NEURAL INFORMATION PROCESSING SYSTEMS

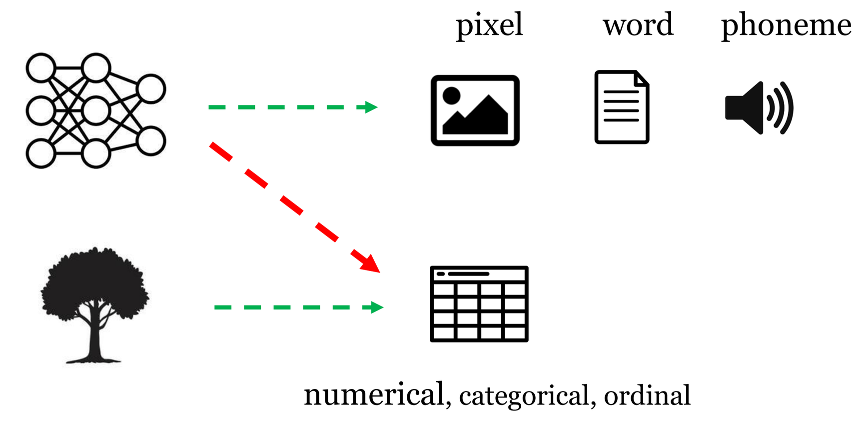
TRL@NeurIPS'23

Tree-Regularized Tabular Embeddings

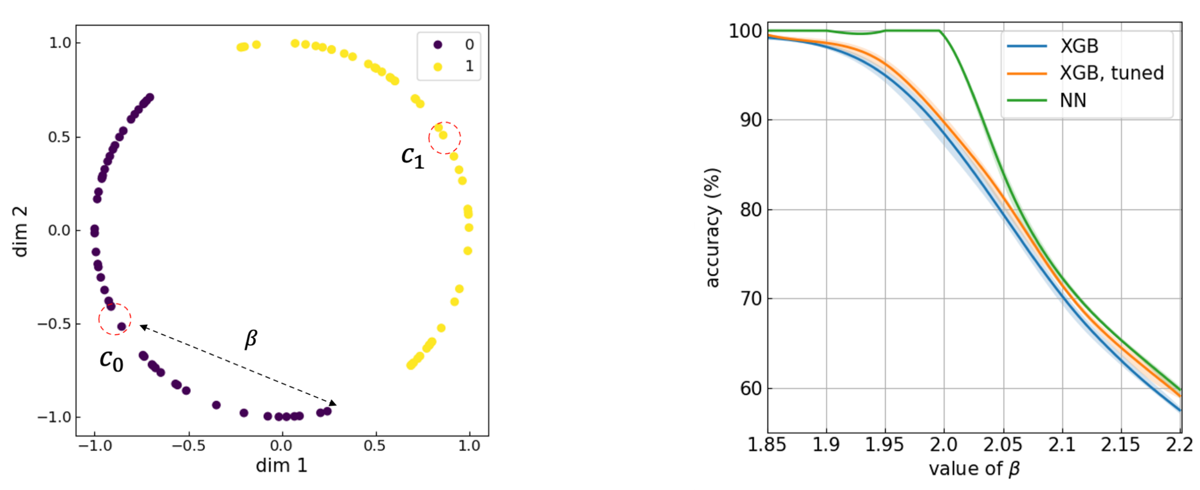
Xuan Li*, Yun Wang, Bo Li

Data-Centric Tabular Learning

Goal: taper the performance gap between tree-based and NN models on tabular data.

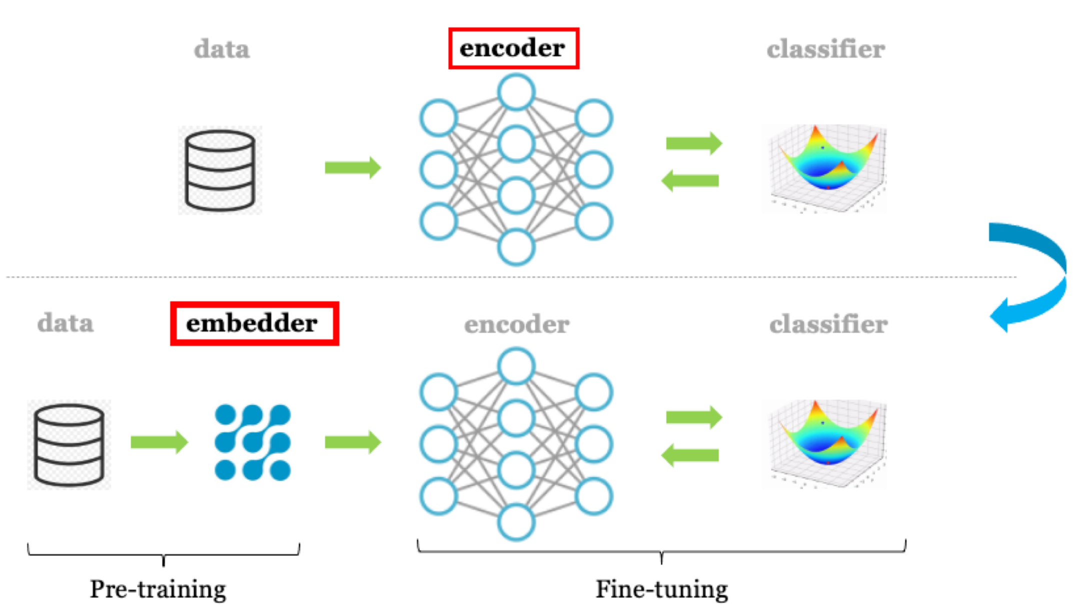


Limitation: tabular data are heterogenous in nature, and an underemphasis on feature alignment could overshadow the efficacy of NN.



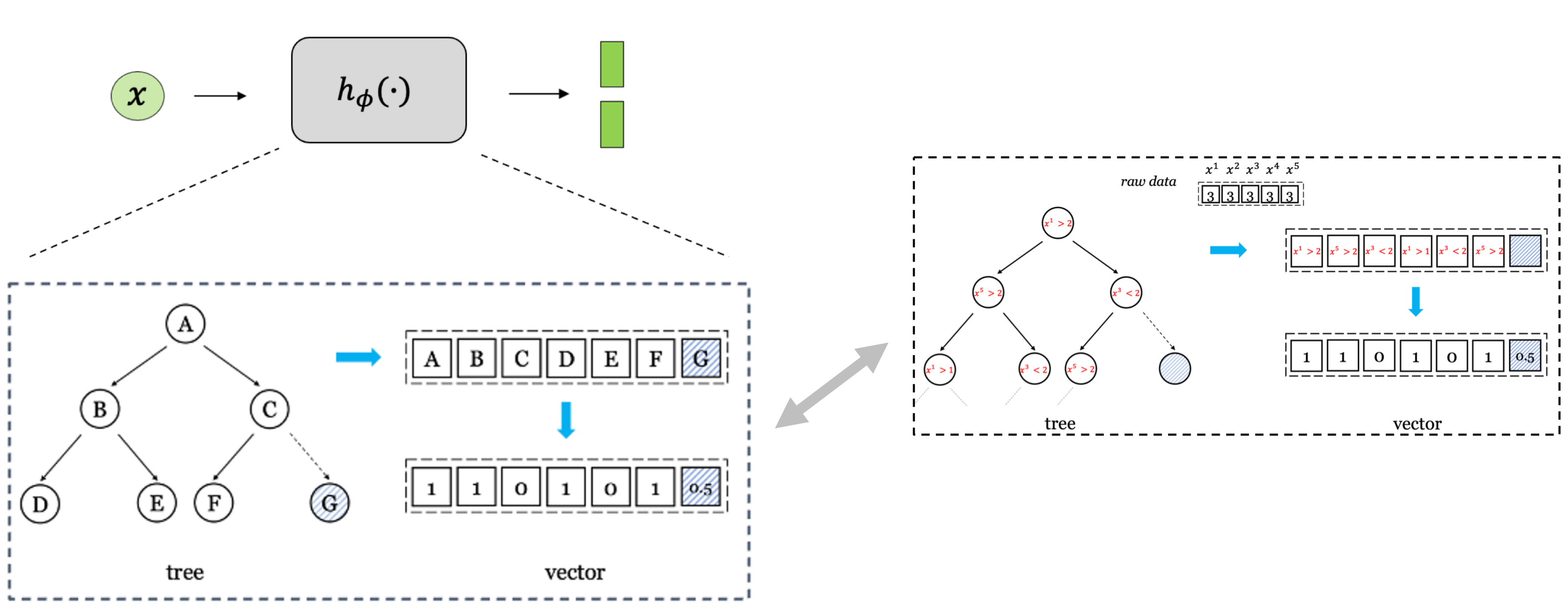
synthetic experiments: NN > tree in homogeneous space

Proposal: calibrate tabular data to fit NN from a data-centric perspective.



In-Batch Tree-Regularized Embeddings

Overview:
- binarize representations through pairwise comparison between variable values and thresholds in tree nodes.
- reformulated as a single vector (T2V) [1] or an array of tokens (T2T) for MLP and transformer blocks.



T2T tokens: generated through level-order traversal with padding

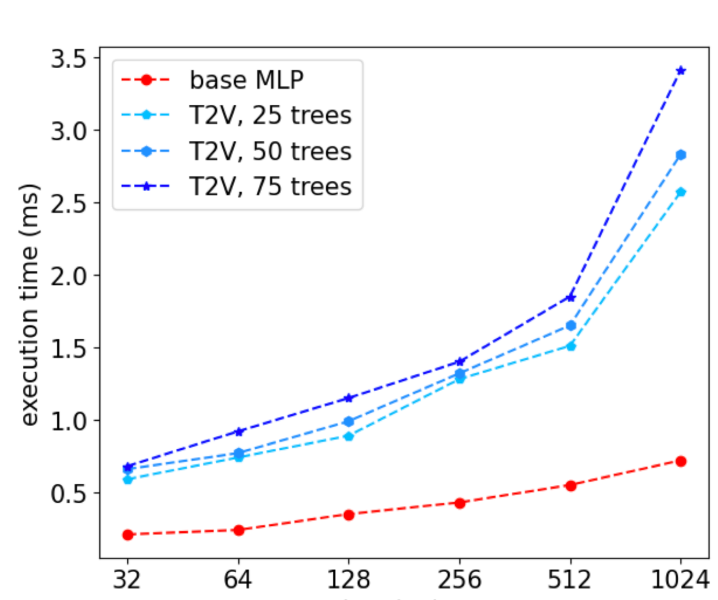
Implementation: in-batch transformation, supporting industrial use cases with hundreds of columns and millions of rows.

```python
class TreeToVectorSimple:
    def __init__(self, xgbTree, dtype=torch.float, device='cpu'):
        self.xgbTree = xgbTree
        self.dtype = dtype
        self.device = device

    def __call__(self, tensor):
        output = self.tree_encoder(tensor)
        return output

    def tree_encoder(self, tensor):
        # fill nan with -1
        tensor = torch.nan_to_num(tensor, nan=-1.0)
        output = self.postprocessing(
                    tensor,
                    self.xgbTree.multiply_matrix,
                    self.xgbTree.offset_vector)
        return output

    def postprocessing(self, x, multiply_matrix, offset_vector):
        x = torch.matmul(x, multiply_matrix.to(self.device))
        x -= offset_vector.to(self.device)
        x[x > 0] = 1.0
        x[x < 0] = 0.0
        return x
```

1

```python
transform_batch = transforms.Compose([TreeToVectorSimple(xgbTree)])

# within each batch
X_train_batch = transform_batch(X_train_batch)
```

2



left: pseudocode of in-batch transformation with matrix manipulation
right: time complexity between T2V and vanilla features with MLP

Evaluations

Experiment results on 91 OpenML benchmark datasets [2] with binary classification task. Reported in percentaged AUC.

Robustness

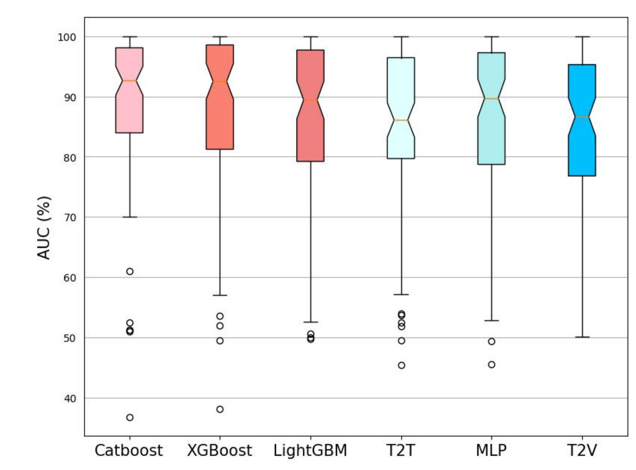
CatBoost	XGBoost	LightGBM	T2V	T2T	MLP	SAINT	ResNet
91	91	91	88	88	88	59	73

 # datasets can be evaluated

Comparison w.r.t. tree-based models

Algorithm	Rank ↓				AUC (%) ↑
	min	max	mean	median	mean
CatBoost	1	6	2.38	2	88.06
XGBoost	1	6	2.83	2	87.70
LightGBM	1	6	3.16	3	86.37
T2T	1	6	4.07	4	84.63
MLP	1	6	4.22	4	84.42
T2V	1	6	4.45	5	83.15

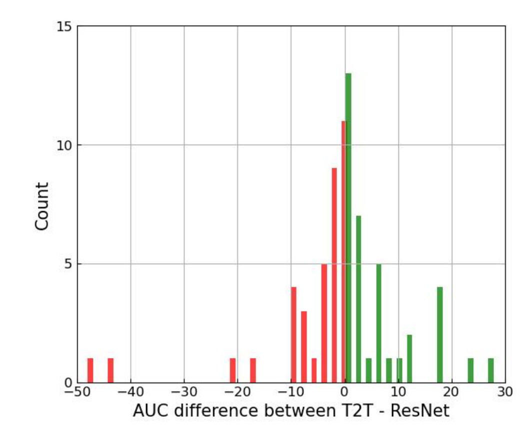
ranked by average AUC



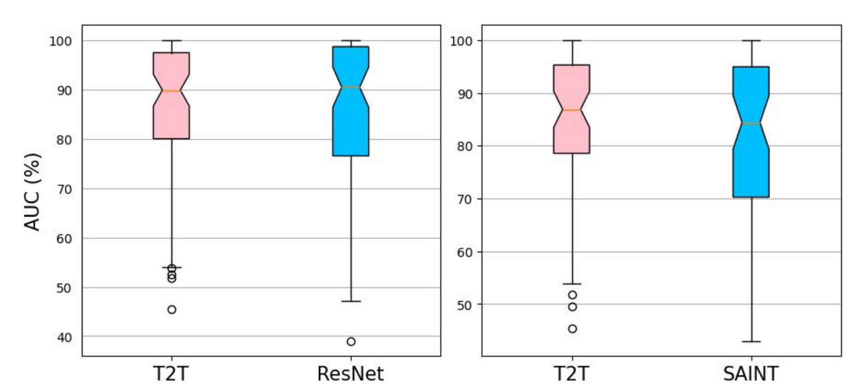
distribution of AUC

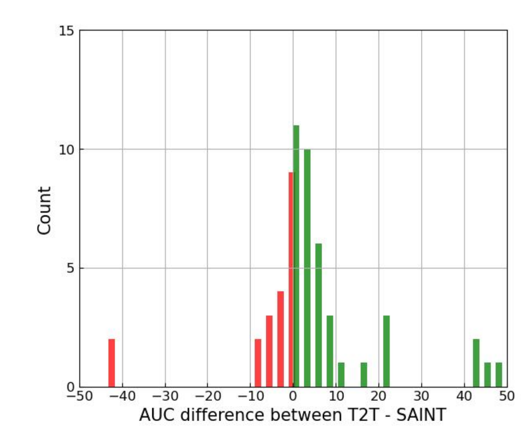
Comparison w.r.t. NN models

Algorithm	Rank ↓				AUC (%) ↑
	min	max	mean	median	mean
ResNet	1	4	2.15	2	84.87
T2T	1	4	2.29	2	84.72
T2V	1	4	2.61	3	83.92
SAINT	1	4	3.01	3	81.46



histogram of T2T – ResNet / SAINT





Main Takeaways

1. Implemented scalable algorithms to obtain tree-regularized embeddings T2V and T2T. The latter is more performant and can serve as tabular tokenizer for multimodal learning with transformer-based framework.
2. Although not reported in the paper, T2V with 4-layered transformer scales and outperforms tree-based models on production binary classification tasks. Interestingly, similar results are also observed in [3].
3. Future works: generalize to regression and multi-class classification tasks; explore consistent encoding of numerical and categorical features; call for industrial-scale benchmark datasets.

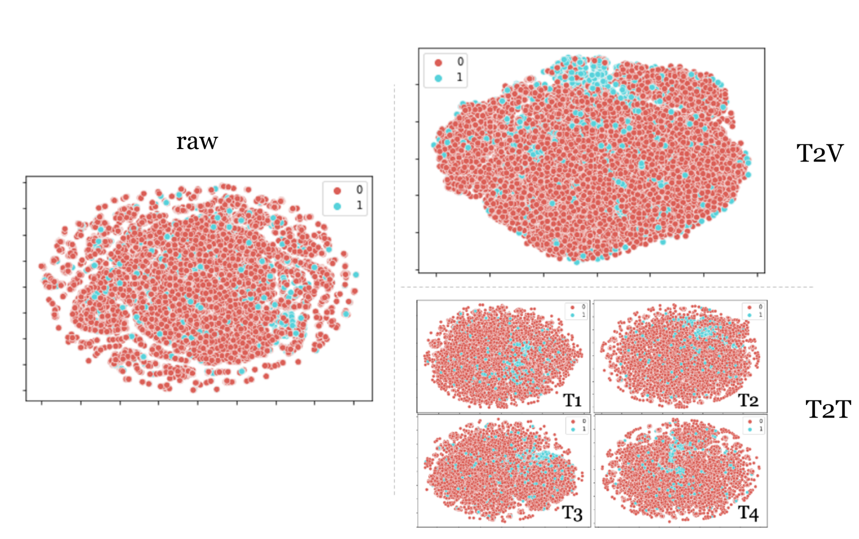
Appendix

Algorithm 1: Tree to Vector (T2V)

Input: xgb_trees, ϵ
Output: emb_map
Init: emb_map = {}
for tree ∈ xgb_trees **do**
  for node ∈ tree **do**
    {var_key, var_val} = node;
    var_val.round(ϵ);
    if {var_key, var_val} ∉ emb_map **then**
      emb_map[var_key].append(var_val);
    end
  end
end

Algorithm 2: Tree to Tokens (T2T)

Input: xgb_trees, τ, η
Output: emb_vec
Init: vec_len = 0, emb_vec = []
for tree ∈ xgb_trees **do**
  l = tree.count_node() ;
  vec_len = max(vec_len, l)
end
for tree ∈ xgb_trees **do**
  vec = tree.to_vec(τ);
  vec.pad(vec_len, η);
  emb_vec.append(vec);
end



left: pseudocode of T2V and T2T algorithm
right: t-SNE plot of raw, T2V and T2T embedding on internal dataset

References
[1] Vadim Borisov et al. "DeepTLF: robust deep neural networks for heterogeneous tabular data"
[2] Duncan McElfresh et al. "When Do Neural Nets Outperform Boosted Trees on Tabular Data?"
[3] Hu, X., et al. "Deepeta: How uber predicts arrival times using deep learning."