Accurate predictions on small data with a tabular foundation model

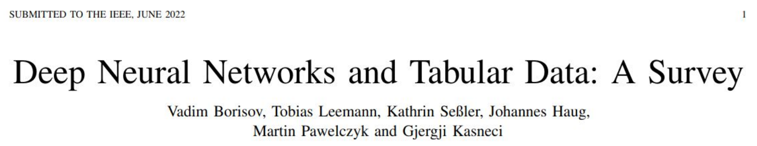
TabPFN version 2.0

1 Introduction

Tabular Data is most Data

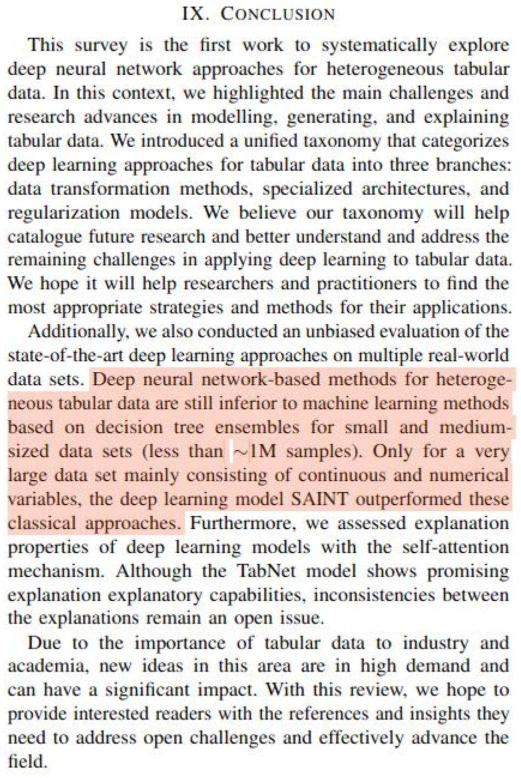
Tabular Data is like
non-elephant zoology

The Deep Learning Revolution has not reached Tabular


SUBMITTED TO THE IEEE, JUNE 2022 1

Deep Neural Networks and Tabular Data: A Survey

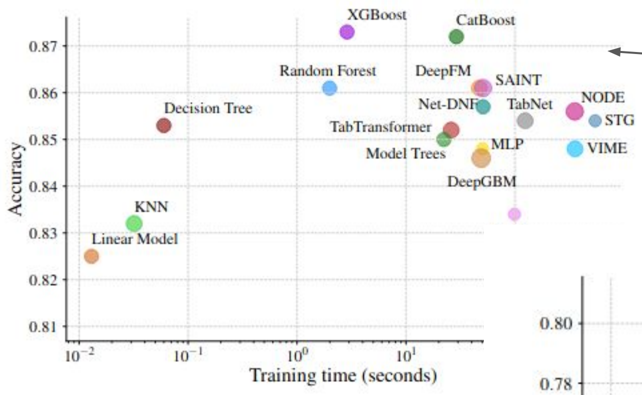
Vadim Borisov, Tobias Leemann, Kathrin Seßler, Johannes Haug, Martin Pawelczyk and Gjergji Kasneci


IX. Conclusion

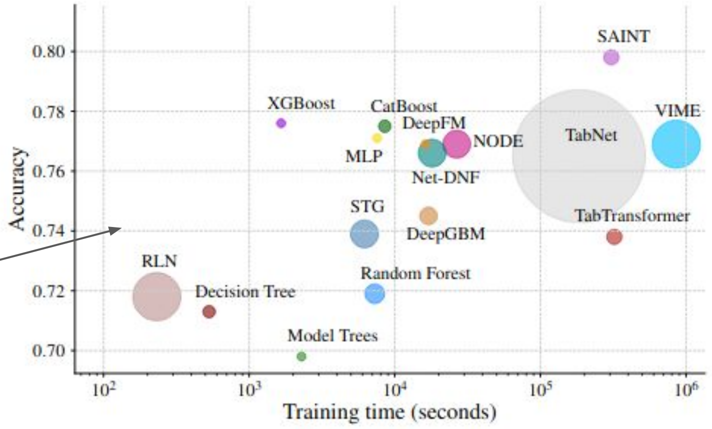
This survey is the first work to systematically explore deep neural network approaches for heterogeneous tabular data. In this context, we highlighted the main challenges and research advances in modelling, generating, and explaining tabular data. We introduced a unified taxonomy that categorizes deep learning approaches for tabular data into three branches: data transformation methods, specialized architectures, and regularization models. We believe our taxonomy will help catalogue future research and better understand and address the remaining challenges in applying deep learning to tabular data. We hope it will help researchers and practitioners to find the most appropriate strategies and methods for their applications.

Additionally, we also conducted an unbiased evaluation of the state-of-the-art deep learning approaches on multiple real-world data sets. Deep neural network-based methods for heterogeneous tabular data are still inferior to machine learning methods based on decision tree ensembles for small and medium-sized data sets (less than ~1M samples). Only for a very large data set mainly consisting of continuous and numerical variables, the deep learning model SAINT outperformed these classical approaches. Furthermore, we assessed explanation properties of deep learning models with the self-attention mechanism. Although the TabNet model shows promising explanation explanatory capabilities, inconsistencies between the explanations remain an open issue.

Due to the importance of tabular data to industry and academia, new ideas in this area are in high demand and can have a significant impact. With this review, we hope to provide interested readers with the references and insights they need to address open challenges and effectively advance the field.



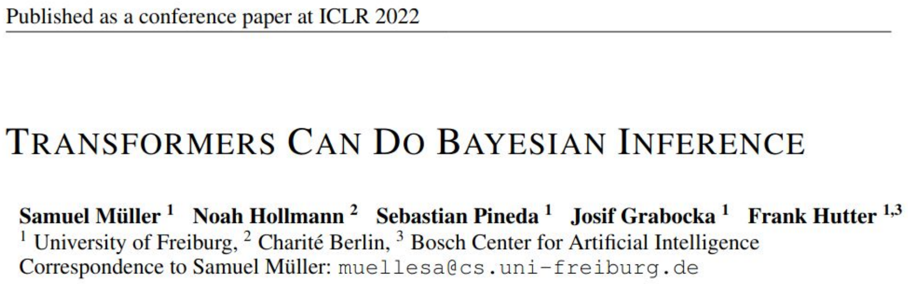
Adult (32k samples)



Higgs (11M samples)

TabPFN

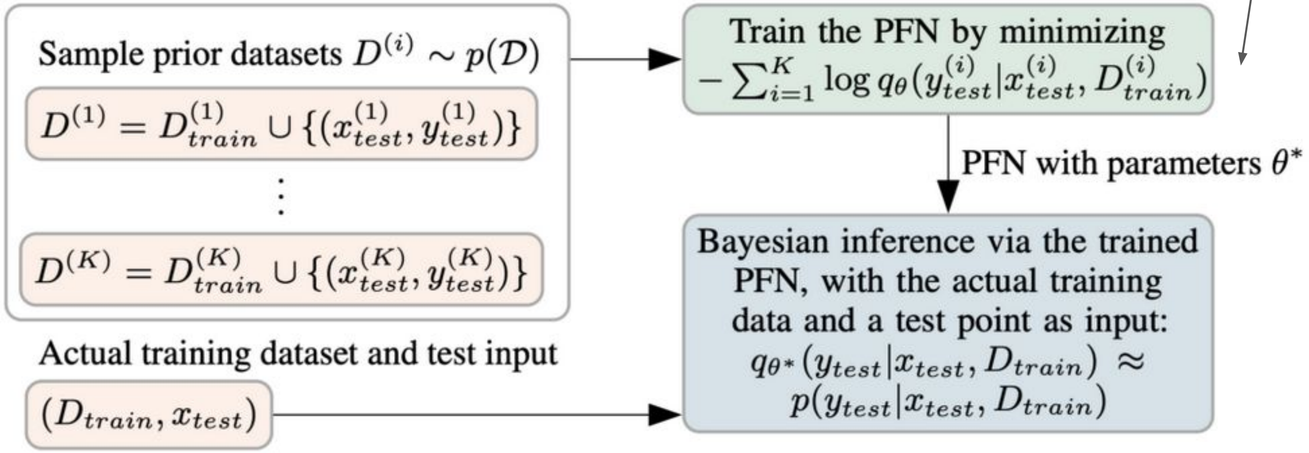
- up to 10k samples and 500 features
- Uses in-context-learning (train and predict in one NN forward pass)
- Transformer trained on synthetic data (100M datasets)
- Prior-data Fitted Network (PFN) approximate Bayesian Inference



Published as a conference paper at ICLR 2022

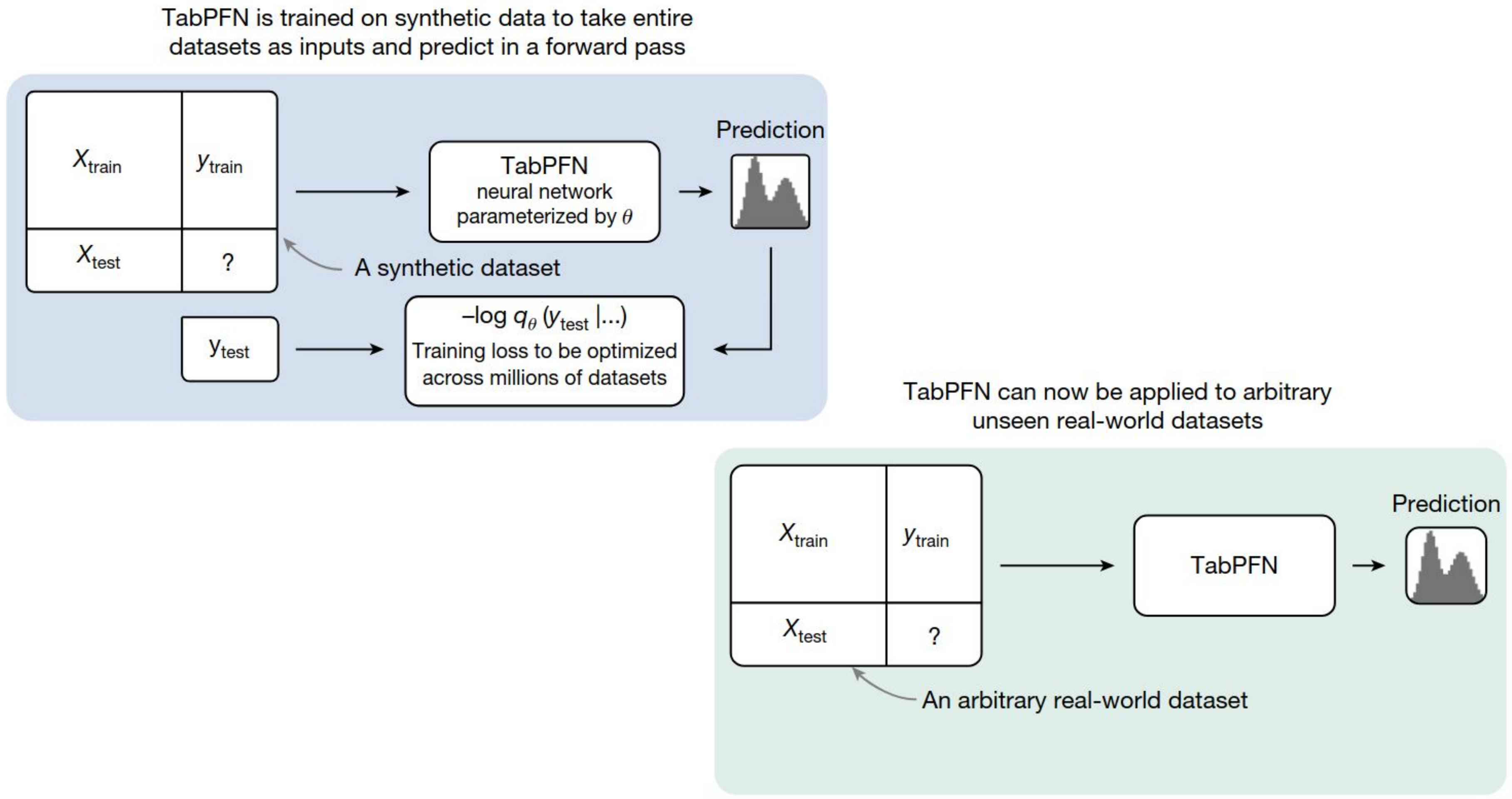
TRANSFORMERS CAN DO BAYESIAN INFERENCE

Samuel Müller [1] **Noah Hollmann** [2] **Sebastian Pineda** [1] **Josif Grabocka** [1] **Frank Hutter** [1,3]
[1] University of Freiburg, [2] Charité Berlin, [3] Bosch Center for Artificial Intelligence
Correspondence to Samuel Müller: `muellesa@cs.uni-freiburg.de`

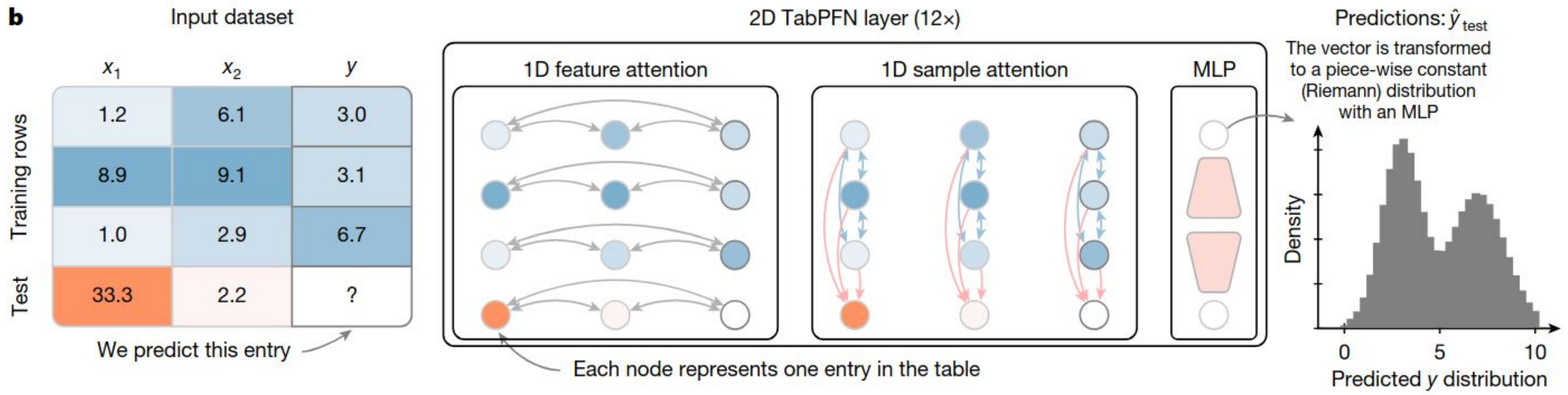

Sample prior datasets $D^{(i)} \sim p(\mathcal{D})$
$D^{(1)} = D^{(1)}_{train} \cup \{(x^{(1)}_{test}, y^{(1)}_{test})\}$
$\vdots$
$D^{(K)} = D^{(K)}_{train} \cup \{(x^{(K)}_{test}, y^{(K)}_{test})\}$
Actual training dataset and test input
(D_{train}, x_{test})
Train the PFN by minimizing $-\sum_{i=1}^{K} \log q_\theta(y^{(i)}_{test} | x^{(i)}_{test}, D^{(i)}_{train})$
PFN with parameters θ^*
Bayesian inference via the trained PFN, with the actual training data and a test point as input: $q_{\theta^*}(y_{test}|x_{test}, D_{train}) \approx p(y_{test}|x_{test}, D_{train})$

2. Methodology

2.1 Architecture


TabPFN is trained on synthetic data to take entire datasets as inputs and predict in a forward pass
X_{train}
y_{train}
X_{test}
?
TabPFN
neural network parameterized by θ
Prediction
A synthetic dataset
y_{test}
$-\log q_\theta (y_{\text{test}} \mid \ldots)$
Training loss to be optimized across millions of datasets
TabPFN can now be applied to arbitrary unseen real-world datasets
X_{train}
y_{train}
X_{test}
?
TabPFN
Prediction
An arbitrary real-world dataset

Architecture



		x_1	x_2	y
Training rows		1.2	6.1	3.0
		8.9	9.1	3.1
		1.0	2.9	6.7
Test		33.3	2.2	?

b
Input dataset
We predict this entry
2D TabPFN layer (12×)
1D feature attention
1D sample attention
MLP
Each node represents one entry in the table
Predictions: $\hat{y}_{\text{test}}$
The vector is transformed to a piece-wise constant (Riemann) distribution with an MLP
Density
0
5
10
Predicted y distribution

Memory and Compute Optimization Tricks

1. Train-State Caching
   - avoid repeating computations during fit-predict
   - speedup of 300x on CPU, 6x on GPU
2. Layer Norm in half precision
3. Flash Attention
4. Activation Checkpointing
5. Sequential State Computation

2.2 Synthetic Data Generation

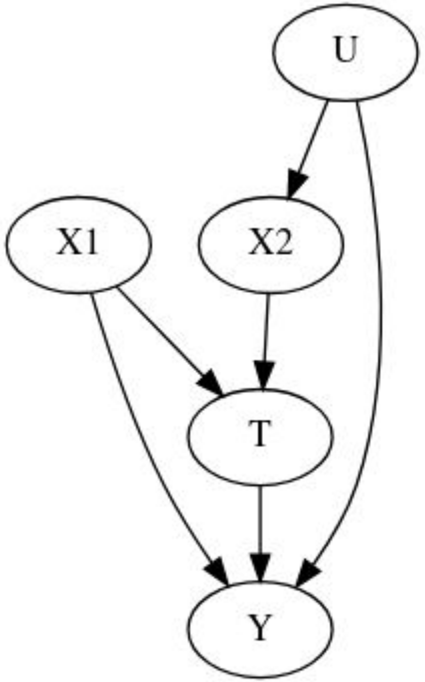
Why use Synthetic Data?

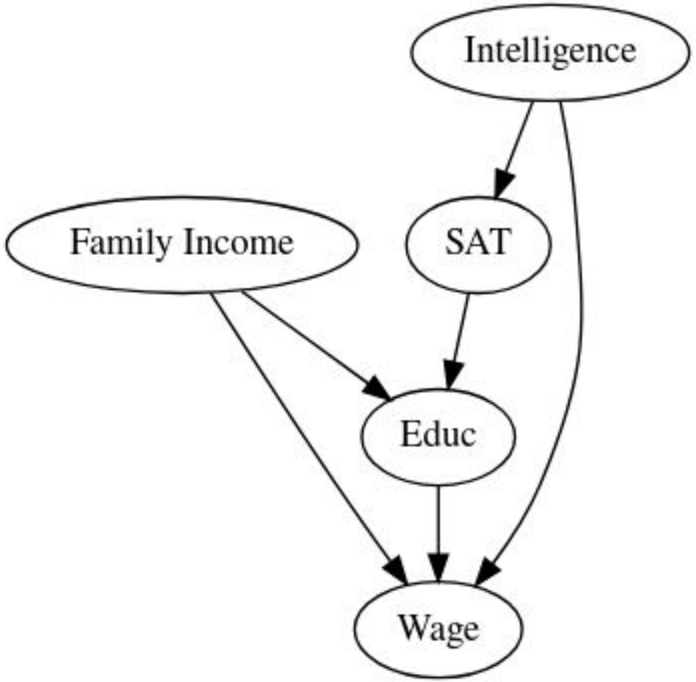
Pretraining with Real World tabular creates Pitfalls

1. Privacy and Copyright
2. Limited date (Data Imbalance)
3. Data Contamination!!!

TabPFN uses data-free Synthetic data
(100 million synthetic datasets!)

Synthetic data based on Structural Causal Models (SCM)





Creating a Dataset: 1/5 Sample underlying parameters

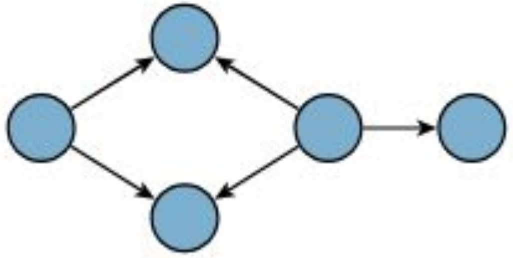
Sample number of data points

Sample number of features

Sample number of nodes

Sample graph complexity

Sample graph



Sampled from analytic distributions (Gamma, LogUni) with hyperparameters chosen to fit with real data

Creating a Dataset: 2/5 Sample a Directed Acyclic Graph

Organization of Growing Random Networks

P. L. Krapivsky and S. Redner
Center for BioDynamics, Center for Polymer Studies, and Department of Physics, Boston University, Boston, MA, 02215

B. Growing Network with Re-direction (GNR)

The GN is built by simultaneous node and link addition and disregards other elemental processes which can occur in the development of large networks. In the context of the web, these include node and link deletion (for out-of-date websites), link re-wiring, the tendency of a new node to connect to nearby nodes, and the copying of links from existing nodes to new nodes. The GNR model incorporates a simple form of link re-wiring into the GN model. At each time step, a new node **n** is added and an earlier node **x** is selected *uniformly* as a possible "target" for attachment. With probability $1 - r$, the link from **n** to **x** is created; in this case, the evolution is the same as in the GN. However, with probability r, the link is *re-directed* to the ancestor node **y** of node **x** (Fig. 2).

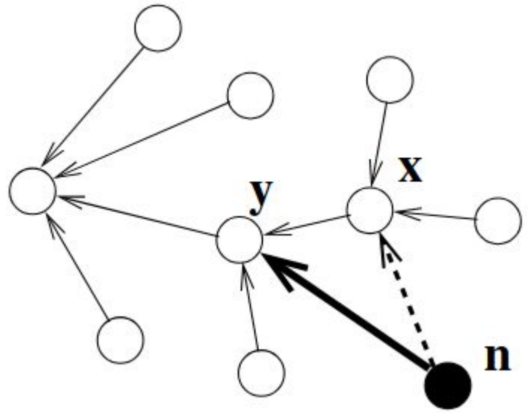
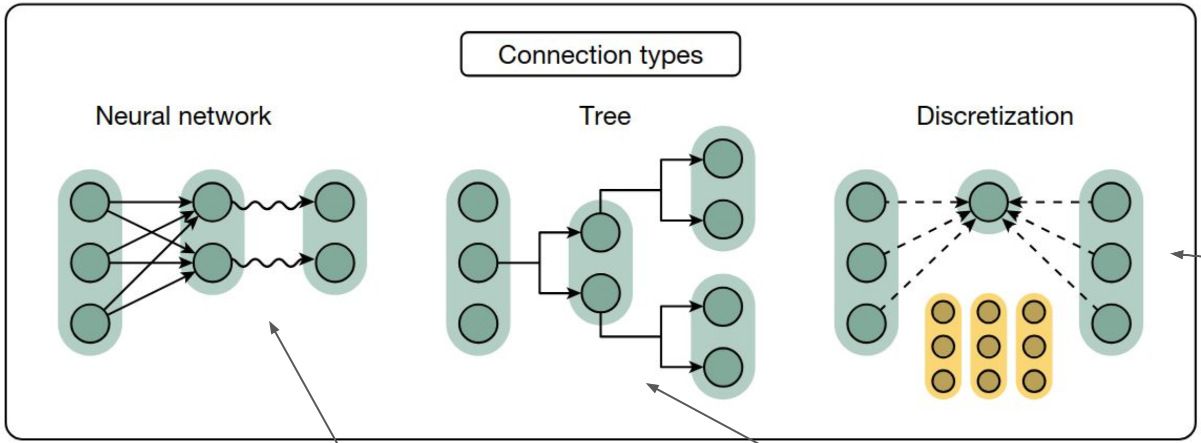


FIG. 2. Illustration of the basic processes in the GNR model. The new node (solid) selects a target node **x**. With probability $1 - r$ a link is established to this target node (dashed arrow), while with probability r the link is established with the ancestor of **x** (thick solid arrow).

Creating a Dataset: 3/5 Sample Computational edge maps

Each **node** is representation **vector**, each **edge** is a **function**

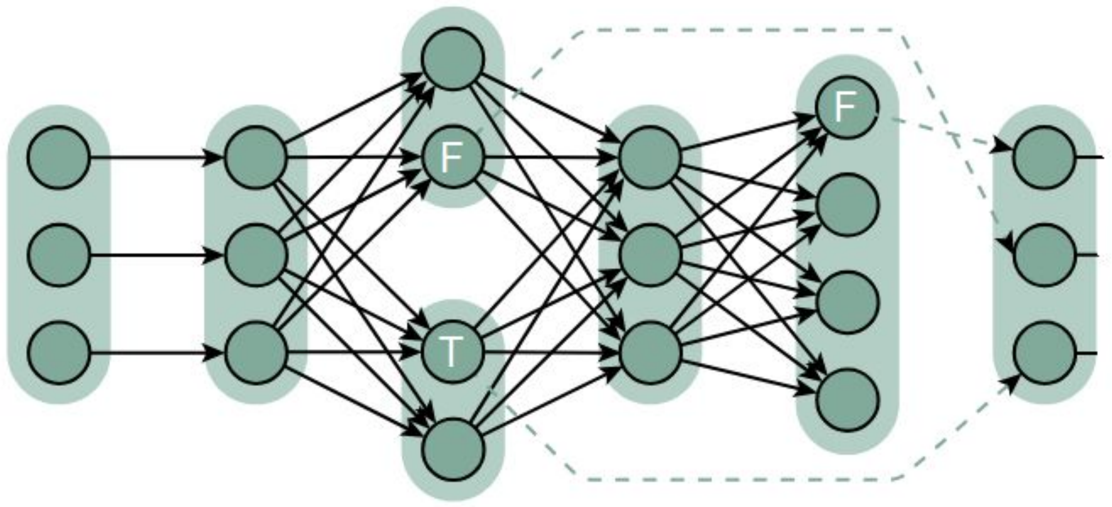


K categories with K vectors,
assign vector to category with nearest neighbor

1-layer NN at Xavier init with element-wise random activation function (ReLu, Log, Tanh, …)

Random decision tree

Creating a Dataset: 4/5 Create Samples



1 For each generated sample, propagate initialization data through the graph

2 Sample random feature (F) and target (T) node positions, and

3 read off data at those positions

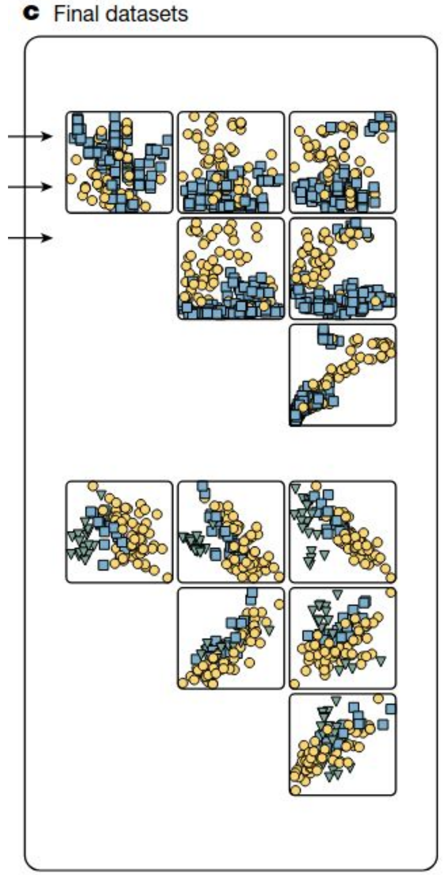
Creating a Dataset: 5/5 Postprocess Samples

Randomly apply some of the following

1. Kumaraswamy feature warping (nonlinear trafo)
2. Quantize some features via bucketing
3. Mask some fraction of the data

Randomly pick target feature

- Regression: use random non postprocessed
- Classification: random categorical feature



3 Experiments

3.1 Qualitative

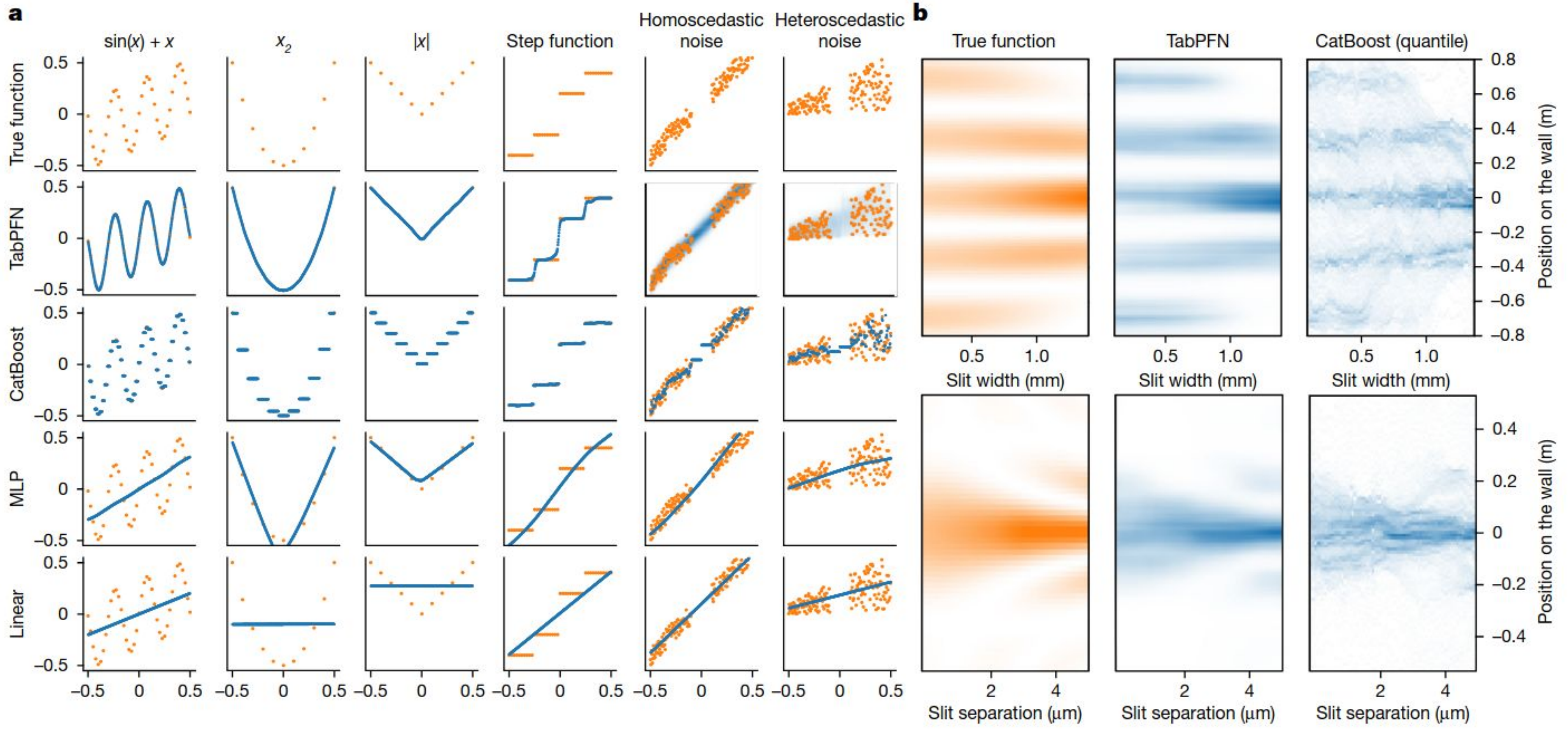


Fig. 3 | The behaviour of TabPFN and a set of baselines on simple functions. In all plots, we use orange for the ground truth and blue for model predictions. **a**, Each column represents a different toy function, each having a single feature (along the *x*-axis) and a target (along the *y*-axis). TabPFN can model a lot of different functions, including noisy functions. **b**, TabPFN can model distributions over outputs out of the box, which is exemplified by predicting the light intensity pattern in a double-slit experiment after observing the positions of 1,000 photons.

3.2 Quantitative

Experimental Setup

- Datasets
  - AutoML Benchmark and OpenML-CTR23 (28 regression, 29 classification)
  - 5 kaggle comps
  - 2 additional benchmarks
- Baselines
  - Tree-based methods (RF, XGB, CatBoost, LightGBM)
  - linear models
  - SVMs (lol)
  - MLP
- Metrics: Accuracy, ROC AUC, R2, RMSE
- 10 runs (random seed and 90/10 split)
- hyperparameter tuning with random search, 5fold CV, time budget 30s-4h

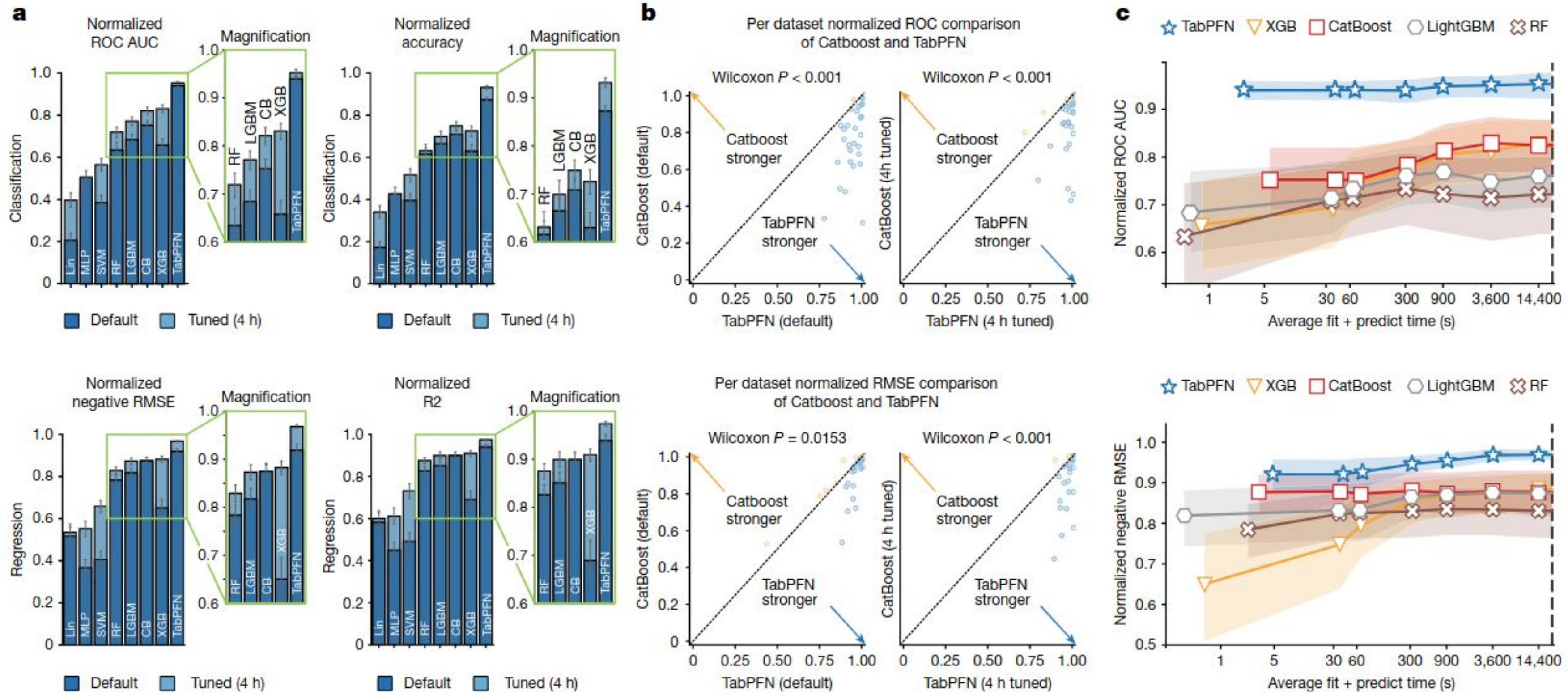


Fig. 4 | Comparison of TabPFN on our test benchmarks, containing datasets with up to 10,000 samples and 500 features. Performance was normalized per dataset before aggregation using all baselines; intervals represent the 95% confidence interval. Wilcoxon *P* refers to the two-sided Wilcoxon signed-rank test *P* value[54]. **a**, Average performance of the default as well as the tuned versions of TabPFN and our baselines. All methods are tuned for ROC AUC or RMSE, respectively, thus decreasing the representativeness of the secondary metrics. LGBM, LightGBM; MLP, multilayer perceptron; SVM, support vector machines; RF, random forest; CB, CatBoost; XGB, XGBoost; Lin, logistic regression for classification and ridge regression for regression tasks. Plots on the right-hand side show a magnified analysis of the strongest baselines considered. **b**, A per-dataset comparison of TabPFN with its strongest baseline, CatBoost. Each dot is the average score on one dataset. **c**, The impact of hyperparameter tuning for the considered methods. The *x*-axis shows the average time required to fit and predict with the algorithm.

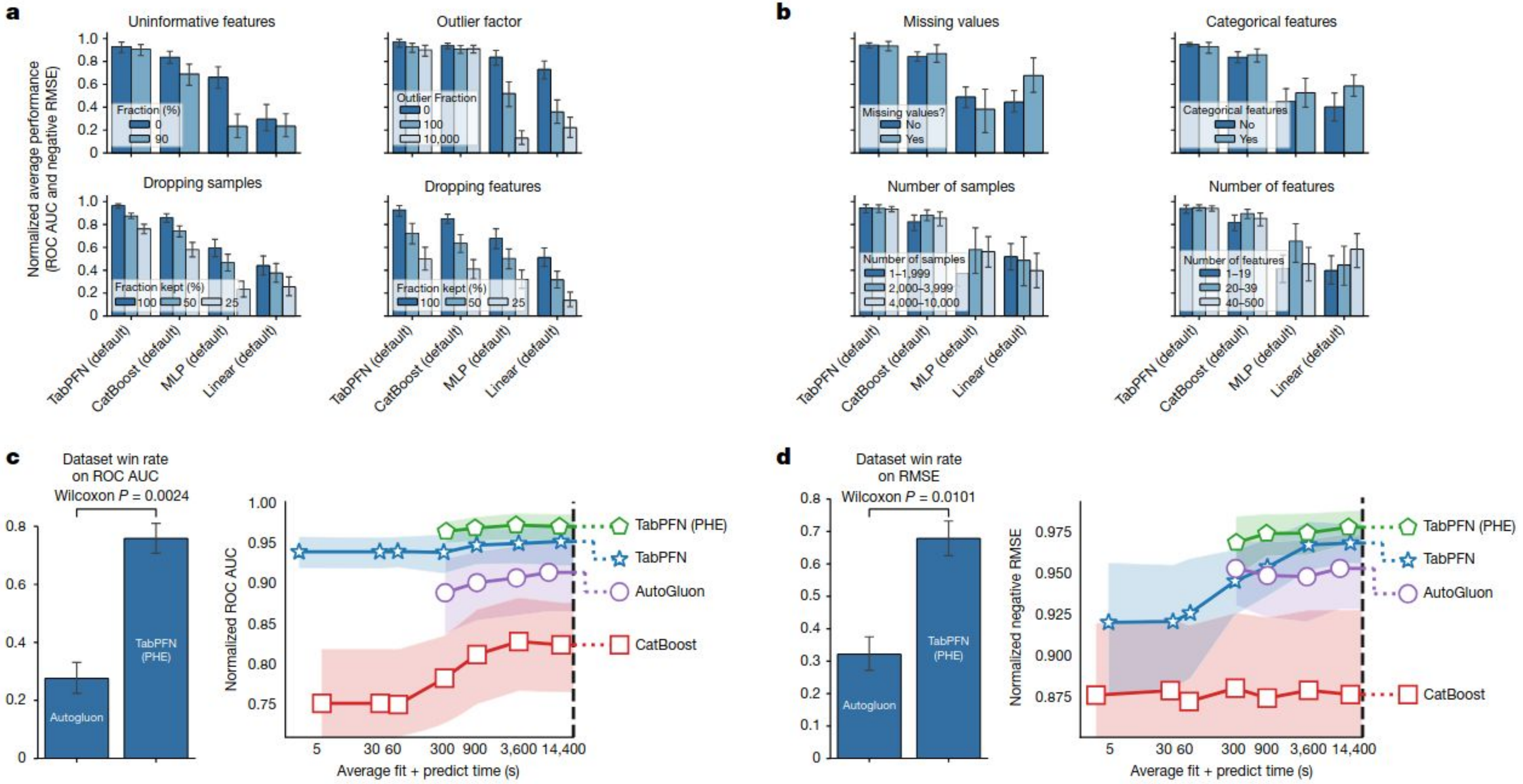


Fig. 5 | Robustness across datasets and performance comparison with tuned ensembles. a, A comparison of modified datasets. We can see that TabPFN is not more vulnerable to the modifications compared with baselines. We also see that TabPFN reproduces the accuracy of CatBoost (default) with only half the training samples provided. Here we normalize scores per dataset (sharing one normalization across all modifications of one experiment) to avoid negative outliers. **b**, We split the test datasets by data characteristics and analyse the performance per subgroup. **c**, Classification performance. Left, the win rate of TabPFN (PHE) against AutoGluon (with one tie excluded); right, the ROC AUC score over time for tuning each method, with the first marker representing the default configuration for the non-ensembling methods. **d**, Regression performance presented as in **c** but using the RMSE metric. Intervals represent the 95% confidence interval and Wilcoxon *P* refers to the two-sided Wilcoxon signed-rank test *P* value[54].

TabPFN as a foundation model

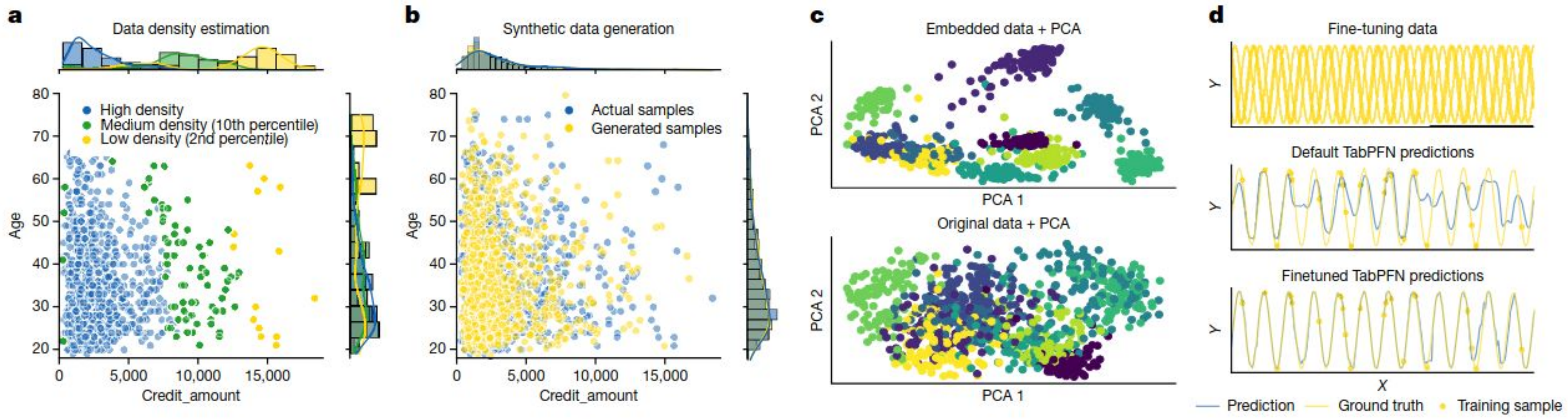


Fig. 6 | Showcase of the application of TabPFN as tabular foundation model. **a**,**b**, On the German Credit Dataset, we perform data density estimation (**a**) and generation of new synthetic samples (**b**). **c**, We show our learned embeddings are useful representations of each sample on the handwritten digits dataset (mfeat-factors) with different classes forming different clusters. **d**, We demonstrate fine-tuning TabPFN for a specific set of tasks. Fine-tuned on a dataset containing various sine curves (top), we see the model makes more accurate predictions on another sine curve dataset.