Unmasking Trees for Tabular Data

Calvin McCarter
BigHat Biosciences
mccarter.calvin@gmail.com

arXiv GitHub

UnmaskingTrees: autoregressive modeling for generative modeling and imputation

Use XGBoost to predict per-feature conditional distributions.
Train only O(*D*) models given *D* features, since XGBoost handles NaNs.

Training: mask features in random order

- Categorical feature: train XGBoost classifier
- Continuous feature: train BaltoBot (see next section)

Inference: unmask features in random order

- Generation: start with fully-masked sample
- Imputation: start with observed features, then generate the rest

Benefits vs diffusion modeling for tabular data (ForestVP & ForestFlow [1]):

- Autoregression has no train-test mismatch for imputation:
  - Models have been directly trained to impute missing data
  - No need for RePaint-based diffusion inpainting
- Provides density estimation
- No need for different models per each diffusion time-step

BaltoBot: modeling a continuous feature's conditional distribution with recursive partitioning

What's the problem?

Background:

- Diffusion: predict only the conditional mean
- Autoregression: predict (then sample from) the conditional distribution. **We must (1) avoid mode collapse, and (2) sample from bimodal (and multimodal) conditional distributions!**

Previous tabular autoregression work (TabMT [2]) used naive quantization:

- Wide bins: loss of high-resolution information
- Narrow bins: statistically inefficient, catastrophic errors

Probabilistic prediction is an important problem in its own right:

- Quantile regression - same problems as naive quantization
- NGBoost - parametric (fails on multimodal distributions).
- Deep ensembles [3] - slow and expensive
- Diffusion (Treeffuser [4]) - slow and expensive

BaltoBot: BALanced Tree Of BOosted Trees

Hierarchical binary partitioning:

- preserves proximity information among bins
- scales training and inference costs O(height of meta-tree) = O(log (# bins))
- works well on mixed-type and count-type data

Algorithm 2 BaltoBot training

Require: dataset ($\mathbf{X} \in \mathbb{R}^{N\times D}, \mathbf{y} \in \mathbb{R}^N$); BaltoBot meta-tree height H;
if H = 0 **or** unique($\mathbf{y}$) $= C$ for some constant C **then**
  Save `bounds` := $(\min(\mathbf{y}), \max(\mathbf{y}))$.
else
  Obtain split point p from KDI quantization on $\mathbf{y}$.
  Train XGBoost binary classifier on $(\mathbf{X}, \mathbf{1}\{\mathbf{y} \le p\})$.
  Train "left-child" BaltoBot on $\{(\mathbf{X}^{(i)}, \mathbf{y}^{(i)}) \in (\mathbf{X}, \mathbf{y}) | \mathbf{y}^{(i)} \le p\}$, with height $H-1$.
  Train "right-child" BaltoBot on $\{(\mathbf{X}^{(i)}, \mathbf{y}^{(i)}) \in (\mathbf{X}, \mathbf{y}) | \mathbf{y}^{(i)} > p\}$, with height $H-1$.
end if

Algorithm 3 BaltoBot inference

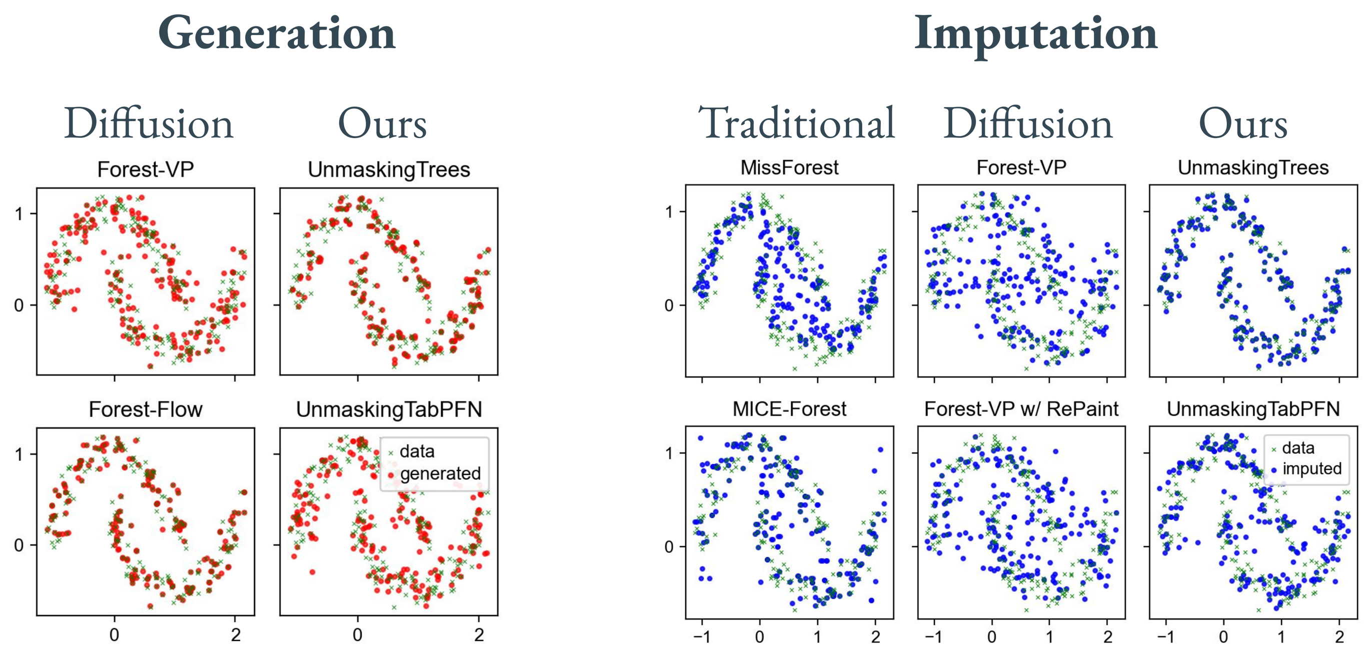
Require: input query $\mathbf{x} \in \mathbb{R}^D$; trained BaltoBot model.
if `bounds` is defined **then**
  Sample uniformly from U(`bounds`).
  Return.
else
  Obtain *prediction* from XGBoost binary classifier.
  if *prediction* = `left-child` **then**
    Run inference on "left-child" BaltoBot with input query $\mathbf{x}$.
  else if *prediction* = `right-child` **then**
    Run inference on "right-child" BaltoBot with input query $\mathbf{x}$.
  end if
end if

UnmaskingTabPFN and BaltoBoTabPFN: in-context learning for tabular generative modeling

Any classification method can be used within the above meta-algorithms!

- Swap XGBoost for TabPFN [5]: pure in-context learning
- NaNTabPFN: a wrapper for TabPFN that handles NaN inputs

UnmaskingTrees is SotA on tabular data with missingness!



Jolicoeur-Martineau et al. benchmark of 27 tabular datasets

Table 1: Tabular data imputation (27 datasets, 3 experiments per dataset, 10 imputations per experiment) with 20% missing. Shown are *averaged rank* over all datasets and experiments (standard-error). Overall best is highlighted; better of Forest-VP versus ours is **boldface blue**.

	MinMAE ↓	AvgMAE ↓	W_{train} ↓	W_{test} ↓	MAD ↓	R^2 ↓	F_1 ↓	P_{bias} ↓	Cov_{rate} ↓
KNN	5.5 (0.5)	6.3 (0.4)	4.9 (0.4)	5.0 (0.4)	8.4 (0)	6.5 (1)	5.7 (1.1)	6.2 (1)	5.4 (0.6)
ICE	6.8 (0.4)	4.7 (0.4)	7.0 (0.5)	7.2 (0.4)	1.6 (0.2)	6.2 (1)	7.0 (0.6)	5.7 (0.9)	5.3 (0.6)
MICE-Forest	3.9 (0.4)	2.5 (0.4)	2.9 (0.2)	3.0 (0.2)	3.6 (0.2)	3.7 (1.4)	3.2 (1)	5.5 (1.2)	4.3 (0.6)
MissForest	2.7 (0.5)	4.0 (0.4)	1.8 (0.3)	2.0 (0.3)	5.5 (0.2)	3.8 (1.4)	2.5 (0.5)	5.5 (1.5)	3.3 (0.5)
Softimpute	6.7 (0.4)	7.6 (0.4)	7.1 (0.5)	7.3 (0.5)	8.4 (0)	6.0 (0.9)	7.8 (0.4)	6.3 (0.9)	6.7 (0.4)
OT	5.9 (0.4)	6.1 (0.3)	6.0 (0.5)	6.0 (0.5)	3.7 (0.3)	6.2 (0.5)	6.8 (0.6)	5.5 (0.8)	4.8 (0.5)
GAIN	4.7 (0.4)	6.5 (0.3)	6.0 (0.3)	6.0 (0.2)	6.9 (0.1)	5.7 (0.8)	5.4 (0.8)	4.7 (1)	5.0 (0.6)
Forest-VP	5.3 (0.4)	4.0 (0.5)	5.8 (0.3)	5.1 (0.4)	**3.2** (0.4)	4.5 (0.9)	4.6 (0.8)	3.3 (0.6)	5.5 (0.7)
UTrees	**3.5** (0.5)	**3.2** (0.5)	**3.5** (0.4)	**3.5** (0.5)	3.8 (0.2)	**2.5** (0.6)	**2.2** (0.6)	**2.3** (0.9)	**4.7** (0.6)

Table 3: Tabular data generation with incomplete data (27 datasets, 3 experiments per dataset, 20% missing values), MissForest is used to impute missing data except in Forest-VP, Forest-Flow, and UnmaskingTrees; *averaged rank* over all datasets and experiments (standard-error). Overall best is highlighted; better of Forest-VP versus Forest-Flow versus ours is **boldface blue**.

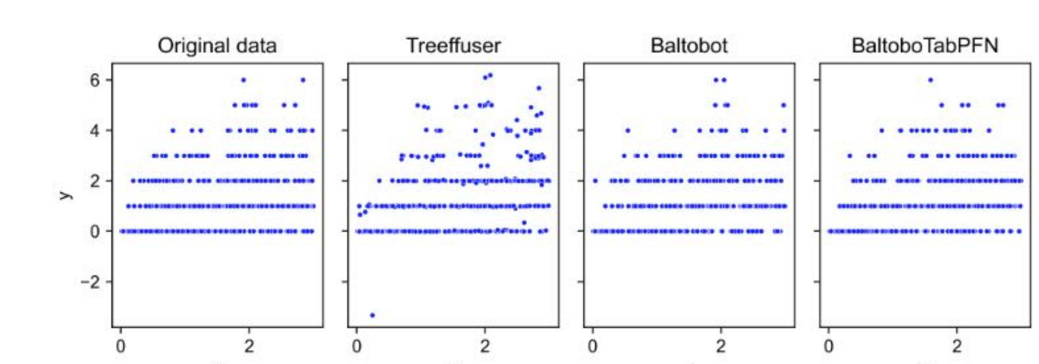
	W_{train} ↓	W_{test} ↓	cov_{train} ↓	cov_{test} ↓	R^2_{fake} ↓	$F1_{fake}$ ↓	$F1_{disc}$ ↓	P_{bias} ↓	cov_{rate} ↓
GaussianCopula	7.0 (0.3)	7.1 (0.2)	7.2 (0.3)	7.1 (0.3)	6.3 (0.4)	6.6 (0.3)	6.7 (0.4)	5.5 (1.0)	7.7 (0.6)
TVAE	5.2 (0.3)	4.9 (0.3)	5.7 (0.3)	5.8 (0.2)	6.0 (1.0)	5.8 (0.5)	5.8 (0.4)	8.0 (0.4)	6.2 (1.0)
CTGAN	8.3 (0.2)	8.4 (0.2)	8.4 (0.2)	8.3 (0.2)	8.3 (0.3)	8.4 (0.2)	6.5 (0.2)	4.8 (1.2)	7.1 (0.7)
CTABGAN	6.7 (0.4)	6.5 (0.4)	7.1 (0.3)	6.8 (0.3)	7.3 (0.6)	7.1 (0.4)	6.6 (0.3)	7.5 (1.0)	6.1 (0.6)
Stasy	5.9 (0.2)	6.1 (0.3)	5.3 (0.2)	5.1 (0.3)	5.8 (0.9)	4.4 (0.4)	5.3 (0.4)	3.7 (0.4)	4.6 (1.1)
TabDDPM	3.0 (0.7)	3.4 (0.7)	2.3 (0.5)	2.9 (0.6)	1.7 (0.3)	3.3 (0.6)	3.9 (0.6)	3.8 (1.2)	2.0 (0.5)
Forest-VP	3.7 (0.2)	3.2 (0.3)	3.9 (0.2)	3.8 (0.3)	3.2 (0.3)	**2.3** (0.3)	4.2 (0.4)	4.2 (0.8)	4.5 (1.1)
Forest-Flow	3.0 (0.3)	**2.6** (0.3)	2.6 (0.3)	2.7 (0.2)	**3.0** (0.7)	3.7 (0.3)	5.0 (0.5)	3.8 (0.9)	**3.2** (0.8)
UTrees	**2.1** (0.2)	2.8 (0.3)	**2.5** (0.2)	**2.5** (0.2)	3.3 (0.8)	3.5 (0.5)	**1.0** (0.0)	**3.7** (0.9)	3.7 (1.0)

Table 4: Tabular data generation with complete data (27 datasets, 3 experiments per dataset); *averaged rank* over all datasets and experiments (standard-error). Overall best is highlighted; better of Forest-VP versus Forest-Flow versus ours is **boldface blue**.

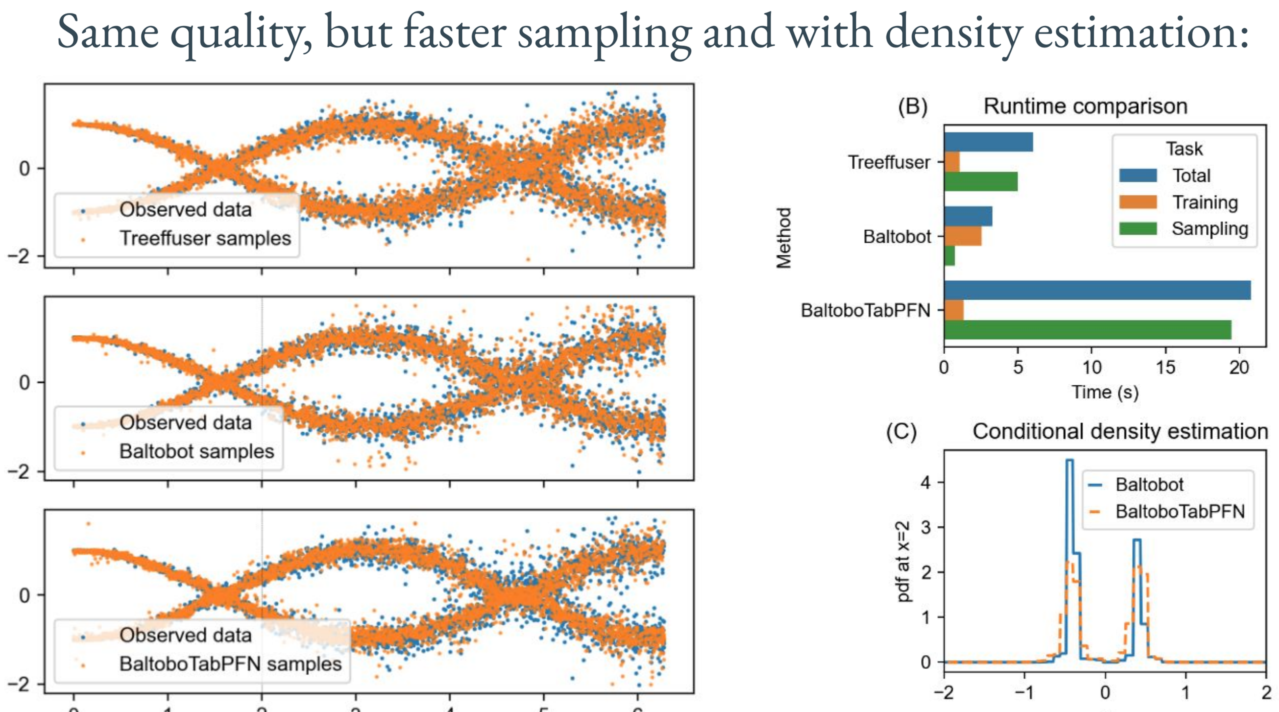
	W_{train} ↓	W_{test} ↓	cov_{train} ↓	cov_{test} ↓	R^2_{fake} ↓	$F1_{fake}$ ↓	$F1_{disc}$ ↓	P_{bias} ↓	Cov_{rate} ↓
GaussianCopula	7.1 (0.3)	7.2 (0.3)	7.3 (0.3)	7.4 (0.3)	6.2 (0.2)	6.4 (0.3)	7.0 (0.4)	6.5 (1.1)	7.5 (0.7)
TVAE	5.3 (0.2)	5.1 (0.2)	5.7 (0.2)	5.7 (0.2)	6.5 (0.7)	6.0 (0.5)	5.5 (0.3)	7.3 (0.6)	6.7 (0.6)
CTGAN	8.4 (0.1)	8.4 (0.2)	8.3 (0.2)	8.1 (0.2)	8.5 (0.2)	8.3 (0.2)	6.7 (0.3)	5.3 (1.1)	7.2 (0.5)
CTAB-GAN+	6.8 (0.3)	6.7 (0.3)	7.2 (0.3)	7.1 (0.3)	6.8 (0.4)	6.9 (0.4)	6.9 (0.3)	7.7 (0.8)	6.7 (0.8)
STaSy	6.1 (0.2)	6.3 (0.2)	5.3 (0.2)	5.4 (0.2)	6.0 (1.2)	5.1 (0.3)	6.1 (0.3)	4.5 (0.8)	4.2 (1.1)
TabDDPM	3.0 (0.7)	3.9 (0.6)	2.8 (0.5)	3.4 (0.5)	1.2 (0.2)	3.8 (0.6)	3.2 (0.4)	3.0 (0.9)	1.4 (0.2)
Forest-VP	3.2 (0.2)	2.8 (0.2)	3.6 (0.3)	3.3 (0.3)	2.8 (0.3)	**2.2** (0.3)	4.3 (0.4)	3.2 (0.9)	3.5 (0.8)
Forest-Flow	**1.9** (0.2)	**1.5** (0.2)	**1.7** (0.2)	**1.8** (0.2)	**2.3** (0.4)	2.4 (0.3)	4.3 (0.4)	**2.8** (0.5)	**2.7** (0.4)
UTrees	3.1 (0.1)	3.1 (0.2)	3.1 (0.2)	2.8 (0.2)	4.7 (0.3)	3.9 (0.3)	**1.0** (0.0)	4.7 (0.7)	5.2 (0.9)

BaltoBot excels at probabilistic prediction!

Better than diffusion on Poisson-distributed data:



Same quality, but faster sampling and with density estimation:



M5 Kaggle dataset for heavy-tailed sales forecasting:

Method	CRPS $\times 10^{-1}$(↓)	RMSE $\times 10^{0}$(↓)	MAE $\times 10^{0}$(↓)
Deep Ensembles	7.05	2.03	0.97
IBUG	8.90	2.12	1.00
NGBoost Poisson	6.86	2.33	0.99
Quantile Regression Forests	7.11	2.88	1.01
Treeffuser	**6.44**	2.09	0.99
BaltoBot	**6.44**	**2.07**	**0.98**
Treeffuser (no tuning)	6.62	2.09	0.99
BaltoBot (no tuning)	6.69	2.19	0.98
BaltoBoTabPFN (no tuning)	6.66	2.06	0.97

References

[1] Jolicoeur-Martineau, Alexia, Kilian Fatras, and Tal Kachman. "Generating and imputing tabular data via diffusion and flow-based gradient-boosted trees." AISTATS 2024.

[2] Gulati, Manbir, and Paul Roysdon. TabMT: Generating tabular data with masked transformers. NeurIPS 2023.

[3] Lakshminarayanan, B., Pritzel, A., & Blundell, C. Simple and scalable predictive uncertainty estimation using deep ensembles. NeurIPS 2017.

[4] Beltran-Velez, Nicolas, et al. "Treeffuser: Probabilistic Predictions via Conditional Diffusions with Gradient-Boosted Trees." NeurIPS 2024.

[5] Hollmann, Noah, et al. "TabPFN: A transformer that solves small tabular classification problems in a second." ICLR 2023.