled per mini-batch.

The actual distribution $[\alpha_i, 1 - \alpha_i]$ for node i is defined by:

$$\alpha_i = \frac{\sum_x P^i(x) p_i(x)}{\sum_x P^i(x)} \tag{4}$$

where $P^i(x)$ is the accumulated path probability of feature vector x from root to node i. Then, we compute a weighted sum of the cross entropy obtained from each non-leaf node (inner node) to get the final path penalty term:

$$L_{pp} = - \sum_{i \in \text{InnerNodes}} 2^{-d_i} [\frac{1}{2} \log(\alpha_i) + \frac{1}{2} \log(1 - \alpha_i)] \tag{5}$$

d denotes the depth of node i. As mentioned in [13], a deeper node is more possible to handle a non-equal split. Its weight in L_{pp} should be decayed; otherwise, such path penalty can be harmful to the model's accuracy. In our early experiments, we also observed that exponentially decaying the weight according to the node's depth d achieved better results.

Besides that, we also regularize each node's weight matrix by L_1 loss. On the one hand, this loss binds the output in the sigmoid function's near-center region, thus helping gradient flow. On the other hand, this regularization term helps the model generate more interpretable results.

Intuitively, a decision tree's interpretability relies on two facts: it uses hierarchical structure, and the decision function at each node is a threshold function on a particular feature. The ensembles of decision trees (both bagging and boosting) are typically regarded as black-box models. However, their interpretability is still better than that of regular deep neural networks. For example, it is possible to compute a feature importance ranking in a trained GBDT. Friedman [12] proposed a feature importance measure based on the relative influence of individual input feature. The importance of feature j on tree T can be approximated by the resulting improvement in squared error loss $\hat{\ell}$, summed up for each inner node t:

$$\hat{I}_j^2(T) = \sum_{t \in \text{InnerNodes}} \hat{\ell}_t^2 1(v_t = j) \tag{6}$$

$1(v_t = j)$ denotes whether the node's splitting feature is feature j.

It is theoretically possible to implement the same approach in SDTR, making its interpretability better than traditional NNs. However, the "split gain"(ℓ_t) at each node t is hard to compute. By GBDT's greedy splitting policy, we can reach an intuitive conclusion that the splitting feature used at the root or top levels of the tree is more important than those used at the bottom, near-leaf levels. Since we have utilized the path penalty term to balance the distribution at each tree classifier, we assume that the near-root nodes, and their corresponding decision functions, are more important in the overall decision process.

Another difficulty lies in the decision function. As we have replaced the hard threshold function with linear sigmoid, we cannot directly use the term $1(v_t = j)$. However, we still hope our decision functions to be sparse; thus, the most important feature(s) at a certain node would be clearer.

By introducing the L_1 regularization loss, we encourage the sparsity at each decisional node. Based on previous observations on L_{pp} and $\hat{I}_j^2(T)$, we have chosen that the weight of L_1 loss at node t should be reweighted in an exponentially decaying fashion, according to the node's depth d_t. Let $w^{(t)}$ be our $1 \times K$ weight matrix associated with an inner node t, the final expression of our regularization loss L_{reg} is:

$$L_{reg} = \sum_{t \in \text{InnerNodes}} 2^{-d_t} \sum_{k=0}^{K} |w_k^{(t)}| \tag{7}$$

By reweighting the L_1 loss, we keep the sum of L_{reg} of each layer at the same magnitude. This regularization approach also controls each decision node's input norm (sigmoid function), keeping the function value near zero, where the gradient descent is effective. Like Batch Normalization [17], this approach helps to effectively prevent the occurrence of over-saturated decision nodes, and avoid the risk of gradient vanishing. Our experiments show that this regularization term brings notable performance gain.

Both path penalty term and the L_1 regularization loss are integrated in the MSE loss function. Our final optimization target is the sum over these three expressions:

$$L = L_{MSE} + \lambda_1 L_{pp} + \lambda_2 L_{reg} \tag{8}$$

The strength of L_{pp} and L_{reg} are controlled by two hyper-parameters λ_1 and λ_2.

C. IMPROVING SDTR: SINGLE TREE

A single regression tree is weak. In a single-tree setting, since our predictions can only be made by R_ℓ at each leaf node, the number of leaf nodes will influence the minimum MSE (Mean Squared Error) that we can obtain.

In order to achieve a better result in the single-tree setting, our tree needs to go deeper. However, merely increasing the depth of the tree leads to a less robust model, and such model requires more steps to converge.

We observed that main obstacle in training deeper SDTR lies in the gradient of the leaf nodes' weights. Consider updating a single leaf node ℓ's response R_ℓ, by taking derivatives on (3), we can obtain:

$$\frac{\partial L_{MSE}}{\partial R_\ell} = 2(R_\ell - y)P_\ell(x) \tag{9}$$

Since $\sum_\ell P_\ell = 1$, as the model's depth increases, the gradient's norm on each leaf node's weight shrinks exponentially, thus causes slow convergence in the training phase. We attempted to resolve this problem by re-weighting the learning rate of R_ℓ.

We manually multiply the learning rate of R_ℓ by the total number of leaf nodes (2^{d-1}). Note that directly increasing the overall learning rate only results in a failing model: previous decision nodes would become unstable. So we only adjust the learning rate associated with each R_ℓ.

Besides toggling the learning rate, we tried to use hidden layers (fully connected layer with ReLU [28] activation) to enhance a single tree's expressive power. Before feeding input features into each decisional node, we first transform them by a fully-connected layer. All decisional nodes share the same FC layer. However, such a method introduces extra parameters, and we have not seen apparent improvement in our experiments. One possible reason is that the hierarchical structure itself is sufficient for representing higher-order logic, making the preceding FC layer unnecessary.

D. IMPROVING SDTR: ENSEMBLE

To resolve the limited accuracy of a single tree, Friedman *et al.* [12] and many following papers used the "ensemble" technique. Generally speaking, there are two ways to ensemble models: one is to take the average output of multiple different models, or the so-called "voting" mechanic; the other is called "boosting". The "boosting tree" structure refers to a sequence of decision trees. The first tree aims to optimize the target value directly; the following trees aim to compensate for the accumulated error made by all previous trees.

The output of SDTR is a weighted average of all leaf nodes. If we view each leaf node as a base learner (which returns a constant), then a single SDTR tree is already a bagging model. Each base learner would deal with a specific division of data; the path probabilities would determine the division, and the division itself is updated through backpropagation. However, this bagging scheme is limited by the single tree structure. To alleviate the problem, we initialized a forest of trees differently, then average their outputs. Such a forest can produce a prediction directly or serves as a "layer" in our boosting scheme.

Despite being successful, the boosting method is harder to implement in an end-to-end training environment. This technique implies that some part of the whole network will be repeatedly trained while the other parts are frozen, introducing considerable overheads. Popov *et al.* [34] proposed a workaround for end-to-end setting: each tree would have multi-dimensional outputs (rather than a scalar), a tree located at successor layers can treat outputs of all its previous trees as additional input features. The final output is then given by averaging across the **first** dimension of all tree's outputs. The method improves accuracy but significantly increases the number of model parameters and time consumption: for trees in successor layers, the dimension of input features increases by $O(l^2)$, thus prohibiting the model from getting deeper.

To alleviate the problem, we made a single tree more expressive than the oblivious decision trees implemented in [34]. We will further justify this in the experiment section. As the complexity of a single tree increases, we can achieve comparable results by lesser boosting layers or using fewer trees in each layer.

IV. EXPERIMENTS

In this section, we compare the result among our approach, FCNN, state-of-the-art neural decision tree method NODE [34], state-of-the-art neural network for tabular data TabNet

[2], and leading GBDT packages including XGBoost [9], LightGBM [19], CatBoost, and DeepForest [49]. For SDTR, we provide results on shallow models (SDTR forest without boosting) and "deeper" ones.

A. OVERVIEW

We built the SDTR model via PyTorch python package [32].[1] All experiments were performed on a single GTX 1080Ti. We test the performance of SDTR on each dataset under three different conditions.

- Single-tree mode: In this mode, only **one** tree was used. We compare it against a fully connected neural network (FCNN) and a small ensemble of oblivious trees (NODE) [34]. The aim of this part is to show that a single SDTR model has comparable expressive power as FCNN and slightly stronger than its oblivious counterparts. We keep the search space of hyperparameters of SDTR identical across all experiments.
- Single-layer mode: In this mode, we constructed multiple trees and integrated them as a forest. We averaged all trees' output as the model's output and used it for both the training and inference phases. The single-layer approach is suitable for parallelization, thus would be a useful metric in practice. The number of trees across all five datasets are $n = 256$.
- Multi-layer boosting mode: In this mode, we arrange multiple trees in "layers" as previously described in section III-D. One layer consists of an SDTR forest, and the output of such forest serves as additional input features for its following layers. This approach tests the best possible performance of a certain model on tabular data. We compare the performance of SDTR with other state-of-the-art models (XGBoost, LightGBM, CatBoost, NODE, TabNet and DeepForest).

Since our regularization term is computed per-batch, some may argue that a bigger batch size would help train a more balanced SDTR tree. We provide ablation results in section V on batch size for single-tree and single-layer setups, keeping all the other hyperparameters fixed; based on the observation, for multi-layer boosting mode, we keep the batch size as 1024.

In all our experiments with SDTR, we stopped training when the validation MSE does not improve for 1000 consecutive batches.

Other models. We also provide results of FCNN, XGBoost [9], LightGBM [19], CatBoost [50], NODE [34], DeepForest [49] and TabNet [2], trained with the same dataset. For FCNN and NODE, we provide the results of both shallow and deep variations. We used the "Dense-LeakyReLU-Dropout" structure as the basic building block of FCNN. The shallow version consists of 2 layers, while the deep version consists of 7. We keep the search space of hyperparameters the same in the same experiment group (Small, Shallow, and Deep). The shallow NODE is built as a forest of $n = 2048$ oblivious trees, while the deep counterpart used the previously mentioned "boosting" technique to stack forests together. As the number of trainable parameters in one single tree of NODE is approximately one magnitude smaller than in SDTR, we also provide a smaller single-layer version of NODE (NODE-small) with the same hyperparameters as NODE-shallow but consists of $n = 10$ trees.

[1] Source code will be available on GitHub soon.

B. DATASETS

To evaluate the performance of proposed SDTR model, we performed regression tasks on five open-source regression datasets: YearPredictionMSD [11], MSLR-WEB10K [35], Yahoo LETOR [7], SARCOS [45] and WINE quality [10]. For the first three datasets, we used the same train-val split for all experiments on different hyperparameters and models, the same as what NODE [34] used in their paper. For SARCOS, we used 20% samples from the train split as a validation set and used the provided test split. For WINE, which did not provide a test split, we manually split the dataset by 8:1:1. The val split is used for hyperparameter tuning and early stopping. Then, we evaluate each derived model on the test split defined by dataset providers (if available). The train/val/test split is fixed for different models.

Here we provide a brief description for each dataset.

YearPredictionMSD (Year): A dataset derived from MillionSong dataset [4]. The features are extracted from the "timbre" features generated by the Echo Nest API. The target is the release year of a certain song, an integer ranging from 1922 to 2011.

MSLR-WEB10K (MSLR): Microsoft Learning-to-rank dataset (MSLR) consists of 136-dimensional feature vectors extracted from query-url pairs, and its label take 5 values from 0 (irrelevant) to 4 (perfectly relevant).

Yahoo LETOR (Yahoo): Similar to MSLR-WEB10K, Yahoo LETOR is a learning-to-rank dataset with query-URL pairs, labeled by integers from 0 to 4. We treat both ranking datasets as regression tasks, using feature vectors to predict the (real-valued) label directly.

SARCOS: The regression task is to solve an inverse dynamics problem for a SARCOS anthropomorphic robot arm. We map the 21 input variables to the first of seven joint torques, transforming the multi-regression problem in [2] into a single one.

WINE quality (WINE): The dataset aims to model red wine quality based on physicochemical tests. Each instance corresponds to a red Vinho Verde wine sample. Input variables are physicochemical properties, for example, density, pH, and fixed acidity. Instances are labelled by a "quality score" (between 0 and 10) based on sensory data.

Table 1 shows details of datasets. They differ in both scales and the number of input features. For the last two datasets, it is more likely to overfit in our training process. The results on SARCOS and WINE can be used to test the models' robustness.

TABLE 1. Details of 3 tabular datasets involved in experiment.

	Train+Val	Test	#Features
YearPredictionMSD	463K	51.6K	90
MSLR-WEB10K	723K	241K	136
Yahoo LETOR	544K	165K	699
SARCOS	44K	4449	21
Wine quality	1441	160	12

C. TRAINING

Here we describe our training routine.

Data Preprocessing: Each input feature is transformed into a normal distribution using quantile transform. Specifically, we use the scikit-learn implementation [33]. This step is critical for fast and robust training. The integer labels are normalized by the mean/standard error on **train** split. We performed quantile transform for all models in our experiments (SDTR, FCNN, NODE, XGBoost, TabNet, and DeepForest).

Training: SDTR is trained end-to-end via SGD-based optimizer. We only used MSE as the objective function in the experiments, but any differentiable objective function can fit into the SDTR model. As in [34], we use Quasi-Hyperbolic Adam (QHAdam) [24] as our optimizer.

The w and b for each node is initialized via Xavier's uniform [14]. R_ℓs are initialized by a standard normal distribution.

Hyperparameter Tuning: For single and shallow settings, we performed a grid search for depth, λ_1 and λ_2. The depth is selected from $\{4, 5, 6, 7, 8\}$, and both λ_1 and λ_2 are selected from $\{0.1, 0.05, 0.01, 0.005, 0.001\}$.

The group of hyperparameters that yields the best test MSE in SDTR-Single was used as hyperparameters in the SDTR-Shallow setting. The number of trees across all datasets is $n = 256$.

For deep settings, we performed a mixture of grid and random search on the hyperparameters of the model via Hyperopt [3]. The tuned hyperparameters include depth of a single tree, output dimensions of each tree, number of trees in one "boosting" layer, number of layers, learning rate, λ_1 and λ_2. Only the last two parameters are searched continuously.

The hyperparameter choices of other benchmark models are described in Appendix A.

D. RESULTS AND ANALYSIS

We use mean squared error (MSE) as our metric across all five datasets. As shown in table 2, SDTR performed well in "single tree" and "shallow" settings while performed slightly worse than NODE/XGBoost after being boosted. In small datasets (SARCOS and WINE), SDTR showed better robustness than other neural models (FCNN, NODE, and TabNet), and is comparable with state-of-the-art undifferentiable models (XGBoost, LightGBM, CatBoost, and DeepForest).

It is worth noting that we use the same structure / hyperparameters for SDTR for all datasets in the same set of experiments. For "shallow" setups, the hyperparameters, including learning rate, tree depth, and λs, are identical to those in the "single" setup. We found that the hyperparameter setting which performed well in the "single tree" setup also performed well after being ensembled. The consistent performance of SDTR shows the robustness of our proposed model. We can perform a low-cost hyperparameter search in the "single tree" setup; the resulting optimal hyperparameters can be used for the following ensemble phase. It might also explain why the shallow SDTR models performed better than their deeper counterparts in YAHOO and SARCOS. Theoretically, shallow models have few advantages against deeper counterparts: their solution space is a subset of deeper models; however, deeper models naturally have more hyperparameters, need more careful tuning, and have a higher risk of overfitting. The boosted model for YAHOO showed signs of overfitting in our training phase. The "boosting" process also introduces significant structural change; we have found no evidence showing that the optimal hyperparameters for "single" and "shallow" setups are still optimal for the boosted models. Furthermore, the boosting scheme significantly increases the training time and memory consumption, while the performance gain against the "shallow" scheme is only trivial. Thus we do not recommend using the boosted version of SDTR. If we would use the same number of trees in specific cases (for example, the memory is limited), placing all trees in the same, "shallow" layer would be a better choice.

TABLE 2. Experiment results.

Model	Year	MSLR	Yahoo	SARCOS	WINE
FCNN-Shallow	85.44	0.5899	0.6431	414.5	0.6650
NODE-Small	80.14	0.5791	0.6137	14.68	0.4329
SDTR-Single	**78.19**	**0.5656**	**0.5858**	**12.33**	**0.4022**
XGBoost-Default	81.11	0.5637	0.5756	6.07	**0.3174**
NODE-Shallow	78.42	**0.5586**	0.5663	4.47	0.4539
SDTR-Shallow	**76.83**	0.5614	**0.5624**	**3.17**	0.3873
XGBoost-Tuned	78.64	0.5545	0.5418	2.21	0.3243
LightGBM	79.64	0.5565	0.5542	3.76	0.3727
CatBoost	77.81	**0.5524**	**0.5411**	**1.33**	0.3656
FCNN-Deep	81.35	0.5616	0.5778	410.2	0.6650
NODE-Deep	**76.30**	0.5569	0.5693	19.09	0.4348
SDTR-Deep	76.67	0.5591	0.5705	6.21	0.3834
TabNet	77.76	0.5654	0.5851	3.93	0.4174
DeepForest	79.87	0.5666	0.5531	3.88	**0.3150**

V. ABLATION STUDY

In this section, we compare the different hyperparameter settings of our SDTR model. We also justify the effectiveness of our proposed training techniques, i.e. exponentially decaying L_1 regularization and gradient re-weighting. Besides, we proposed a new "average entropy" metric to evaluate the model's interpretability, finding that the new metric is consistent with our optimization target (MSE) in tree regression tasks, possibly explaining the effectiveness of tree-structured model in this field.

A. ABLATION STUDY: BATCH SIZE AND GRADIENT RE-WEIGHTING

In many cases, the size of each batch is critical in models that use Batch Normalization layers [17]. Despite the fact

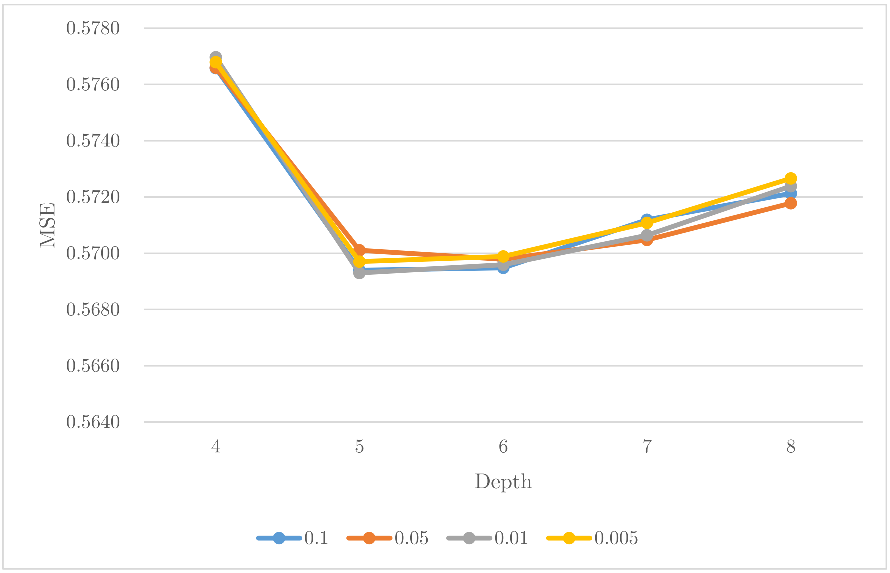


FIGURE 2. The ablation result on Microsoft-Web10K dataset. Each line denotes a different choice of λ_1.

TABLE 3. Ablation results on batch size and gradient re-weighting.

Batch size	Year	MSLR	Yahoo
256	78.87	0.5709	0.5963
256 (g)	78.56	0.5702	0.5925
1024	78.90	0.5709	0.6035
1024 (g)	78.73	0.5708	0.5923
4096	78.87	0.5708	0.6081
4096 (g)	78.61	0.5699	0.5942
16384	78.99	0.5733	0.6243
16384 (g)	78.57	0.5698	0.5941

that our model did not use BN, the regression terms on each inner node are still computed per-batch, which naturally led to a guess that larger batch size results in a more precise approximation, thus improving the model's overall MSE. We provide ablation results in table 3. We keep all hyperparameters (including random seeds) the same and coordinated our early-stopping conditions to ensure that we stop training after not observing improvement for exactly the same amount of data. For each batch size setting, we provide two results (with or without gradient re-weighting).

Although a larger batch size accelerated the overall training process, we did not observe apparent improvements in the model's accuracy. On the other hand, a smaller batch size (and more training batches) might benefit our QHAdam optimizer, for the optimizer will update its momentums more frequently. The consistent performance on different choices of batch size makes SDTR capable of implementations in low-resource settings.

Multiplying the learning rate related to "responses" R_ℓ improved the performance by approximately 2%, which proved the usefulness of the learning-rate re-weighting technique.

B. ABLATION STUDY: λ_1 AND DEPTH FOR SINGLE-TREE SETTING

To test the stability against different hyperparameters, we ran ablation studies on the MSLR-Web10K dataset with different λ_1 and tree depth choices. We provide the results in figure 2. To control the variables, we set $\lambda_2 = 0$ in this set of experiments. Because of the absence of exponentially decaying L_1 regularization, the performance in this set of experiments is slightly worse than our main result on MSLR.

To sum up, the value of λ_1 is flexible and just slightly affects the model's performance, while a suitable depth of tree will bring a significant boost. Based on our observations, the optimal choice of depth depends on the dataset's characteristics.

C. ABLATION STUDY: SINGLE-LAYER ENSEMBLE

As shown in previous results, the model can achieve higher precision when we integrate multiple trees in the same layer and take the average of each tree as our final output. To investigate this method, we performed an ablation study on the different number of trees and the depth of each tree on the Microsoft-Web10K dataset. The results are provided in figure 3.

A straightforward conclusion is that the model's accuracy improves as the number of trees increases. Also, the accuracy seems to be related to the accuracy of every single tree. The hyperparameter choice, which performs well on its own (such as depth $=$ 5 in single-tree ablation), also yields the best result when forming an ensemble. As mentioned before, this characteristic of SDTR is friendly to hyperparameter tuning.

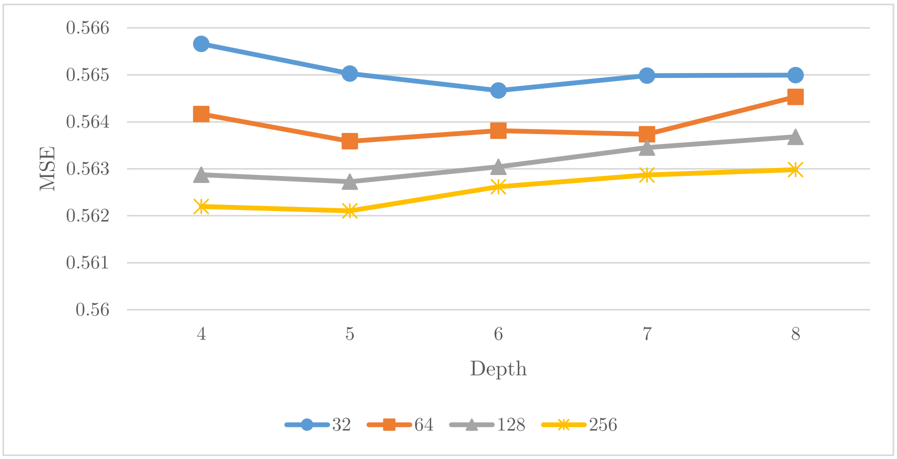


FIGURE 3. The single-layer ensemble ablation result on the Microsoft-Web10K dataset. Each line denotes a different choice of the number of trees for ensembling.

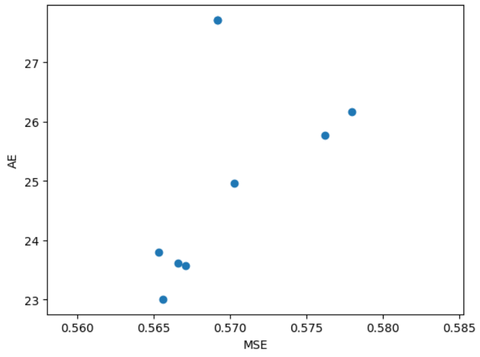


FIGURE 4. The visualization of table 4. For different settings of λ_1, MSEs and AEs are correlated.

D. ABLATION STUDY: L1 REGULARIZATION AND INTERPRETABILITY ANALYSIS

As proposed in section III-B, a sparse weight matrix implies better interpretability. To quantitatively evaluate an SDTR's interpretability, we propose an "average entropy" metric suitable for our experimental setup.

For each non-leaf node in our tree, the $1 \times k$ input features are multiplied by a $k \times 1$ weight matrix W to obtain a scalar, which is fed into a sigmoid function. Note that all input features are already normalized, so the scale of weights on different features are roughly the same. Naturally, a larger weight (absolute value) means the corresponding feature is more effective; thus, we normalize W to get a new matrix W':

$$W' = \frac{|W| + \epsilon}{\sum(|W| + \epsilon)} \tag{10}$$

ϵ is a small positive scalar to prevent the occurrence of zeros. In all of our experiments we use $\epsilon = 10^{-3}$. The values inside our new W' is between [0, 1] and sums up to 1. This W' can be viewed as a multinoulli distribution, and thus we

TABLE 4. Ablation results on L1 regularization (MSLR).

λ_2	MSE	AE
1	0.5780	26.173
0.5	0.5762	25.770
0.1	0.5703	24.966
0.05	0.5671	23.568
0.01	**0.5656**	**23.000**
0.005	0.5666	23.615
0.001	**0.5653**	23.794
0	0.5675	27.869

can compute its entropy:

$$H(W') = -\sum_{w' \in W'} w' \log(w') \tag{11}$$

Intuitively, if we have one (or several) dominating feature(s), the entropy of conducted multinoulli will be small; otherwise, if all features are of equal importance, the previous expression will reach its maximum.

Again, we use the weighted sum of such entropy of all nodes to get our final "average entropy" (AE) metric. Let $W'(t)$ be the normalized weight matrix corresponding to an inner node t, our metric is then given by:

$$AE = \sum_{t \in \text{InnerNodes}} 2^{-d_t} H(W'(t)) \tag{12}$$

We use this metric to evaluate the effectiveness of our exponentially decaying L_1 regularization. Table 4 shows the relationship among λ_2, MSE and AE. $\lambda_2 = 0$ means we do not use L_1 regularization at all.

It can be shown that the existence of the L_1 regularization term improved the model's interpretability, and a proper value of λ_2 improved the model's performance. An interesting finding is that a model with a smaller MSE tends to have a smaller AE. Figure 4 further illustrates this phenomenon. This phenomenon also showed up in our experiments on the other two datasets. Note that we do not use AE to optimize our models: it did not show up in our loss function. Furthermore,

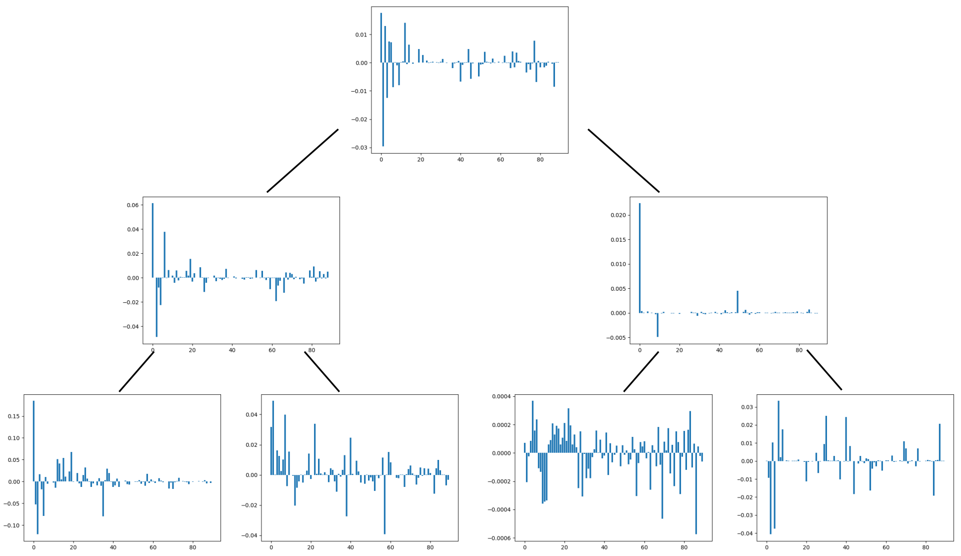


FIGURE 5. The visualization of an depth = 4 SDTR tree's first three layers.

the early-stopping scheme relies solely on the MSE on the validation set. This finding may imply that the tabular data regression task favors sparse weight matrixs, at least on these datasets.

We also provide a visualization of the first three layers for a shallow SDTR single tree (depth = 4), trained on the YearPredictionMSD dataset. The model reached 79.32 MSE. Each sub-gram denotes the weights associated with each node. Our L_1 regularization efficiently encouraged sparsity for $d = 1$ and $d = 2$ nodes; because of the exponentially decaying weight, the regularization strength (and sparsity) decreases in deeper layers. One can already conclude some most important features (in this example, feature #0), and exclude some unused features in following experiments.

It is worth noting that ensemble and boosting will harm the interpretability of the base model. This observation also holds for SDTR. In our ''single layer ensemble'' scheme, the final output is an average of every tree regressor's prediction. Every tree's contribution to the final output is associated with the statistics of the tree's path probabilities and leaf predictions. We leave this direction for the future.

E. ABLATION STUDY: GRADIENT REWEIGHTING ON SDT

Theoretically, gradient vanishing at leaf nodes also exists in the original SDT for classification tasks. To justify our gradient reweighting technique, we performed an ablation study on the original SDT model proposed by Frosst and Hinton [13]. We used depth = 5 and learning rate 10^{-3} on the MNIST classification task, using cross entropy as our loss function, and the only difference between the two models is whether the gradient reweighting technique is used.

Figure 6 showed our results on MNIST. The reweighting method significantly accelerated our training process, reaching a near-optimal result after a few hundred epochs. It is worth noting that directly increasing the learning rate by 2^{depth} caused the model to fail: the loss functions soon became untractable (nan) after 100 epochs. The final accuracy on the test set was 94.49% (reweight) versus 94.94% (original), which implies that the gradient reweighting might introduce trivial performance loss. Such loss can be easily covered by tuning hyperparameters.

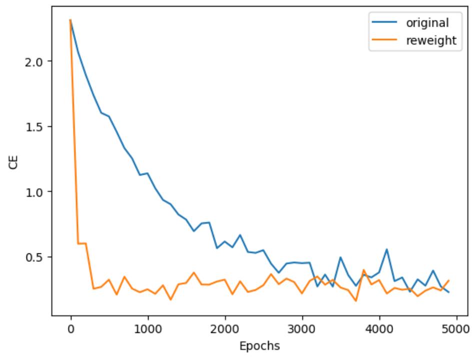


FIGURE 6. The loss function (cross entropy) curve for original model and reweighted model.

VI. CONCLUSION

In this paper, we integrated an existing model (Soft Decision Tree) to handle heterogeneous tabular data regression, which troubles DNNs for a long time. We proved by experiments that despite our model performs slightly worse than NODE and XGBoost under fine-tune settings, it exceeds the two competitors in shallow settings and performs far better than FCNN. The SDTR model performs best in extremely constrained low-memory, single tree setting.

The differentiable nature of SDTR makes it possible to be incorporated into complex pipelines, and it can easily fit into the back-propagation regime. As a regressor, it is possible to use SDTR to extract valuable features from tabular data or directly solve the regression problems with a low cost.

APPENDIX A OPTIMIZATION OF HYPERPARAMETERS

For each model, we use a grid search on the group of parameters described below. All models are trained on the same train/val/test split. Some models, e.g., XGBoost and SDTR, require a validation set to perform early-stopping and prevent overfitting. For those models, we trained them using the train split, performed early-stopping according to the val, and report the best result on the test split among choices of different hyperparameters.

A. FCNN

As described above, the FCNN uses "Dense-LeakyReLU-Dropout" structure as the basic building block. FCNN-Shallow consists of 2 blocks, and FCNN-Deep consists of 7 blocks. The negative slope for LeakyReLU is 0.01. We use Adam optimizer for back-propagation.

The number of neurons in hidden layers were set to be a proportion of input feature size k.

- *hidden_size*: Uniform choice in $\{k/4, k/2, k, 2k\}$ (rounded down).
- *learning_rate*: Log-uniform distribution $[10^{-5}, 10^{-2}]$
- *dropout*: Uniform distribution $[0, 0.5]$

B. XGBoost

The XGBoost-Default setting uses the default XGBRegressor class with *max_depth* $= 3$, *learning_rate* $= 0.1$ and *n_estimators* $= 100$.

To make the comparison fair, XGBoost-Tuned setting uses the same search space as in [34]. We describe the hyperparameter choice below.

- η: Log-uniform distribution in $[e^{-7}, 1]$
- *max_depth*: Discrete uniform distribution in $[2, 10]$
- *subsample*: Uniform distribution $[0.5, 1]$
- *colsample_bytree*: Uniform distribution $[0.5, 1]$
- *colsample_bylevel*: Uniform distribution $[0.5, 1]$
- *min_child_weight*: Log-uniform distribution $[e^{16}, e^{5}]$
- α: Uniform choice in $\{0$, Log-uniform distribution $[e^{-16}, e^{2}]\}$
- λ: Uniform choice in $\{0$, Log-uniform distribution $[e^{-16}, e^{2}]\}$
- γ: Uniform choice in $\{0$, Log-uniform distribution $[e^{-16}, e^{2}]\}$

C. LightGBM

The hyperparameters of LightGBM are sampled by uniform choice from below. The bold text stands for the default hyperparameters.

- *num_leaves*: {15, **31**, 63, 127, 255}
- *learning_rate*: {**0.1**, 0.05, 0.01, 0.005}
- *n_estimators*: {**100**, 300, 500}

D. CatBoost

The hyperparameters of CatBoost are sampled by uniform choice from below. The bold text stands for the default hyperparameters.

- *num_trees*: {500, **1000**, 2000, 4000, 8000}
- *tree_depth*: {**6**, 7, 8, 9, 10}
- *l2_leaf_reg*: {0, 1.0, 2.0, **3.0**, 4.0, 5.0, 6.0}

E. NODE

The hyperparameters of NODE-Small are sampled from below.

- *tree_depth*: Uniform choice in {4, 5, 6, 7, 8}.
- *tree_number*: 10 (fixed)
- *learning_rate*: Log-Uniform distribution $[10^{-4}, 10^{-2}]$

NODE-Shallow inherits the best hyperparameters from NODE-Small, while replacing the *tree_number* with 2048.

Based on our observations, the boosting process greatly influences the model's structure and logic. We use the same search space as in [34]:

- *num_layers*: {2, 4, 8}
- *total_tree_count*: {1024, 2048}
- *tree_depth*: {6, 8}
- *tree_output_dim*: {2, 3}

F. SDTR

The hyperparameters of SDTR-Single and SDTR-Shallow are described above. For SDTR-Deep setting, we performs hyperparameter search in the following space:

- *tree_depth*: {6, 7, 8}
- *num_layers*: {2, 4, 8}
- *total_tree_count*: {1024, 2048}
- *tree_output_dim*: {2, 3}
- *learning_rate*: $\{10^{-2}, 10^{-3}, 10^{-4}\}$
- λ_1: Log-uniform distribution in $[10^{-4}, 10^{-2}]$
- λ_2: Log-uniform distribution in $[10^{-4}, 10^{-2}]$

G. TabNet

We use the PyTorch implementation of TabNet.[2] The hyperparameters are sampled by uniform choice from below. The bold text stands for the default hyperparameters.

- n_d and n_a: {4, **6**, 8, 10} ($n_d = n_a$ in all experiments)
- *n_steps*: {**3**, 4, 5, 6}
- γ: {1.1, 1.2, **1.3**, 1.4}
- λ_{sparse}: {0.1, **0.01**, 0.001}

H. DeepForest

We use the CascadeForestRegressor in the Deep-Forest library.[3] The hyperparameters of CascadeForestRegressor are sampled by uniform choice from below. The bold text stands for the default hyperparameters.

- *n_bins*: {100, 200, **255**}
- *n_trees*: {50, **100**, 200}

[2]https://github.com/dreamquark-ai/tabnet
[3]https://github.com/LAMDA-NJU/Deep-Forest/

REFERENCES

[1] J. Abellán and C. J. Mantas, "Improving experimental studies about ensembles of classifiers for bankruptcy prediction and credit scoring," *Expert Syst. Appl.*, vol. 41, no. 8, pp. 3825–3830, Jun. 2014.

[2] S. O. Arik and T. Pfister, "TabNet: Attentive interpretable tabular learning," 2019, *arXiv:1908.07442*. [Online]. Available: http://arxiv.org/abs/1908.07442

[3] J. Bergstra, D. Yamins, and D. Cox, "Making a science of model search: Hyperparameter optimization in hundreds of dimensions for vision architectures," in *Proc. Int. Conf. Mach. Learn.*, 2013, pp. 115–123.

[4] T. Bertin-Mahieux, D. P. Ellis, B. Whitman, and P. Lamere, "The million song dataset," in *Proc. 12th Int. Conf. Music Inf. Retr. (ISMIR)*, 2011.

[5] L. Breiman, "Random forests," *Mach. Learn.*, vol. 45, no. 1, pp. 5–32, 2001.

[6] L. Breiman, J. Friedman, C. J. Stone, and R. A. Olshen, *Classification Regression Trees*. Boca Raton, FL, USA: CRC Press, 1984.

[7] O. Chapelle and Y. Chang, "Yahoo! learning to rank challenge overview," in *Proc. Learn. Rank Challenge*, 2011, pp. 1–24.

[8] D. Che, Q. Liu, K. Rasheed, and X. Tao, "Decision tree and ensemble learning algorithms with their applications in bioinformatics," in *Software Tools and Algorithms for Biological Systems*. New York, NY, USA: Springer, 2011, pp. 191–199. [Online]. Available: https://link.springer.com/content/pdf/10.1007/978-1-4419-7046-6.pdf

[9] T. Chen and C. Guestrin, "XGBoost: A scalable tree boosting system," in *Proc. 22nd ACM SIGKDD Int. Conf. Knowl. Discovery Data Mining*, 2016, pp. 785–794.

[10] P. Cortez, A. Cerdeira, F. Almeida, T. Matos, and J. Reis, "Modeling wine preferences by data mining from physicochemical properties," *Decis. Support Syst.*, vol. 47, no. 4, pp. 547–553, Nov. 2009.

[11] D. Dua and C. Graff. (2017). *UCI Machine Learning Repository*. [Online]. Available: http://archive.ics.uci.edu/ml

[12] J. H. Friedman, "Greedy function approximation: A gradient boosting machine," *Ann. Statist.*, vol. 29, no. 5, pp. 1189–1232, 2001.

[13] N. Frosst and G. Hinton, "Distilling a neural network into a soft decision tree," 2017, *arXiv:1711.09784*. [Online]. Available: http://arxiv.org/abs/1711.09784

[14] X. Glorot and Y. Bengio, "Understanding the difficulty of training deep feedforward neural networks," in *Proc. 13th Int. Conf. Artif. Intell. Statist.*, 2010, pp. 249–256.

[15] A. Graves, A.-R. Mohamed, and G. Hinton, "Speech recognition with deep recurrent neural networks," in *Proc. IEEE Int. Conf. Acoust., Speech Signal Process.*, May 2013, pp. 6645–6649.

[16] K. He, X. Zhang, S. Ren, and J. Sun, "Deep residual learning for image recognition," in *Proc. IEEE Conf. Comput. Vis. Pattern Recognit. (CVPR)*, Jun. 2016, pp. 770–778.

[17] S. Ioffe and C. Szegedy, "Batch normalization: Accelerating deep network training by reducing internal covariate shift," in *Proc. Int. Conf. Mach. Learn.*, 2015, pp. 448–456.

[18] M. I. Jordan and R. A. Jacobs, "Hierarchical mixtures of experts and the EM algorithm," *Neural Comput.*, vol. 6, no. 2, pp. 181–214, Mar. 1994.

[19] G. Ke, Q. Meng, T. Finley, T. Wang, W. Chen, W. Ma, Q. Ye, and T.-Y. Liu, "LightGBM: A highly efficient gradient boosting decision tree," in *Proc. Adv. Neural Inf. Process. Syst.*, 2017, pp. 3146–3154.

[20] G. K. Kumar, P. Viswanath, and A. A. Rao, "Ensemble of randomized soft decision trees for robust classification," *Sādhanā*, vol. 41, pp. 273–282, Mar. 2016.

[21] A. Crimi, B. H. Menze, O. Maier, M. Reyes, S. Winzeck, and H. Handels, Eds., "Brainlesion: Glioma, multiple sclerosis, stroke and traumatic brain injuries," in *Proc. 2nd Int. Workshop, BrainLes, Challenges BRATS, ISLES mTOP, Held Conjunction (MICCAI)*, in Lecture Notes in Computer Science, vol. 10154, Athens, Greece, Oct. 2016.

[22] Y. LeCun and C. Cortes. (2010). *MNIST Handwritten Digit Database*. [Online]. Available: http://yann.lecun.com/exdb/mnist/

[23] A. Léon and L. Denoyer, "Policy-gradient methods for decision trees," in *Proc. ESANN*, 2016, pp. 1–6.

[24] R. Tanno, K. Arulkumaran, D. C. Alexander, A. Criminisi, and A. V. Nori, "Adaptive neural trees," in *Proc. 36th Int. Conf. Mach. Learn. (ICML)*, in Proceedings of Machine Learning Research, vol. 97, K. Chaudhuri and R. Salakhutdinov, Eds. Long Beach, CA, USA: PMLR, Jun. 2019, pp. 6166–6175. [Online]. Available: http://proceedings.mlr.press/v97/tanno19a.html

[25] V. Mnih, K. Kavukcuoglu, D. Silver, A. A. Rusu, J. Veness, M. G. Bellemare, A. Graves, M. Riedmiller, A. K. Fidjeland, G. Ostrovski, S. Petersen, C. Beattie, A. Sadik, I. Antonoglou, H. King, D. Kumaran, D. Wierstra, S. Legg, and D. Hassabis, "Human-level control through deep reinforcement learning," *Nature*, vol. 518, no. 7540, pp. 529–533, 2015.

[26] D. C. Montgomery, E. A. Peck, and G. G. Vining, *Introduction to Linear Regression Analysis*, vol. 821. Hoboken, NJ, USA: Wiley, 2012.

[27] E. A. Nadaraya, "On estimating regression," *Theory Probab. Appl.*, vol. 9, no. 1, pp. 141–142, 1964.

[28] V. Nair and G. E. Hinton, "Rectified linear units improve restricted Boltzmann machines," in *Proc. ICML*, 2010, pp. 1–8.

[29] A. Onan, "A fuzzy-rough nearest neighbor classifier combined with consistency-based subset evaluation and instance selection for automated diagnosis of breast cancer," *Expert Syst. Appl.*, vol. 42, no. 20, pp. 6844–6852, Nov. 2015.

[30] A. Onan, S. Korukoğlu, and H. Bulut, "Ensemble of keyword extraction methods and classifiers in text classification," *Expert Syst. Appl.*, vol. 57, pp. 232–247, Sep. 2016.

[31] A. Onan, S. Korukoğlu, and H. Bulut, "A multiobjective weighted voting ensemble classifier based on differential evolution algorithm for text sentiment classification," *Expert Syst. Appl.*, vol. 62, pp. 1–16, Nov. 2016.

[32] A. Paszke, S. Gross, F. Massa, A. Lerer, J. Bradbury, G. Chanan, T. Killeen, Z. Lin, N. Gimelshein, L. Antiga, A. Desmaison, A. Köpf, E. Yang, Z. DeVito, M. Raison, A. Tejani, S. Chilamkurthy, B. Steiner, L. Fang, J. Bai, and S. Chintala, "Pytorch: An imperative style, high-performance deep learning library," in *Proc. Adv. Neural Inf. Process. Syst.*, 2019, pp. 8026–8037.

[33] 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," *J. Mach. Learn. Res.*, vol. 12, pp. 2825–2830, Oct. 2011.

[34] 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," *J. Mach. Learn. Res.*, vol. 12, pp. 2825–2830, 2011. [Online]. Available: http://dl.acm.org/citation.cfm?id=2078195

[35] T. Qin and T.-Y. Liu, "Introducing LETOR 4.0 datasets," 2013, *arXiv:1306.2597*. [Online]. Available: http://arxiv.org/abs/1306.2597

[36] G. Ridgeway, "Generalized boosted models: A guide to the GBM package," *Update*, vol. 1, no. 1, p. 2007, 2007.

[37] S. Rota Bulo and P. Kontschieder, "Neural decision forests for semantic image labelling," in *Proc. IEEE Conf. Comput. Vis. Pattern Recognit.*, Jun. 2014, pp. 81–88.

[38] V. Sandulescu and M. Chiru, "Predicting the future relevance of research institutions–the winning solution of the KDD cup 2016," 2016, *arXiv:1609.02728*. [Online]. Available: http://arxiv.org/abs/1609.02728

[39] D. F. Specht, "A general regression neural network," *IEEE Trans. Neural Netw.*, vol. 2, no. 6, pp. 568–576, Nov. 1991.

[40] A. Suarez and J. F. Lutsko, "Globally optimal fuzzy decision trees for classification and regression," *IEEE Trans. Pattern Anal. Mach. Intell.*, vol. 21, no. 12, pp. 1297–1311, Dec. 1999.

[41] R. Tanno, K. Arulkumaran, D. C. Alexander, A. Criminisi, and A. V. Nori, "Adaptive neural trees," in *Proc. Mach. Learn. Res. (ICML)*, vol. 97. PMLR, 2019, pp. 6166–6175.

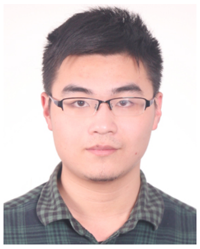
[42] T. M. Therneau and P. M. Grambsch, "The Cox model," in *Modeling Survival Data: Extending the Cox Model*. Springer, 2000, pp. 39–77.

[43] C.-F. Tsai, Y.-C. Lin, D. C. Yen, and Y.-M. Chen, "Predicting stock returns by classifier ensembles," *Appl. Soft Comput.*, vol. 11, no. 2, pp. 2452–2459, Mar. 2011.

[44] A. Vaswani, N. Shazeer, N. Parmar, J. Uszkoreit, L. Jones, A. N. Gomez, L. Kaiser, and I. Polosukhin, "Attention is all you need," in *Proc. Adv. Neural Inf. Process. Syst.*, 2017, pp. 5998–6008.

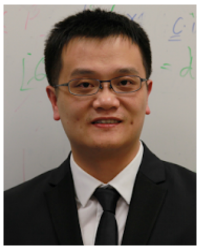
[45] S. Vijayakumar and S. Schaal, "Locally weighted projection regression: An $o(n)$ algorithm for incremental real time learning in high dimensional space," in *Proc. 17th Int. Conf. Mach. Learn. (ICML)*, vol. 1, 2000, pp. 288–293.

[46] M. Volkovs, G. W. Yu, and T. Poutanen, "Content-based neighbor models for cold start in recommender systems," in *Proc. Recommender Syst. Challenge*, 2017, pp. 1–6.
[47] Y. Yang, I. Garcia Morillo, and T. M. Hospedales, "Deep neural decision trees," 2018, *arXiv:1806.06988*. [Online]. Available: http://arxiv.org/abs/1806.06988
[48] O. T. Yíldíz, O. I. Rsoy, and E. Alpaydín, "Bagging soft decision trees," in *Proc. JMLR Workshop Conf.* Brookline, MA, USA: PMLR, 2016, pp. 25–36.
[49] Z.-H. Zhou and J. Feng, "Deep forest: Towards an alternative to deep neural networks," in *Proc. 26th Int. Joint Conf. Artif. Intell.*, Aug. 2017, pp. 3553–3559.
[50] L. O. Prokhorenkova, G. Gusev, A. Vorobev, A. V. Dorogush, and A. Gulin, "CatBoost: Unbiased boosting with categorical features," in *Proc. Annu. Conf. Neural Inf. Process. Syst. (NeurIPS)*, S. Bengio, H. M. Wallach, H. Larochelle, K. Grauman, N. Cesa-Bianchi, and R. Garnett, Eds. Montréal, QC, Canada, Dec. 2018, pp. 6639–6649. [Online]. Available: https://proceedings.neurips.cc/paper/2018/hash/14491b756b3a51daac41c24863285549-Abstract.html



HAORAN LUO was born in Fujian, China, in 1996. He received the B.S. degree from Shanghai Jiao Tong University, Shanghai, China, in 2018, where he is currently pursuing the M.S. degree in computer science.

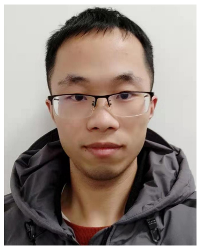
From 2018 to 2019, he was an Intern with Bosch (China) Investment Company Ltd. From 2019 to 2020, he was an Intern with the Algorithm Department, 360 Digitech Inc., Shanghai. His research interests include machine learning, ensemble learning, and generative models.



FAN CHENG (Member, IEEE) received the bachelor's degree in computer science and engineering from Shanghai Jiao Tong University, in 2007, and the Ph.D. degree in information engineering from The Chinese University of Hong Kong, in 2012. From 2012 to 2014, he was a Postdoctoral Fellow with the Institute of Network Coding, The Chinese University of Hong Kong. Since 2015, he has been a Research Fellow with the Department of ECE, NUS, Singapore. He joined the faculty of the Department of Computer Science and Engineering, Shanghai Jiao Tong University, in 2016.

HENG YU (Graduate Student Member, IEEE) was born in Anhui, China, in 1997. He received the B.S. degree from Shanghai Jiao Tong University, Shanghai, China, in 2018, where he is currently pursuing the M.S. degree in computer science.

He was an Algorithm Intern with the Artificial Intelligence Department, Xiaomi Corporation, Beijing, in 2018. From 2019 to 2020, he was an Intern with the Algorithm Department, 360 Digitech Inc., Shanghai. His research interests include unsupervised learning, natural language processing, and computational advertising theory.



YUQI YI (Student Member, IEEE) was born in Hunan, China, in 1996. He received the B.S. degree from Shanghai Jiao Tong University, Shanghai, China, in 2019, where he is currently pursuing the M.S. degree in computer science.

He was an Algorithm Intern with the YouTu Department, Tencent, Shenzhen, in 2018. He was an Intern with the Anti-Fraud Department, 360 Digitech Inc., Shanghai. From 2020 to 2021, he was an Intern with the Recommendation Department, Bytedance, Shanghai. His research interests include data mining, fraud detection, and recommendation systems.