Team up GBDTs and DNNs:
Advancing Efficient and Effective Tabular Prediction with Tree-hybrid MLPs

KDD 2024

資訊所 P76124752 莊上緣

Outline

- Introduction
- Related Work
- Tree-Hybrid Simple MLP
- Experiments
- Conclusions

Outline

- **Introduction**
- Related Work
- Tree-Hybrid Simple MLP
- Experiments
- Conclusions

Introduction

- Table data is **ubiquitous** in real-world applications, so we need a powerful, efficient, and user-friendly table prediction method.
- Current prevalent tabular prediction (i.e., classification and regression) models can be generally categorized into two main types:
  - GBDT(Gradient Boosting Decision Tree)
  - DNN(Deep Neural Network)

Introduction

GBDT (Gradient Boosting Decision Tree)

- advantages
    - greedy feature selection
    - tree pruning
    - efficient ensemble
- disadvantages
    - hyperparameter-sensitive
    - not well-suited in extreme tabular scenarios (such as large-scale tables with intricate feature interactions)

Introduction

DNN (Deep Neural Network)

- advantages
    - capability of mining subtle feature interactions
    - smooth back-propagation optimization
    - high-dimensional feature spaces
- disadvantages
    - over-parameterized
    - data-hungry(need more training data than tree-based method)
    - processing latency

Introduction

	GBDT	DNN
advantages	• greedy feature selection • tree pruning • efficient ensemble	• capability of **mining feature interactions** • smooth back-propagation optimization • high-dimensional feature spaces

Introduction

	GBDT	DNN
advantages	• greedy feature selection • tree pruning • efficient ensemble	• capability of **mining feature interactions** • smooth back-propagation optimization • high-dimensional feature spaces
disadvantages	• hyperparameter-sensitive • not well-suited in **extreme** tabular scenarios (such as **large-scale** tables with intricate feature interactions)	• over-parameterized • data-hungry(need more training data than tree-based method) • processing latency

Introduction

	GBDT	DNN
advantages	• greedy feature selection • tree pruning • efficient ensemble	• capability of **mining feature interactions** • smooth back-propagation optimization • high-dimensional feature spaces
disadvantages	• hyperparameter-sensitive • not well-suited in **extreme** tabular scenarios (such as **large-scale** tables with intricate feature interactions)	• over-parameterized • data-hungry(need more training data than tree-based method) • processing latency
Shared Limitations	In order to achieve respective SOTA results, both of them need **expensive training costs(heavy hyperparameter search)**. But this is carbon-unfriendly and is not compatible in computation-limited or real-time applications	

Introduction

To address the model selection dilemma, we comprehensively combine the advantages of both GBDTs and DNNs, and propose a new **Tree-hybrid simple MLP (T-MLP):**

- using **GBDT feature gate** to perform sample-specific **feature selection** in a **greedy** fashion
- GBDT-inspired **pruned** MLP architectures to process the selected salient features
- the whole framework is optimized using **back-propagation** with these GBDTs' properties

T-MLP is high-performing, efficient, and lightweight!

Outline

- Introduction
- **Related Work**
- Tree-Hybrid Simple MLP
- Experiments
- Conclusions

Related Work

Model Frameworks for Tabular Prediction

GBDTs	Widely used in tabular prediction for efficiency and robustness.
DNNs	Traditionally for unstructured data, now adapted for tabular tasks.
DNN Architectures	Includes NODE, TabNet, AutoInt.
Recent Designs	FT-Transformer, SAINT leverage feature fusion, often matching or surpassing GBDTs in specific scenarios.

Related Work

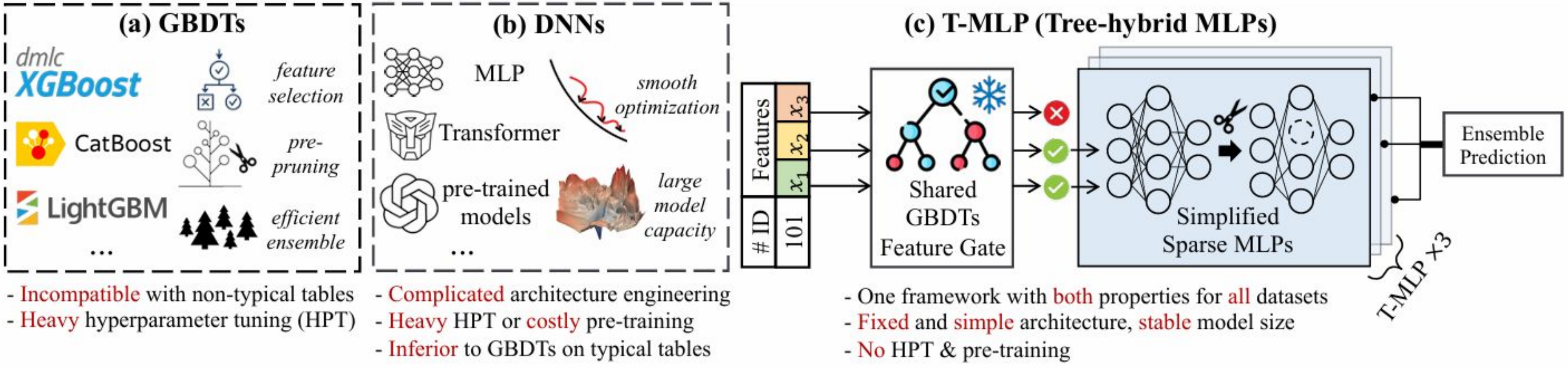
Lightweight DNNs

Lightweight DNNs	Aim to balance performance, compactness, and efficiency.
Recent Models	Examples include MLP-Mixer and gMLP, which match CNNs and Transformers in performance while being simpler.
Model Compression	Pruning is a common technique used.
Tabular DNNs	Lightweight designs and compression methods are underexplored.
T-MLP	Introduces techniques for improved compactness and effectiveness in tabular prediction.

Outline

- Introduction
- Related Work
- **Tree-Hybrid Simple MLP**
- Experiments
- Conclusions

Architecture Overview



Preliminaries

Problem Statement

Tabular Dataset $X \in \mathbb{R}^{N \times F}$

Target $y \in \mathbb{R}^{N}$

optimal solution $f : \in \mathbb{R}^{N \times F} \rightarrow \mathbb{R}^{N}$

N is the total number of data

F is the total number of features

Target is to minimize the empirical difference between the predictions $\hat{y}$ and the targets y.

Preliminaries

GBDT Feature Frequency

the process of GBDT inference on a sample $x \in \mathbb{R}^F$ provides T times a single decision tree prediction:

$$\hat{y}^{(k)} = \mathrm{CART}^{(k)}(x), k \in \{1, 2, \ldots, T\}$$

We denote this accessed feature list of the k-th decision tree as a binary vector:

$$\alpha^{(k)} \in \{0, 1\}^F$$

0 indicates that the corresponding feature of this sample is not used by the k-th decision
1 indicates that it is accessed.

we can represent the GBDT feature frequency of the sample with the sum of the$k$ decision trees' binary vectors, as:

$$\alpha = \sum_k \alpha^{(k)}$$

Preliminaries

GBDT Feature Frequency

For example, You have a simple dataset with three features (Feature 1, Feature 2, Feature 3), and the GBDT model includes three decision trees.

Preliminaries

GBDT Feature Frequency

For example, You have a simple dataset with three features (Feature 1, Feature 2, Feature 3), and the GBDT model includes three decision trees.

Decision Tree 1: uses Feature 1 and Feature 2 for splitting, feature usage list is [1, 1, 0]

Preliminaries

GBDT Feature Frequency

For example, You have a simple dataset with three features (Feature 1, Feature 2, Feature 3), and the GBDT model includes three decision trees.

Decision Tree 1: uses Feature 1 and Feature 2 for splitting, feature usage list is [1, 1, 0]

Decision Tree 2: uses only Feature 3 for splitting, feature usage list is [0, 0, 1]

Preliminaries

GBDT Feature Frequency

For example, You have a simple dataset with three features (Feature 1, Feature 2, Feature 3), and the GBDT model includes three decision trees.

Decision Tree 1: uses Feature 1 and Feature 2 for splitting, feature usage list is [1, 1, 0]

Decision Tree 2: uses only Feature 3 for splitting, feature usage list is [0, 0, 1]

Decision Tree 3: uses Feature 1 and Feature 3 for splitting, feature usage list is [1, 0, 1]

Preliminaries

GBDT Feature Frequency

For example, You have a simple dataset with three features (Feature 1, Feature 2, Feature 3), and the GBDT model includes three decision trees.

Decision Tree 1: uses Feature 1 and Feature 2 for splitting, feature usage list is [1, 1, 0]

Decision Tree 2: uses only Feature 3 for splitting, feature usage list is [0, 0, 1]

Decision Tree 3: uses Feature 1 and Feature 3 for splitting, feature usage list is [1, 0, 1]

Feature frequency = [1, 1, 0] + [0, 0, 1] + [1, 0, 1] = **[2, 1, 2]**

Preliminaries

GBDT Feature Frequency

For example, You have a simple dataset with three features (Feature 1, Feature 2, Feature 3), and the GBDT model includes three decision trees.

Decision Tree 1: uses Feature 1 and Feature 2 for splitting, feature usage list is [1, 1, 0]

Decision Tree 2: uses only Feature 3 for splitting, feature usage list is [0, 0, 1]

Decision Tree 3: uses Feature 1 and Feature 3 for splitting, feature usage list is [1, 0, 1]

Feature frequency = [1, 1, 0] + [0, 0, 1] + [1, 0, 1] = **[2, 1, 2]**

This vector reveals the **model's preference for each feature**; features with higher frequencies are used more often in the decision-making process and are likely to have a larger influence on the final prediction outcome.

Preliminaries

Feature Tokenizer

Inspired by the classical language models (e.g., BERT), recent dominating Transformer-based tabular models adopted distributed feature representation by **embedding tabular values into vector spaces** and treating the values as “unordered” word vectors.

Given ***F*1 numerical** features and ***F*2 categorical** features, the feature tokenizer outputs feature embedding by stacking projected features (and an extra [**CLS**] token embedding, Similar to the [CLS] token in BERT, this is an **additional global representation token added to capture the overall features** of the entire input.)

$$E \in \mathbb{R}^{(1+F_1+F_2)\times d}$$

$$E = \operatorname{stack}\left(\left[e_{\mathrm{CLS}}, e_{\mathrm{num}}^{(1)}, \ldots, e_{\mathrm{num}}^{(F_1)}, e_{\mathrm{cat}}^{(1)}, \ldots, e_{\mathrm{cat}}^{(F_2)}\right]\right)$$

GBDT Feature Gate

- Traditional DNN models for tabular data attempt to **imitate** the behavior of GBDT by performing hard feature selection through backpropagation in neural networks.
- However, due to differences between the **continuity of neural networks** and the **discrete nature of GBDT**, this approach may be **incompatible** with the essence of GBDT, limiting the model's potential.
- To resolve this issue, the authors propose **GBDT Feature Gate (GFG)**, a GBDT-based feature selector that incorporates GBDT weights as tensors to faithfully replicate its feature selection behavior.

GBDT Feature Gate

given a GFG initialized by a T-tree GBDT, the feature selection process on an F-feature sample:

$$x \; (\hat{E} = \mathrm{GFG}(x) \in \mathbb{R}^{F \times d})$$

GBDT Feature Gate

given a GFG initialized by a T-tree GBDT, the feature selection process on an F-feature sample:

$$x \; (\hat{E} = \mathrm{GFG}(x) \in \mathbb{R}^{F\times d})$$

is formulated as:

$$E = \mathrm{FeatureTokenizer}(x) \in \mathbb{R}^{F\times d}, \tag{1}$$

GBDT Feature Gate

given a GFG initialized by a T-tree GBDT, the feature selection process on an F-feature sample:

$$x \; (\hat{E} = \mathrm{GFG}(x) \in \mathbb{R}^{F \times d})$$

is formulated as:

$$E = \mathrm{FeatureTokenizer}(x) \in \mathbb{R}^{\boxed{F \times d}}, \qquad (1)$$

The **extra [CLS] embedding is omitted** in this subsection

for notation brevity

$$E \in \mathbb{R}^{\boxed{(1+F_1+F_2) \times d}}$$

GBDT Feature Gate

given a GFG initialized by a T-tree GBDT, the feature selection process on an F-feature sample:

$$x\ (\hat{E} = \mathrm{GFG}(x) \in \mathbb{R}^{F\times d})$$

is formulated as:

$$E = \mathrm{FeatureTokenizer}(x) \in \mathbb{R}^{F\times d}, \tag{1}$$

$$\alpha = \mathrm{GBDTFeatureFrequency}(x) \in \mathbb{R}^{F}, \tag{2}$$

GBDT Feature Gate

given a GFG initialized by a T-tree GBDT, the feature selection process on an F-feature sample:

$$x\ (\hat{E} = \mathrm{GFG}(x) \in \mathbb{R}^{F\times d})$$

is formulated as:

$$E = \mathrm{FeatureTokenizer}(x) \in \mathbb{R}^{F\times d}, \tag{1}$$

$$\alpha = \mathrm{GBDTFeatureFrequency}(x) \in \mathbb{R}^{F}, \tag{2}$$

$$\hat{\alpha} = \alpha/T \in \mathbb{R}^{F}, \bar{\alpha} = \mathrm{BinarySampler}(\hat{\alpha}) \in \{0,1\}^{F}, \tag{3}$$

GBDT Feature Gate

given a GFG initialized by a T-tree GBDT, the feature selection process on an F-feature sample:

$$x \ (\hat{E} = \mathrm{GFG}(x) \in \mathbb{R}^{F \times d})$$

is formulated as:

$$E = \mathrm{FeatureTokenizer}(x) \in \mathbb{R}^{F \times d}, \qquad (1)$$

$$\boxed{\alpha} = \mathrm{GBDTFeatureFrequency}(x) \in \mathbb{R}^{F}, \qquad (2)$$

$$\boxed{\hat{\alpha} = \alpha / T} \in \mathbb{R}^{F}, \bar{\alpha} = \mathrm{BinarySampler}(\hat{\alpha}) \in \{0, 1\}^{F}, \qquad (3)$$

Divide FeatureFrequency by the total count T to convert it into a vector with values between 0 and 1, representing the **selection probability for each feature**.

GBDT Feature Gate

given a GFG initialized by a T-tree GBDT, the feature selection process on an F-feature sample:

$$x\ (\hat{E} = \mathrm{GFG}(x) \in \mathbb{R}^{F\times d})$$

is formulated as:

$$E = \mathrm{FeatureTokenizer}(x) \in \mathbb{R}^{F\times d}, \tag{1}$$

$$\alpha = \mathrm{GBDTFeatureFrequency}(x) \in \mathbb{R}^{F}, \tag{2}$$

$$\hat{\alpha} = \alpha / T \in \mathbb{R}^{F}, \bar{\alpha} = \mathrm{BinarySampler}(\hat{\alpha}) \in \{0,1\}^{F}, \tag{3}$$

$$\hat{E}_{:,i} = \begin{cases} \bar{\alpha} \odot E_{:,i} & \text{if } \textit{training} \\ \hat{\alpha} \odot E_{:,i} & \text{if } \textit{inference} \end{cases}, i \in \{1, 2, \ldots, d\}. \tag{4}$$

GBDT Feature Gate

$$E = \text{FeatureTokenizer}(x) \in \mathbb{R}^{F \times d}, \tag{1}$$

$$\alpha = \text{GBDTFeatureFrequency}(x) \in \mathbb{R}^{F}, \tag{2}$$

$$\hat{\alpha} = \alpha / T \in \mathbb{R}^{F}, \bar{\alpha} = \text{BinarySampler}(\hat{\alpha}) \in \{0, 1\}^{F}, \tag{3}$$

$$\hat{E}_{:,i} = \begin{cases} \bar{\alpha} \odot E_{:,i} & \text{if } \textit{training} \\ \hat{\alpha} \odot E_{:,i} & \text{if } \textit{inference} \end{cases}, i \in \{1, 2, \ldots, d\}. \tag{4}$$

using sparse feature masks from real GBDT feature access probabilities to perform **hard feature selection du**

training

GBDT Feature Gate

$$E = \text{FeatureTokenizer}(x) \in \mathbb{R}^{F \times d}, \tag{1}$$

$$\alpha = \text{GBDTFeatureFrequency}(x) \in \mathbb{R}^{F}, \tag{2}$$

$$\hat{\alpha} = \alpha / T \in \mathbb{R}^{F}, \bar{\alpha} = \text{BinarySampler}(\hat{\alpha}) \in \{0, 1\}^{F}, \tag{3}$$

$$\hat{E}_{:,i} = \begin{cases} \bar{\alpha} \odot E_{:,i} & \text{if } \textit{training} \\ \boxed{\hat{\alpha} \odot E_{:,i} \quad \text{if } \textit{inference}} \end{cases}, i \in \{1, 2, \ldots, d\}. \tag{4}$$

use the **soft probabilities** during **inference** for deterministic prediction.

GBDT Feature Gate

$$\hat{E}_{:,i} = \begin{cases} \bar{\alpha} \odot E_{:,i} & \text{if } \textit{training} \\ \hat{\alpha} \odot E_{:,i} & \text{if } \textit{inference} \end{cases} \quad , i \in \{1, 2, \ldots, d\} \, . \tag{4}$$

Difference between hard feature selection and soft probabilities:

α = [2, 1, 2]

$\hat{\alpha}$ = [0.66, 0.33, 0.66]

Hard feature selection:

Given threshold is 0.5, then downstream processes will only use feature1 and feature3

Soft probabilities:

downstream processes will use all features, but depends on probabilities.

GBDT Feature Gate

Quick initialization and training of the GFG:

The authors mention that they use a **fast-trained XGBoost** model to initialize GFG and embed it into the T-MLP model. Even with **default hyperparameters**, this GBDT model is sufficient to guide feature selection without significantly impacting overall feature preference in subsequent training.

Pure MLP Basic Block

. **Inspired by vision MLPs**: The authors observed that a key success factor for MLP architectures in vision (e.g., for image processing) is their use of **linear projections** and "**attention-like**" mechanisms for **feature interactions**.

Pure MLP Basic Block

. To emulate this effect, the authors introduce a unit called SGU (Spatial Gating Unit) to perform feature-level operations.The block is similar to a single **feed-forward neural network (FFN)** in the **transformer** with an extra SGU (Eq. (6)) for feature-level in teraction.

$$\hat{E}^{(l+1)} = \mathrm{SGU}(\mathrm{GELU}(\mathrm{LayerNorm}(\hat{E}^{(l)})W_1))W_2 + \hat{E}^{(l)}$$

$$\mathrm{SGU}(X) = W_3 \mathrm{LayerNorm}(X_{:,:d'}) \odot X_{:,d':}\ .$$

$$X \in \mathbb{R}^{F\times 2d'} \quad W_1 \in \mathbb{R}^{d\times 2d'} \quad W_2 \in \mathbb{R}^{d'\times d} \quad W_3 \in \mathbb{R}^{F\times F}$$

d' corresponds to the FFN intermediate dimension size.

Pure MLP Basic Block

$$\hat{E}^{(l+1)} = \mathrm{SGU}\boxed{(\mathrm{GELU}(\mathrm{LayerNorm}(\hat{E}^{(l)})W_1))W_2} + \hat{E}^{(l)}$$

$$\mathrm{SGU}(X) = W_3 \mathrm{LayerNorm}(X_{:,:d'}) \odot X_{:,d':} \,.$$

$$X \in \mathbb{R}^{F \times 2d'} \quad W_1 \in \mathbb{R}^{d \times 2d'} \quad W_2 \in \mathbb{R}^{d' \times d} \quad W_3 \in \mathbb{R}^{F \times F}$$

The input features are transformed using the GELU activation function and LayerNorm, followed by linear transformations with two weight matrices, W1 and W2. This process is similar to the Feed-Forward Network (FFN) in Transformers.

Pure MLP Basic Block

$$\hat{E}^{(l+1)} = \mathrm{SGU}\boxed{(\mathrm{GELU}(\mathrm{LayerNorm}(\hat{E}^{(l)})W_1))W_2} + \hat{E}^{(l)}$$

$$\mathrm{SGU}(X) = W_3\mathrm{LayerNorm}(X_{:,:d'}) \odot X_{:,d':} \, .$$

Residual Connection

$$X \in \mathbb{R}^{F\times 2d'} \quad W_1 \in \mathbb{R}^{d\times 2d'} \quad W_2 \in \mathbb{R}^{d'\times d} \quad W_3 \in \mathbb{R}^{F\times F}$$

The input features are transformed using the GELU activation function and LayerNorm, followed by linear transformations with two weight matrices, W1 and W2. This process is similar to the Feed-Forward Network (FFN) in Transformers.

Pure MLP Basic Block

$$\hat{E}^{(l+1)} = \mathrm{SGU}(\mathrm{GELU}(\mathrm{LayerNorm}(\hat{E}^{(l)})W_1))W_2 + \hat{E}^{(l)}$$

$$\boxed{\mathrm{SGU}(X) = W_3 \mathrm{LayerNorm}(X_{:,:d'}) \odot X_{:,d':}}\,.$$

$$X \in \mathbb{R}^{F \times 2d'} \quad W_1 \in \mathbb{R}^{d \times 2d'} \quad W_2 \in \mathbb{R}^{d' \times d} \quad \boxed{W_3 \in \mathbb{R}^{F \times F}}$$

The SGU is used for feature-level interaction, where W3 performs a linear transformation on the features, similar to attention mechanisms. It generates gating effects by interacting with a portion of the input features X (It can determining each feature's contribution and enabling selective enhancement or suppression.)

Sparsity with User-controllable Pruning

The T-MLP model draws on the **pre-pruning** technique from GBDT, which controls model complexity and enhances generalization through **user-defined hyperparameters**, such as **maximum tree depth and minimum samples per leaf node**.

Sparsity with User-controllable Pruning

The T-MLP model draws on the **pre-pruning** technique from GBDT, which controls model complexity and enhances generalization through **user-defined hyperparameters**, such as **maximum tree depth and minimum samples per leaf node**.

Specifically, T-MLP introduces **two fine-grained variables** to **mask parameters** in the hidden and intermediate dimensions, allowing certain weights to be retained or removed during computation.

Sparsity with User-controllable Pruning

The T-MLP model draws on the pre-pruning technique from GBDT, which controls model complexity and enhances generalization through user-defined hyperparameters, such as maximum tree depth and minimum samples per leaf node.

Specifically, T-MLP introduces **two fine-grained variables** to **mask parameters** in the hidden and intermediate dimensions, allowing certain weights to be retained or removed during computation.

$$z_{\mathrm{h}} \in \{0,1\}^{d} \qquad \mathrm{diag}(z_{\mathrm{h}})W_1 \qquad W_1 \in \mathbb{R}^{d \times 2d'}$$

$$z_{\mathrm{in}} \in \{0,1\}^{d'} \qquad \mathrm{diag}(z_{\mathrm{in}})W_2 \qquad W_2 \in \mathbb{R}^{d' \times d}$$

$$\hat{E}^{(l+1)} = \mathrm{SGU}(\mathrm{GELU}(\mathrm{LayerNorm}(\hat{E}^{(l)})W_1))W_2 + \hat{E}^{(l)}$$

Sparsity with User-controllable Pruning

Differences between TMLP’s user-controllable pruning and traditional tabular DNN pruning:

(1) Unlike previous tabular DNNs focused only on feature sparsity, T-MLP applies sparsity to **both input features and hidden dimensions**, addressing overlooked **over-parameterization**.

Sparsity with User-controllable Pruning

Differences between TMLP’s user-controllable pruning and traditional tabular DNN pruning:

(1) Unlike previous tabular DNNs focused only on feature sparsity, T-MLP applies sparsity to **both input features and hidden dimensions**, addressing overlooked **over-parameterization**.

(2) Existing models tie sparsity strictly to **prediction loss**, but T-MLP uses **user-defined sparsity rates** for independent, controllable pruning, similar to GBDT pre-pruning.

Since the sparsity rate is preset, T-MLP **avoids repeated pruning** and adjustments during training. This keeps the model structure **stable** and **maintains consistent sparsity**, preventing the **instability** or **over-pruning** seen in loss-driven pruning.

Overall Workflow and Efficient Ensemble

Input Processing: Tabular features are embedded with a feature tokenizer.

Overall Workflow and Efficient Ensemble

Input Processing: Tabular features are embedded with a feature tokenizer.

Feature Selection: Features are selected via the sampled feature mask **(GFG)**.

Overall Workflow and Efficient Ensemble

Input Processing: Tabular features are embedded with a feature tokenizer.

Feature Selection: Features are selected via the sampled feature mask **(GFG)**.

Pruned Block Processing: Features pass through a single pruned basic block **(SGU)**.

Overall Workflow and Efficient Ensemble

Input Processing: Tabular features are embedded with a feature tokenizer.

Feature Selection: Features are selected via the sampled feature mask **(GFG)**.

Pruned Block Processing: Features pass through a single pruned basic block **(SGU)**.

Parameter Masking: Pruning masks z_{h} and z_{in} are sampled using L0 regularization.

Overall Workflow and Efficient Ensemble

Input Processing: Tabular features are embedded with a feature tokenizer.

Feature Selection: Features are selected via the sampled feature mask **(GFG)**.

Pruned Block Processing: Features pass through a single pruned basic block **(SGU)**.

Parameter Masking: Pruning masks z_{h} and z_{in} are sampled using L0 regularization.

Prediction: The [CLS] token feature is used in a standard prediction head to generate the final prediction, **similar to other tabular Transformers**.

Overall Workflow and Efficient Ensemble

Input Processing: Tabular features are embedded with a feature tokenizer.

Feature Selection: Features are selected via the sampled feature mask **(GFG)**.

Pruned Block Processing: Features pass through a single pruned basic block **(SGU)**.

Parameter Masking: Pruning masks z_{h} and z_{in} are sampled using L0 regularization.

Prediction: The [CLS] token feature is used in a standard prediction head to generate the final prediction, **similar to other tabular Transformers**.

$$\hat{y} = \mathrm{FC}(\mathrm{ReLU}(\mathrm{LayerNorm}(\hat{E}^{(l)}_{[\mathrm{CLS}],:})))$$

FC denotes a fully connected layer. We use the cross entropy loss for classification and the MSE loss for regression as in previous tabular DNNs

Overall Workflow and Efficient Ensemble

Efficient Ensemble:

Ensemble Version: training three TMLP branches simultaneously.

Overall Workflow and Efficient Ensemble

Efficient Ensemble:

Ensemble Version: training three TMLP branches simultaneously.

Shared Feature Gate: All branches share the GBDT feature gate and start from the same initialization.

Overall Workflow and Efficient Ensemble

Efficient Ensemble:

Ensemble Version: training three TMLP branches simultaneously.

Shared Feature Gate: All branches share the GBDT feature gate and start from the same initialization.

Learning Rates: Each branch uses a distinct fixed learning rate.

Overall Workflow and Efficient Ensemble

Efficient Ensemble:

Ensemble Version: training three TMLP branches simultaneously.

Shared Feature Gate: All branches share the GBDT feature gate and start from the same initialization.

Learning Rates: Each branch uses a distinct fixed learning rate.

Ensemble Prediction: The final prediction is the average output of all branches, similar to a bagging ensemble.

Overall Workflow and Efficient Ensemble

Efficient Ensemble:

Ensemble Version: training three TMLP branches simultaneously.

Shared Feature Gate: All branches share the GBDT feature gate and start from the same initialization.

Learning Rates: Each branch uses a distinct fixed learning rate.

Ensemble Prediction: The final prediction is the average output of all branches, similar to a bagging ensemble.

Training Efficiency: Multi-processing enables simultaneous training, with training duration set by the slowest-converging branch.

Outline

- Introduction
- Related Work
- Tree-Hybrid Simple MLP
- **Experiments**
- Conclusions

Experimental Setup

using four recent high-quality tabular benchmarks:

- FT-Transformer (FT-T, 11 datasets)
- T2G-Former (T2G, 12 datasets)
- SAINT (26 datasets)
- Tabular Benchmark(TabBen, 39 datasets)

Learning rate:

single T-MLP: 1e-4

T-MLP(3): 1e-4, 5e-4, and 1e-3

Table 2: Dataset statistics on four experimental benchmarks. "# bin., # mul., and # reg." are the amounts of binary classification, multi-class classification, and regression datasets. "# small, # middle, # large, and # ex. large" represent the amounts of small ($N \leq 3K$), middle ($3K < N \leq 10K$), large ($10K < N \leq 100K$), and extremely large ($N > 100K$) datasets, where N denotes the training data size. "# wide and # ex. wide" are the amounts of wide ($32 < F \leq 64$) and extremely wide ($F > 64$) datasets, where F is the feature amount. "bin. metric, mul. metric, and reg. metric" represent the evaluation metrics used for each task type in the benchmarks. "R-Squared" score is the coefficient of determination.

	# bin.	# mul.	# reg.	# small	# middle	# large	# ex. large	# wide	# ex. wide	bin. metric	mul. metric	reg. metric
FT-T [22]	3	4	4	0	0	6	5	2	5	ACC	ACC	RMSE
T2G [62]	3	5	4	0	3	7	2	2	2	ACC	ACC	RMSE
SAINT [48]	9	7	10	10	3	12	1	6	9	AUC	ACC	RMSE
TabBen [23]	15	0	24	2	37	0	0	5	2	ACC	N/A	R-Squared

Framework preferences among the datasets

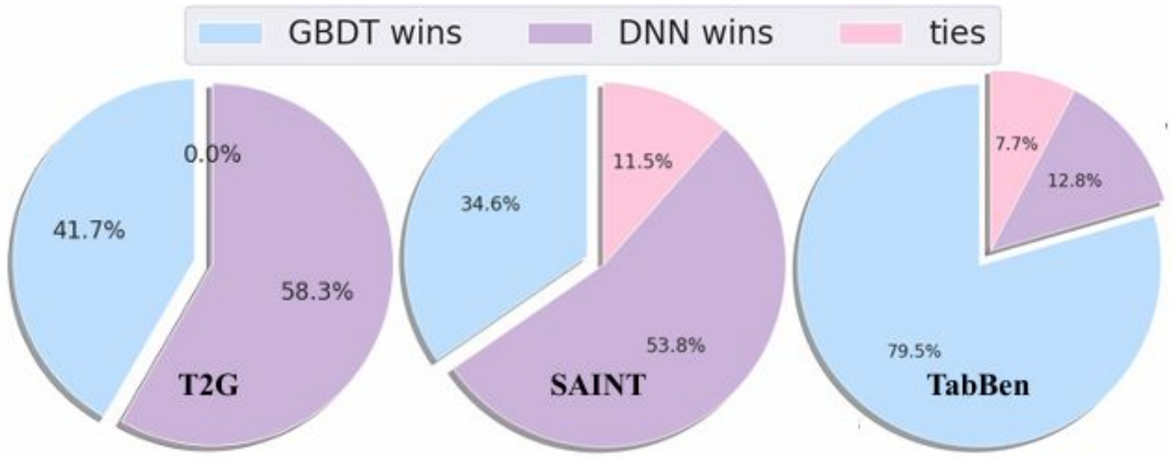


Figure 2: The winning rates of GBDTs and DNNs on three benchmarks, which represent the proportion of each framework achieving the best performance in the benchmarks. It exhibits varying framework preferences among the datasets used in different tabular prediction works.

Comparison with Advanced DNNs

Table 3: Cost-effectiveness comparison on the FT-T benchmark. Classification datasets and regression datasets are evaluated using the accuracy and RMSE metrics, respectively. "Rank" denotes the average values (standard deviations) of all the methods across the datasets. "T" represents the average overhead of the used training time against T-MLP, and "T^*" compares only the duration before achieving the best validation scores. All the training durations are estimated with the original hyperparameter search settings. "P" denotes the average parameter number of the best model configuration provided by the FT-T repository. TabNet is not compared considering its different backend (Tensorflow) in the evaluation. The top performances are marked in bold, and the second best ones are underlined (similar marks are used in the subsequent tables).

	CA ↓	AD ↑	HE ↑	JA ↑	HI ↑	AL ↑	EP ↑	YE ↓	CO ↑	YA ↓	MI ↓	Rank	T	T^*	P(M)
TabNet	0.510	0.850	0.378	0.723	0.719	0.954	0.8896	8.909	0.957	0.823	0.751	9.0 (1.5)	N/A	N/A	N/A
SNN	0.493	0.854	0.373	0.719	0.722	0.954	0.8975	8.895	0.961	0.761	0.751	7.8 (1.1)	×42.76	×24.87	1.12
AutoInt	0.474	0.859	0.372	0.721	0.725	0.945	0.8949	8.882	0.934	0.768	0.750	7.4 (2.1)	×121.68	×112.31	1.14
GrowNet	0.487	0.857	N/A	N/A	0.722	N/A	0.8970	8.827	N/A	0.765	0.751	N/A	N/A	N/A	N/A
MLP	0.499	0.852	0.383	0.719	0.723	0.954	0.8977	8.853	0.962	0.757	0.747	6.5 (1.7)	×27.41	×28.46	0.55
DCNv2	0.484	0.853	0.385	0.716	0.723	0.955	0.8977	8.890	0.965	0.757	0.749	6.4 (1.8)	×31.15	×40.65	4.17
NODE	0.464	0.858	0.359	0.727	0.726	0.918	0.8958	8.784	0.958	**0.753**	**0.745**	5.4 (3.2)	×386.54	×353.38	16.59
ResNet	0.486	0.854	**0.396**	0.728	0.727	**0.963**	0.8969	8.846	0.964	0.757	0.748	4.5 (2.2)	×56.20	×58.46	6.16
FT-T	0.459	0.859	0.391	**0.732**	0.720	0.960	**0.8982**	8.855	**0.970**	0.756	0.746	3.3 (2.4)	×117.35	×97.49	2.12
T-MLP	0.447	0.864	0.386	0.728	0.729	0.956	0.8977	8.768	0.968	0.756	0.747	3.1 (0.9)	×1.00	×1.00	0.79
T-MLP(3)	**0.438**	**0.867**	0.386	**0.732**	**0.730**	0.960	0.8978	**8.732**	0.969	0.755	**0.745**	**1.7 (0.8)**	×1.05	×1.08	2.37

Comparison with Advanced DNNs

Table 3: Cost-effectiveness comparison on the FT-T benchmark. Classification datasets and regression datasets are evaluated using the accuracy and RMSE metrics, respectively. "Rank" denotes the average values (standard deviations) of all the methods across the datasets. "T" represents the average overhead of the used training time against T-MLP, and "T^*" compares only the duration before achieving the best validation scores. All the training durations are estimated with the original hyperparameter search settings. "P" denotes the average parameter number of the best model configuration provided by the FT-T repository. TabNet is not compared considering its different backend (Tensorflow) in the evaluation. The top performances are marked in bold, and the second best ones are underlined (similar marks are used in the subsequent tables).

	CA ↓	AD ↑	HE ↑	JA ↑	HI ↑	AL ↑	EP ↑	YE ↓	CO ↑	YA ↓	MI ↓	Rank	T	T^*	P(M)
TabNet	0.510	0.850	0.378	0.723	0.719	0.954	0.8896	8.909	0.957	0.823	0.751	9.0 (1.5)	N/A	N/A	N/A
SNN	0.493	0.854	0.373	0.719	0.722	0.954	0.8975	8.895	0.961	0.761	0.751	7.8 (1.1)	×42.76	×24.87	1.12
AutoInt	0.474	0.859	0.372	0.721	0.725	0.945	0.8949	8.882	0.934	0.768	0.750	7.4 (2.1)	×121.68	×112.31	1.14
GrowNet	0.487	0.857	N/A	N/A	0.722	N/A	0.8970	8.827	N/A	0.765	0.751	N/A	N/A	N/A	N/A
MLP	0.499	0.852	0.383	0.719	0.723	0.954	0.8977	8.853	0.962	0.757	0.747	6.5 (1.7)	×27.41	×28.46	0.55
DCNv2	0.484	0.853	0.385	0.716	0.723	0.955	0.8977	8.890	0.965	0.757	0.749	6.4 (1.8)	×31.15	×40.65	4.17
NODE	0.464	0.858	0.359	0.727	0.726	0.918	0.8958	8.784	0.958	**0.753**	**0.745**	5.4 (3.2)	×386.54	×353.38	16.59
ResNet	0.486	0.854	**0.396**	0.728	0.727	**0.963**	0.8969	8.846	0.964	0.757	0.748	4.5 (2.2)	×56.20	×58.46	6.16
FT-T	0.459	0.859	0.391	**0.732**	0.720	0.960	**0.8982**	8.855	**0.970**	0.756	0.746	3.3 (2.4)	×117.35	×97.49	2.12
T-MLP	0.447	0.864	0.386	0.728	0.729	0.956	0.8977	8.768	0.968	0.756	0.747	3.1 (0.9)	×1.00	×1.00	0.79
T-MLP(3)	**0.438**	**0.867**	0.386	**0.732**	**0.730**	0.960	0.8978	**8.732**	0.969	0.755	**0.745**	**1.7 (0.8)**	×1.05	×1.08	2.37

Comparison with Advanced DNNs

Table 4: Cost-effectiveness comparison on the T2G benchmark with similar notations as in Table 3. The baseline performances and configurations are also reused from the T2G repository. According to the T2G paper, for the extremely large dataset Year, FT-T and T2G use 50-iteration hyperparameter tuning (HPT), DANet-28 follows its default hyperparameters, and the other baseline results are acquired with 100-iteration HPT.

	GE ↑	CH ↑	EY ↑	CA ↓	HO ↓	AD ↑	OT ↑	HE ↑	JA ↑	HI ↑	FB ↓	YE ↓	Rank	T	T^*	P(M)
XGBoost	0.684	0.859	0.725	**0.436**	3.169	**0.873**	**0.825**	0.375	0.719	0.724	**5.359**	8.850	4.3 (3.1)	×32.78	×42.88	N/A
MLP	0.586	0.858	0.611	0.499	3.173	0.854	0.810	0.384	0.720	0.720	5.943	8.849	8.3 (1.9)	×13.73	×11.45	0.64
SNN	0.647	0.857	0.616	0.498	3.207	0.854	0.812	0.372	0.719	0.722	5.892	8.901	8.3 (1.5)	×22.74	×12.54	0.82
TabNet	0.600	0.850	0.621	0.513	3.252	0.848	0.791	0.379	0.723	0.720	6.559	8.916	10.2 (2.4)	N/A	N/A	N/A
DANet-28	0.616	0.851	0.605	0.524	3.236	0.850	0.810	0.355	0.707	0.715	6.167	8.914	10.6 (2.0)	N/A	N/A	N/A
NODE	0.539	0.859	0.655	0.463	3.216	0.858	0.804	0.353	0.728	0.725	5.698	8.777	7.0 (3.0)	×329.79	×288.21	16.95
AutoInt	0.583	0.855	0.611	0.472	3.147	0.857	0.801	0.373	0.721	0.725	5.852	8.862	8.1 (2.0)	×68.30	×55.52	0.06
DCNv2	0.557	0.857	0.614	0.489	3.172	0.855	0.802	0.386	0.716	0.722	5.847	8.882	8.4 (2.0)	×24.40	×21.63	2.30
FT-T	0.613	0.861	0.708	0.460	3.124	0.857	0.813	**0.391**	0.732	0.731	6.079	8.852	4.7 (2.6)	×64.68	×50.90	2.22
T2G	0.656	0.863	**0.782**	0.455	3.138	0.860	0.819	**0.391**	**0.737**	**0.734**	5.701	8.851	3.1 (1.7)	×88.93	×87.04	1.19
T-MLP	0.706	0.862	0.717	0.449	3.125	0.864	0.814	0.386	0.728	0.729	5.667	8.768	3.3 (0.9)	×1.00	×1.00	0.72
T-MLP(3)	**0.714**	**0.866**	0.747	0.438	**3.063**	0.867	0.823	0.386	0.732	0.730	5.629	**8.732**	**1.9 (0.8)**	×1.09	×1.11	2.16

Comparison with Pre-trained DNNs

Table 5: The average values (standard deviations) of all the method ranks on the SAINT benchmark of three task types. $|D|$ is the dataset number in each group. Notably, all the baseline results are based on HPT, and SAINT variants need further training budgets on pre-training and data augmentation. More detailed results are given in the Appendix.

Task Type:	Binclass ($\|D\|$=9)	Multiclass ($\|D\|$=7)	Regression ($\|D\|$=10)
RF	7.8 (3.3)	7.3 (2.2)	9.1 (4.2)
ExtraTrees	7.8 (3.8)	9.6 (1.9)	8.6 (3.5)
KNeighborsDist	13.7 (0.7)	11.6 (3.5)	12.9 (1.8)
KNeighborsUnif	14.4 (0.5)	12.4 (3.4)	14.0 (1.0)
LightGBM	5.7 (3.3)	3.9 (2.8)	6.5 (3.2)
XGBoost	4.2 (2.8)	6.7 (3.5)	7.3 (2.9)
CatBoost	**3.9 (2.8)**	7.2 (2.4)	5.6 (2.7)
MLP	10.7 (1.8)	10.1 (3.9)	N/A
NeuralNetFastAI	N/A	N/A	11.9 (2.2)
TabNet	13.2 (2.0)	13.5 (1.1)	10.2 (4.5)
TabTransformer	10.8 (1.4)	10.0 (3.6)	10.0 (2.9)
SAINT-s	7.8 (2.4)	7.9 (6.1)	4.7 (3.8)
SAINT-i	7.2 (2.6)	7.1 (2.7)	5.9 (3.5)
SAINT	4.2 (2.7)	5.2 (2.2)	**4.2 (2.3)**
T-MLP	4.6 (2.8)	4.6 (3.0)	4.6 (3.3)
T-MLP(3)	**3.9 (1.9)**	**2.9 (2.5)**	5.0 (2.9)

Comparison with Extensively Tuned GBDTs

Table 6: The average values (standard deviations) of all the method ranks on TabBen (four dataset types). "Num." and "Cat." denote numerical datasets (all features are numerical) and categorical datasets (some features are categorical), respectively. "Classif." and "Reg." denote classification and regression tasks. "Num. Reg." group includes only results of regression on numerical datasets (similar notations are for the others). $|D|$ is the dataset number in each group. Baseline test results are obtained based on the best validation results during ~400 iterations of HPT (according to the TabBen paper and repository). Detailed results are given in the Appendix.

Dataset Type:	Num. Classif. ($\|D\|$=9)	Num. Reg. ($\|D\|$=14)	Cat. Classif. ($\|D\|$=6)	Cat. Reg. ($\|D\|$=10)
MLP	8.4 (0.8)	N/A	N/A	N/A
ResNet	6.9 (1.9)	6.5 (1.9)	7.8 (1.0)	7.7 (0.5)
FT-T	5.7 (1.9)	5.5 (2.3)	5.5 (2.2)	6.7 (1.1)
SAINT	6.9 (1.4)	5.5 (2.2)	8.0 (1.1)	N/A
GBT	4.7 (2.0)	4.3 (1.7)	5.2 (2.3)	4.3 (1.1)
HistGBT	N/A	N/A	5.2 (2.3)	4.3 (1.3)
RF	4.6 (2.1)	4.8 (2.2)	4.0 (3.2)	5.8 (1.9)
XGBoost	2.6 (1.4)	**2.4 (1.5)**	**2.8 (1.5)**	2.1 (1.0)
T-MLP	3.2 (1.6)	4.3 (1.9)	3.5 (2.3)	3.6 (1.4)
T-MLP(3)	**2.1 (1.4)**	2.7 (1.5)	3.0 (1.3)	**1.8 (0.7)**

Main ablation and comparison

Table 7: Main ablation and comparison on classical tables in various task types and data scales. The top 4 rows: ablations on key designs in the T-MLP framework. The bottom 2 rows: results of T-MLP with neural network feature gate (NN FG).

Dataset:	CA (21K) ↓	AD (49K) ↑	HI (98K) ↑	YE (515K) ↓
T-MLP	0.4471	0.864	0.729	8.768
w/o sparsity	0.4503	0.857	0.726	8.887
w/o GBDT FG	0.4539	0.859	0.728	8.799
w/o both	0.4602	0.856	0.724	8.896
T-MLP (NN FG)	0.4559	0.852	0.718	8.925
w/o sparsity	0.4557	0.840	0.713	8.936

Main ablation and comparison

Authors notice a recent attempt on sample specific sparsity for biomedical tables using a **gating network**; it was originally designed for **low-sample-size** tabular settings and helped prediction interpretability in the **biomedical domain.**

Dataset:	CA (21K) ↓	AD (49K) ↑	HI (98K) ↑	YE (515K) ↓
T-MLP	0.4471	0.864	0.729	8.768
w/o sparsity	0.4503	0.857	0.726	8.887
w/o GBDT FG	0.4539	0.859	0.728	8.799
w/o both	0.4602	0.856	0.724	8.896
T-MLP (NN FG)	0.4559	0.852	0.718	8.925
w/o sparsity	0.4557	0.840	0.713	8.936

Main ablation and comparison

Complex Structure Needs: Large datasets require complex models for effective feature selection.

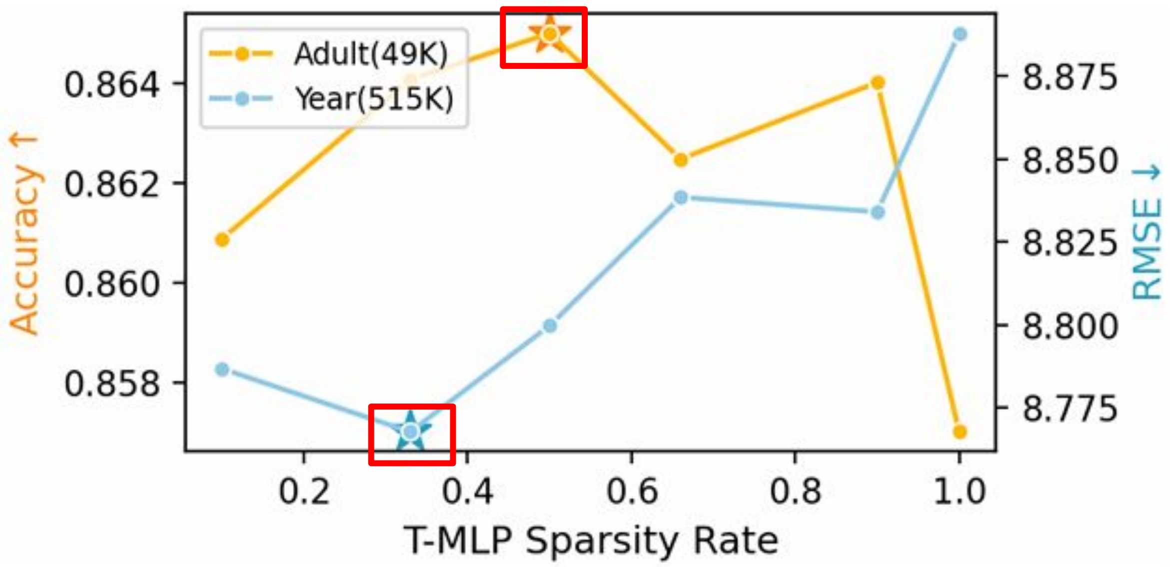
Optimization Compatibility: Discrete selection doesn't align well with neural networks' smooth optimization.

Confirmation Bias Risk: Neural networks may reinforce incorrect patterns, leading to poorly guided feature selection.

Dataset:	CA (21K) ↓	AD (49K) ↑	HI (98K) ↑	YE (515K) ↓
T-MLP	0.4471	0.864	0.729	8.768
w/o sparsity	0.4503	0.857	0.726	8.887
w/o GBDT FG	0.4539	0.859	0.728	8.799
w/o both	0.4602	0.856	0.724	8.896
T-MLP (NN FG)	0.4559	0.852	0.718	8.925
w/o sparsity	0.4557	0.840	0.713	8.936

hurts the performance as **data scales increase**

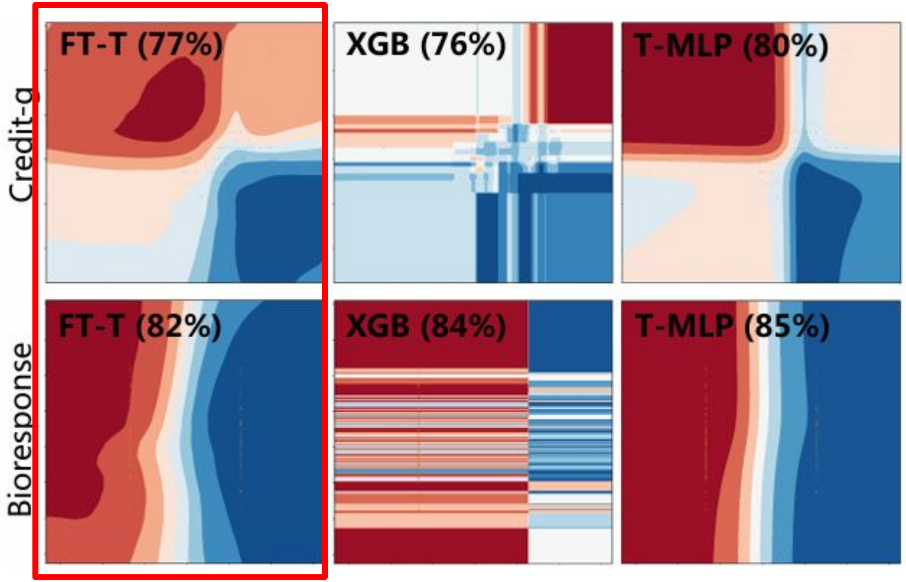
Sparsity Promotes Tabular DNNs



suitable model sparsity often **promotes** tabular prediction, but both excessive and insufficient sparsity cannot achieve the best results

Superiority Interpretability of T-MLP

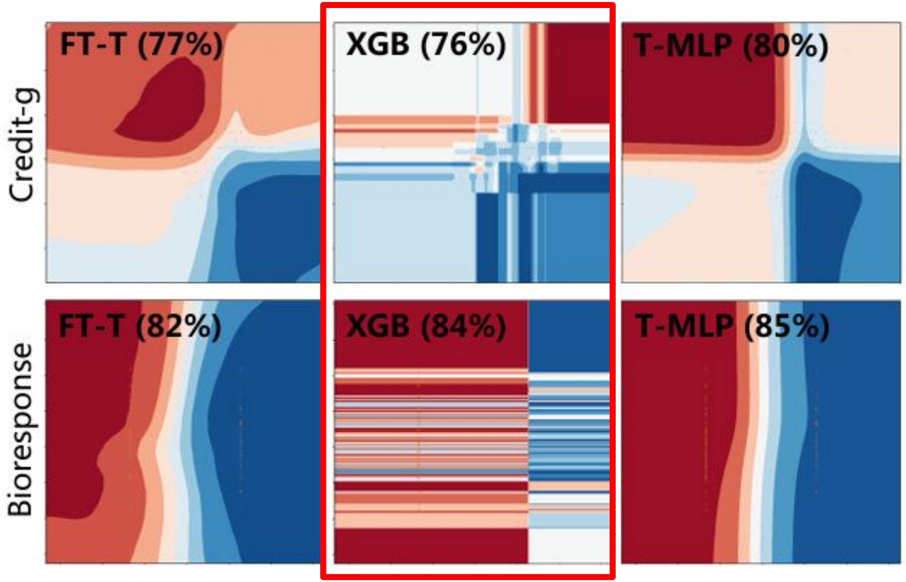
visualize decision boundaries



DNNs capture only main patterns and may miss fine-grained sub-patterns, leading to overfitting.

Superiority Interpretability of T-MLP

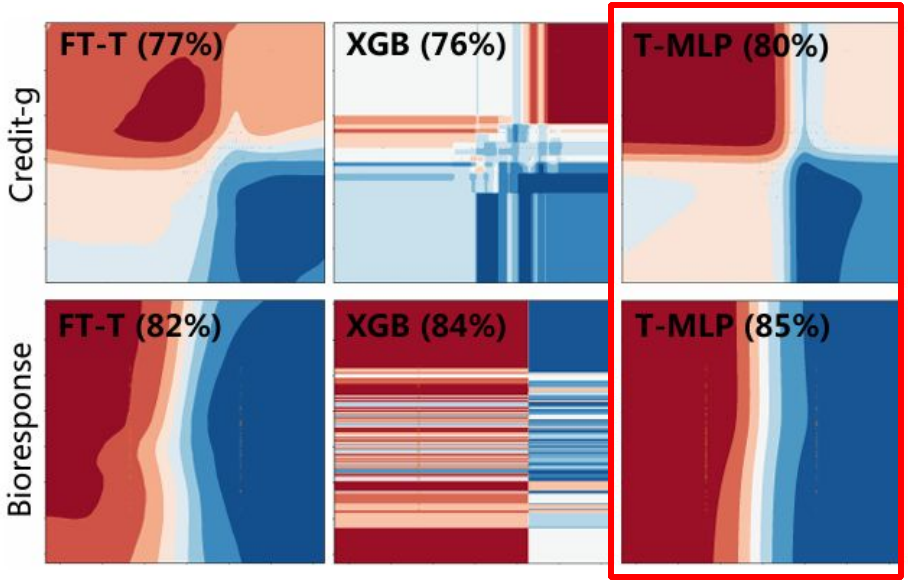
visualize decision boundaries



GBDT decision boundaries are often jagged with excessive splits, making them appear irregular and prone to overfitting noise in the data.

Superiority Interpretability of T-MLP

visualize decision boundaries



Different from common DNNs and GBDTs, T-MLP exhibits a **novel intermediate pattern** that combines characteristics from both DNNs and GBDTs. T-MLP combines GBDT's rule-based splits (e.g., vertical or horizontal lines) with DNN's smoothing, giving it an edge in avoiding overfitting.

Outline

- Introduction
- Related Work
- Tree-Hybrid Simple MLP
- Experiments
- **Conclusions**

Conclusion

T-MLP Overview: A hybrid framework that combines GBDTs and DNNs for effective tabular prediction.

Key Components:

- GBDT feature gate (GFG).
- DNN pruning techniques (SGU and User-controllable pruning).
- Original DNN back-propagation optimizer.

Performance: Achieves competitive results on diverse benchmarks with reduced runtime.

Applications: T-MLP offers a practical, economical solution for tabular prediction and supports research in hybrid tabular models.

謝謝大家~