Trees with Attention for Set Prediction Tasks

A. Set-Tree model

In this appendix we provide proofs and additional details about Set-Trees that did not fit in the main paper due to space limitations.

A.1. Set-compatible split criteria

Table 1 summarizes some familiar statistical functions from the family of set-compatible split criteria presented in this study:

$$|\mathbf{S}|^{-\beta} \sum_{x \in \mathbf{S}} (x_j)^{\alpha} \geq \theta$$

Table 1. Examples of some of the familiar statistical functions that can be used as split criteria.

α	β	Operator	α	β	Operator
0	0	size	-1	1	harmonic mean
1	0	sum	0	1	geometric mean
1	1	average	∞	0/1	max
2	1	variance	-∞	0/1	min

A.2. Theoretical properties

In the following, we introduce additional proofs of the theorems presented in the paper.

Theorem 2. This theorem can be proved by a reduction to the bin packing problem which is NP-complete. Recall that in the bin packing problem a finite set of items $x_1, \ldots, x_n \in \mathbb{Z}^+$ is given, together with a bin capacity B and the number of bins k. The goal is to determine if the items can be partitioned into k groups such that the sum of the items in each group is at most B.

This problem can be converted into a set function in the following way, we define the set:

$$\mathbf{S} = \{(x_i, 0, 0)\}_{i=1}^{n} \cup \{(0, k, 0), (0, 0, B)\}$$

The function f is evaluated to True if the set of values appearing in the first coordinates of $\mathbf{S}$ can be partitioned into k bins of size at most B where k is the maximal value of the second element in every item in $\mathbf{S}$ and B is the maximal value of the 3^{rd} coordinate of all the elements. It is easy to verify that f is a set function. If there is a circuit $C \in \mathbb{C}$ such that C can compute f then C determines the bin packing problem. However, C is a finite circuit and therefore has finite number of gates. Each gate has computational complexity that is polynomial in the size of its input which is $3(n+2)$. Therefore, the circuit determines the bin-packing problem in time that is $|C|\text{poly}(n)$ which contradicts the assumption that $P \neq NP$. □

Lemma 1. *If T_1 and T_2 are trees then there exists trees T^+ and $T^\times$ such that T^+ computes the function $T_1 + T_2$ and $T^\times$ computes the function $T_1 T_2$.*

Proof. The tree $T^\times$ can be generated using a tensor operation: replace every leaf of T_1 by a copy of the tree T_2. Multiply the leafs' values of the copy of T_2 by the corresponding T_1 leaf's value. The tree T^+ can be generated using a similar tensor operation where we add the value of the corresponding leaf of T_1 to the leafs' values of each copy of T_2 .

The correction of this construction follows since there is one-to-one mapping from the leafs of the tensored tree and the product/sum of the leafs of T_1 and T_2. An example x arrives at a leaf (l_1, l_2) in the tensored tree iff it arrives in the leaf l_1 in T_1 and in the leaf l_2 in T_2. Therefore, by providing the right value at the leafs of the tensored tree we obtain the stated results. □

B. Experiments details

In this appendix we provide additional information about the experiments to allow reproducibility, information that did not fit in the main paper due to space limitations.

For all tree-based models (GBT and GBeST) and for all tasks, we scanned a pre-defined hyperparameter search space. Unless specified otherwise, we used 10% of the train records as validation set. We used early-stopping when training all of the tree-based models with patience=5 and tolerance=0.001. We report the test results for the configuration that performed the best over the validation set.

The trees' maximal depth (d) was chosen within $\{5, 6, 8, 10\}$, the number of estimators (N_{est}) was chosen within $\{50, 100, 200, 300\}$ and the learning rate (lr) was chosen within $\{0.2, 0.1, 0.05\}$. We also applied known tree-regularization techniques, the fraction of train records sam-

pled per tree (f_s) was chosen within $\{1, 0.8, 0.5\}$ and the fraction of features sampled per tree (f_f) was chosen within $\{1, 0.8, 0.5\}$ (where 1 is using all the records).

The group of split criteria used is: $\mathbf{O} = \{$max, min, mean, sum, harmonic mean, geometric mean, second moment mean$\}$. Unless specified otherwise we limit the attention-set to either be the entire set or one that is generated in the last 5 nodes leading to the current node. The attention-set limit is marked as: $AS_{limit} = 5$. In a few particular cases, we changed the search space, those cases are detailed below (for example, the point clouds dataset, ModelNet40, contains only three spatial features per item, therefore we used only $f_f = 1$). We repeated every experiment five times using different seeds for randomized initialization. When possible, each initialization is using a different train/test split of the dataset. We report the mean metrics and the standard deviation.

For all the experiments we used a machine with 2 Intel Xeon Silver 4114 @ 2.20GHz CPUs. The deep-learning baselines were conducted using PyTorch 1.4.0 (Paszke et al., 2019), and trained with the Adam optimizer (Kingma & Ba, 2014) on NVIDIA GeForce RTX 2080 Ti GPU. GBT baselines were conducted using XGBoost 1.1.1 (Chen & Guestrin, 2016). The following sections provide additional details for the data processing, training, and evaluation protocols of the reported experiments.

B.1. First Quadrant

Models and training: GBeST used $lr = 0.1$, $d = 6/10$, $f_s = 0.5/0.8$ correspond to the 2D and to the 100D configurations. GBT used $lr = 0.05$, $d = 5$, $f_s = 1$ for the 2D configuration and $lr = 0.1$, $d = 8$, $f_s = 0.8$ for the 100D configuration. We used DeepSets permutation-invariant framework where the first component θ is a 3-layer MLP with sizes (64, 128, 128) with Relu activation and dropout of $p = 0.2$. The second component ρ is a 2-layer MLP with hidden size of 128. The models were trained for 50 epochs with batch size of 128 and early stopping with a patience of 3 epochs. The recurrent model contained an input projection layer with 64 hidden units, an LSTM cell with 32 hidden units and an output projection layer. It was trained for 50 epochs with batch size of 128 witht the same early stopping methodology.

B.2. Drug Prescription Errors

Data preparation: The raw (Johnson et al., 2016) dataset contained 50,212 hospital stays. As described in the paper, we limited the number of valid drug prescriptions in a set to be between 2 to 100 and excluded drugs that were prescribed fewer than 10 times or more than 10K times. After applying those filtering operations we ended up with 49,811 hospital stays and a catalog of 1,946 unique drugs.

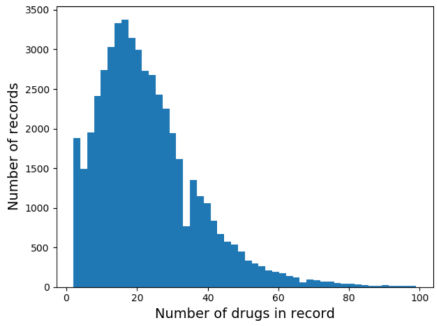

Figure 1. Distribution of sets sizes for drug prescription error prediction, derived from MIMIC-III dataset.

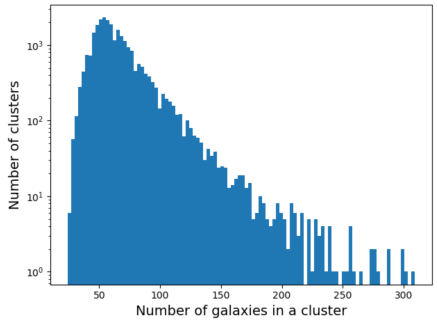

Figure 2. Distribution of galaxies clusters sizes from the RedMapper DR8 dataset.

We randomly split the patients to train and test folds (90%/10%) such that the patients in the train and test sets are disjoint. We generated five randomly partitioned views of the data for training and evaluation. We report the mean results for the five views of the dataset. In Figure 1 we present the distribution of drug prescription sets sizes used in this task.

Models and training: GBT used $lr = 0.1$, $d = 8$, $f_s = 1$, $f_f = 0.8$ for *MIR* setup and $lr = 0.1$, $d = 6$, $f_s = 1$, $f_f = 0.8$ for *SIR* setup. GBeST used $lr = 0.05$, $d = 6$, $f_s = 0.8$, $f_f = 1$ for *MIR* setup and $lr = 0.1$, $d = 5$, $f_s = 1$, $f_f = 0.5$ for *SIR* setup. We used DeepSets invariant framework where the first component θ is a 3-layer MLP with hidden states (64 , 128, 64) with Relu activation and dropout of $p = 0.2$. The second component ρ is a 3-layer MLP with hidden sizes (64, 32, 16). The bidirectional recurrent neural networks had two linear projection layers with shape: (32, 64), the RNN cell hidden size is 32 and an output hidden layer of 64. All DNN models were trained for 30 epochs, with Adam optimizer with a learning rate of 1e-3 and a batch size of 64. We used early stopping while monitoring the validation loss with a patience of 3 epochs.

B.3. Redshift Estimation in Galaxies Clusters

Data preparation: We used the galaxies clusters dataset from the RedMaPPer DR8 catalog (Rykoff et al., 2014). It contains the photometric features for 26,111 red galaxy clusters. 70,505 of the galaxies has corresponding spectroscopic redshift. We processed the raw data using the Astropy package (Astropy Collaboration & Robitaille, 2013). We followed the same process as in (Zaheer et al., 2017) and extracted the following photometric features: u, g, r, i and z band measurement and their corresponding errors. We also used the raw image measurement and its error, the galaxies' location in the sky (in decimal degrees), their projected distance (as measured from earth) and the probability of each galaxy being the cluster center. Figure 2 visualizes the distribution of the extracted clusters sizes.

Models and training: GBT used $lr = 0.05$, $N_{est} = 50$, $d = 5$, $f_f = 0.5$ and $f_s = 0.5$. GBeST used $lr = 0.2$, $N_{est} = 100$, $d = 6$, $f_f = 0.8$ and $f_s = 0.8$. The DeepSets and the MLP results are taken from (Zaheer et al., 2017).

B.4. Jets Tagging

Data preparation: We considered two datasets for this task. *Quark-Gluon tagging* dataset comprised from jets initiated by quarks and by gluons. The number of particles in each jet varied between 1 to 148 and each particle is characterized by its four-momentum vector (px, py, pz, E). We followed the preprocessing pipeline of (Qu & Gouskos, 2020) and extract seven variables from the 4-momentum vector for each particle. We followed the splitting of 1.6M/200k/200k records for training, validation, and test. *Top Tagging* dataset includes jets derived from hadronically decaying top quarks. It contains 2M jets with varying lengths between 1 and 99. The original data is separated to 1.2M/400K/400K data splits correspond to train/validation/test. Each particle is identified by its 4-momentum vector.

Models and training: For *Quark-Gluon tagging* dataset, GBT used $lr = 0.1$, $d = 5$, $f_f = 0.5$ and $f_s = 1$. GBeST used used $lr = 0.2$, $d = 5$, $f_f = 1$ and $f_s = 0.8$. For *Top Tagging* dataset, GBT used $lr = 0.1$, $d = 6$, $f_f = 0.5$ and $f_s = 0.8$. GBeST used used $lr = 0.1$, $d = 10$, $f_f = 0.8$ and $f_s = 0.8$. The results for the reported deep-learning baselines PFN and ParticleJet are taken from (Qu & Gouskos, 2020).

B.5. Two-sample Hypothesis Testing

Models and training: Since the sets it this task has only a single feature, we used $f_f = 1$ for all the tree-based models. GBT used $lr = 0.05$, $d = 5$ and $f_s = 1$. GBeST used $lr = 0.1$, $d = 6$ and $f_s = 0.5$. The DeepSets models' first component θ is a 2-layer MLP with hidden sizes (25, 50) and the second component ρ is a 2-layer MLP with hidden sizes (25, 2). We trained the models with Adam optimized and a learning rate of 1e-3 for 30 epochs. We used batch size of 64 and applied early stopping with a patience of 5 epochs.

B.6. Point Cloud Classification

Models and training: For this task we set $f_f = 1$. GBeST used $lr = 0.1$, $N_{est} = 200$, $d = 6$ and $f_s = 1$ and for GBT $lr = 0.05$, $N_{est} = 200$, $d = 6$ and $f_s = 0.8$. DeepSets baseline results are taken from Zaheer et al. (2017).

B.7. Poker Hands Classification

Models and training: For this task we set $f_f = 1$. For the two setups, GBT used $lr = 0.2$, $d = 8$ and $f_s = 1$. GBeST used $lr = 0.2$, $d = 8$ and $f_s = 0.5$. DeepSets baseline comprised from two components. The first component θ is a MLP with hidden sizes of (32, 64, 128) and the second component ρ is a MLP with hidden size of 64. DeepSets models were trained with Adam optimized and with a learning rate of 1e-3 for 50 epochs.

C. Implementation

In this appendix, we report Set-Tree's runtime and compare it to the runtime of the baselines presented in the paper. The current implementation is sub-optimal in terms of runtime, it was primarily designed for research and prototyping. Nevertheless, we recognize the importance of reporting the runtime of the current implementation. We are currently developing a more efficient implementation of Set-Tree that will be available for the benefit of the community.

We implemented a vanilla decision tree in our tree-learning framework and we compare Set-Tree's runtime to this model (named DT). We also compare the runtime of Set-Tree against Sklearn (Pedregosa et al., 2011) and XGBoost (Chen & Guestrin, 2016). During the tree-learning process, Set-Tree uses caching in order to avoid excessive calculations of the statistical moments. The statistical moments for every node's attention-set are stored on a first-in-first-out (FIFO) memory queue. We also compare Set-Tree to an ablated version in which the caching mechanism is disabled (we name this baseline Set-Tree-NC).

For the runtime evaluation we used the synthetic data generated for the first quadrant experiment and a subset of the drugs prescription errors dataset generated from MIMIC-III database (Johnson et al., 2016). We generated 10K random records from the first quadrant dataset for the two configurations of 2D/100D, each record consisted of 20 items. We trained the models with the following hyperparameters: number of trees $N_{est} = 1$, maximal number of leafs $N_{leafs} = 10/50$ (for the two configurations), number of (α, β) pairs $N_{[\alpha,\beta]} = 7$ and the attention-set limit

Table 2. Runtime performances, measured in seconds. **OH** stands for overhead, the ratio between the model's runtime and the runtime of the baseline **DT**. **Set-Tree-NC** is an ablated version of Set-Tree without using the caching mechanism.

	First Quadrant				Drug Prescription Errors			
	2D		100D		SIR		MIR	
Model	**Runtime**	**OH**	**Runtime**	**OH**	**Runtime**	**OH**	**Runtime**	**OH**
Sklearn	1.292±1.5e-3	-	9.556±0.208	-	1.311±0.145	-	1.970±0.135	-
XGBoost	1.871±0.412	-	7.081±0.871	-	1.866±0.022	-	1.772±0.865	-
DT	1.487±0.011	-	16.158±0.644	-	2.812±0.261	-	1.992±0.3177	-
Set-Tree-NC	26.063±0.570	17.5	254.761±0.199	15.76	98.971±0.314	35.19	105.170±2.204	35.15
Set-Tree	15.773±1.013	10.61	179.984±7.459	11.12	87.301±8.561	31.04	93.192±0.317	31.14

$AS_{limit} = 5$. For the drugs prescription errors runtime experiment we subsampled 10K random records from the two configurations, *Single Item Replacement (SIR)*, and *Multiple Items Replacement (MIR)*. We trained the models with the following hyperparameters: $N_{est} = 1$, $N_{leafs} = 100$, $N_{[\alpha,\beta]} = 7$ and $AS_{limit} = 5$. We run each experiment 10 times and report the mean runtime and the standard deviation. We conducted all tests on a single CPU and without using multi-threading. The runtime calculation for the non-set-compatible models is including the prepossessing step (the extraction of the statistical moments). Table 2 summarizes the runtime performances.

We consider two sources for computational overhead, sub-optimal implementation and the additional complexity introduced by Set-Tree. For mitigating the first, we compare a tree model trained with Sklearn to DT (vanilla decision tree implemented with our current framework). As can be seen from Table 2 DT's runtime is greater by a factor of 1.15-2.14 in compare to the same model trained with Sklearn. This performance gap will be closed in the next version of our framework. For measuring the computational overhead of Set-Tree, we compare its runtime to DT. For the 2D first quadrant experiment, Set-Tree's runtime is 10.61 times greater than DT and for the 100D configuration it is 11.12 times greater. When the caching mechanism is disabled, the runtime overhead for the two configurations is 17.5 and 15.76. The computational overhead for the drug prescription errors experiment is greater (a factor of ~ 31) because this task is more complex and the number of leafs per tree is greater ($N_{leafs} = 100$ in compare to $N_{leafs} = 10/50$). Utilizing split criteria accelerations, such as using multi-threading or GPU compatible implementation, can further reduce those overheads.

When learning a tree ensemble such as GBeST, another method that can be used to further reduce the runtime is "operation dropout". Meaning, given the family of split criteria parameterized by pairs of (α, β), sample a subset of pairs for each learned tree. This technique allows GBeST to cover large group of optional split criteria while restricting the training runtime. This method is also a form of regularization, similar to subsampling the features in vanilla decision trees. In a future work, we intend to further investigate those methods.

References

Astropy Collaboration and Robitaille. Astropy: A community Python package for astronomy. 558, October 2013.

Chen, T. and Guestrin, C. Xgboost: A scalable tree boosting system. In *Proceedings of the 22nd acm sigkdd international conference on knowledge discovery and data mining*, pp. 785–794, 2016.

Johnson, A. E., Pollard, T. J., Shen, L., Lehman, L.-w. H., Feng, M., Ghassemi, M., Moody, B., Szolovits, P., Celi, L. A., and Mark, R. G. Mimic-iii, a freely accessible critical care database. *Scientific data*, 3:160035, 2016.

Kingma, D. P. and Ba, J. Adam: A method for stochastic optimization. *arXiv preprint arXiv:1412.6980*, 2014.

Paszke, A., Gross, S., Massa, F., Lerer, A., Bradbury, J., Chanan, G., Killeen, T., Lin, Z., Gimelshein, N., Antiga, L., Desmaison, A., Kopf, A., Yang, E., DeVito, Z., Raison, M., Tejani, A., Chilamkurthy, S., Steiner, B., Fang, L., Bai, J., and Chintala, S. Pytorch: An imperative style, high-performance deep learning library. In *Advances in Neural Information Processing Systems 32*, pp. 8024–8035. Curran Associates, Inc., 2019.

Pedregosa, F., Varoquaux, G., Gramfort, A., Michel, V., Thirion, B., Grisel, O., Blondel, M., Prettenhofer, P., Weiss, R., Dubourg, V., Vanderplas, J., Passos, A., Cournapeau, D., Brucher, M., Perrot, M., and Duchesnay, E. Scikit-learn: Machine learning in Python. *Journal of Machine Learning Research*, 12:2825–2830, 2011.

Qu, H. and Gouskos, L. Jet tagging via particle clouds. *Physical Review D*, 101(5), Mar 2020.

Rykoff, E. S., Rozo, E., Busha, M. T., Cunha, C. E., Finoguenov, A., Evrard, A., Hao, J., Koester, B. P., Leauthaud, A., Nord, B., and et al. redmapper. i. algorithm

and sdss dr8 catalog. *The Astrophysical Journal*, 785(2), Apr 2014.

Zaheer, M., Kottur, S., Ravanbakhsh, S., Poczos, B., Salakhutdinov, R. R., and Smola, A. J. Deep sets. In *Advances in neural information processing systems*, pp. 3391–3401, 2017.