TabPFN: A Transformer That Solves Small Tabular Classification Problems in a Second

Noah Hollmann *
University of Freiburg &
Berlin Institute of Health
Charité University Medicine Berlin
noah.hollmann@charite.de

Samuel Müller *
University of Freiburg
muellesa@cs.uni-freiburg.de

Katharina Eggensperger
University of Freiburg
eggenspk@cs.uni-freiburg.de

Frank Hutter
University of Freiburg &
Bosch Center for Artificial Intelligence
fh@cs.uni-freiburg.de

Abstract

We present TabPFN, a trained Transformer model that can do tabular supervised classification for small datasets in *less than a second*, needs no hyperparameter tuning and is competitive with state-of-the-art classification methods. TabPFN is entailed in the weights of our network, which accepts training and test samples as a set-valued input and yields predictions for the entire test set in a single forward pass. TabPFN is a Prior-Data Fitted Network (PFN) and is trained offline once, to approximate Bayesian inference on synthetic datasets drawn from our prior. Our prior incorporates ideas from causal learning: It entails a large space of structural causal models with a preference for simple structures. Afterwards, the trained TabPFN approximates Bayesian prediction on any unseen tabular dataset, without any hyperparameter tuning or gradient-based learning. On 30 datasets from the OpenML-CC18 suite, we show that our method outperforms boosted trees and performs on par with complex state-of-the-art AutoML systems with a $70\times$ speedup. This increases to a $3\,200\times$ speedup when a GPU is available. We provide all our code and the trained TabPFN at `https://anonymous.4open.science/r/TabPFN-2AEE`. We also provide an online demo at `https://huggingface.co/spaces/TabPFN/TabPFNPrediction`.

1 Introduction

Many real-world applications in medicine, finance, research, predictive maintenance or sensor data modelling rely on strong Machine Learning (ML) algorithms for tabular data. While neural networks excel on many ML tasks, Gradient-Boosted Decision Trees (GBDT; 14) still dominate the landscape of supervised ML for tabular data [38], largely due to their short training time and robustness.

However, given that tree-based models are not differentiable, they cannot easily be composed and jointly trained with other blocks based on Deep Learning (DL). While many recent works try to address this problem with a native DL approach for tabular classification [2, 39, 18, 20], we go one step further and simply apply a transformer [42] that itself learned to do classification, completely bypassing the question of how to train a DL model on a tabular dataset and the issues that arise from it, such as large computational cost and mitigating overfitting on small datasets [18].

*Equal contribution.

Table Representation Learning workshop at NeurIPS 2022.

We propose a radical change to how tabular classification is usually done. We do *not* fit a new model from scratch to the training portion of a new dataset. Instead, we replace this step by performing a single forward pass on a large transformer that has been pre-trained to solve probabilistic classification tasks which are generated from a prior tabular dataset prior, integrating principles of simplicity and causal reasoning. Our approach reduces the time to solve a tabular classification task to less than a second and simplifies the application to a neural network (NN) forward-pass, a stable, differentiable and highly portable standard procedure.

Our method builds on Prior-Data Fitted Networks (PFNs; 24; see Section 2), which learn the training and prediction algorithm itself. PFNs approximate Bayesian inference given any prior one can sample from and yield the posterior predictive distribution (PPD) directly. Thus, PFNs approximate the probability distributions of classification labels by implicitly weighting all explanations for the data that are entailed in the prior. While inductive biases in NNs and GBDTs depend on them being efficient to implement (e.g. through L_2 regularization, dropout [40] or limited tree-depth), in PFNs, one can simply design a dataset-generating algorithm that encodes the desired prior. This fundamentally changes the way we can design learning algorithms.

[24] demonstrated PPD approximation with PFNs, amongst other experiments, on binary classification for very small, balanced tabular datasets with only up to 30 training examples and very few features. We build on this work to create a state-of-the-art model for tabular multi-class classification tasks that we evaluate on real-world datasets with up to $1\,000$ training samples, 100 features and 10 classes, comprising mixed feature types, missing data and unbalanced targets. We focus on small datasets because (1) small datasets are often encountered in real-world applications, (2) existing DL methods are most limited in this domain and (3) TabPFN is significantly more expensive to evaluate for large datasets, detailed in Appendix A.

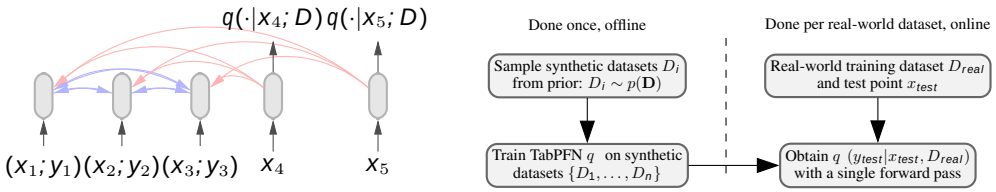
We design a prior (see Section 3) based on Bayesian Neural Networks (BNNs; Neal 26, Gal 15) and Structural Causal Models (SCMs; Pearl 27, Peters et al. 31) to model complex feature dependencies and potential causal mechanisms underlying tabular data. Our prior also takes ideas from Occam's razor: explanations based on simpler SCMs and BNNs have a higher likelihood. In our prior, we define the hyperparameters describing our prior's hypotheses via probability distributions, e.g., a log-scaled Uniform distribution for the average number of nodes in data-generating SCMs. The resulting PPD implicitly models uncertainty over these hyperparameters, weighting hyperparameters in explanations of the data by their likelihood given the data and their prior probability. Thus, obtaining the PPD corresponds to an infinitely large ensemble of hyperparameters, i.e., instantiations of SCMs and BNNs, in a single forward-pass, and requires no cross-validation or model selection.

Finally, in Section 4, we analyze the performance and behavior of TabPFN on different tasks and compare it to previous approaches for tabular classification on small datasets. Quantitatively, they yield much better performance than any individual "base-level" classification algorithm, such as gradient-boosting via XGBoost [6], LightGBM [19] and CatBoost [32], and in a single second yield performance competitive to what the best available AutoML frameworks [8, 11] achieves in 5–30 minutes.

2 Background on Prior-Data Fitted Networks

We first provide a brief recap on how PFNs work and refer to [24] for more details.

The Posterior Predictive Distribution for Supervised Learning In the Bayesian framework for supervised learning, the prior defines a space of hypotheses Φ on the relationship of a set of inputs x to the output labels y. Each hypothesis ϕ can be seen as a mechanism generating a data distribution from which we can draw samples forming a dataset. For example, given a prior based on Structural Causal Models, Φ is the space of Structural Causal Models, a hypothesis ϕ one specific SCM and a dataset comprises samples generated through the SCM. In practice, a dataset comprises training data with labels observed and test data where labels are missing or held out to assess predictive performance. The PPD for a test sample x_{test} is conditioned on the set of training samples $D_{train} := \{(x_1, y_1), \ldots, (x_n, y_n)\}$ and specifies the distribution of its label $p(\cdot|x_{test}, D_{train})$. It is obtained by integration over the space of hypotheses Φ, where the weight of a hypothesis is



(a) Architecture and attention mechanism

(b) Prior-fitting and inference

Figure 1: Left (a): Training samples $\{(x_1, y_1), \ldots, (x_3, y_3)\}$ are transformed to 3 tokens, which attend to each other; test samples x_4 and x_5 attend only to the training samples. Right (b): The PFN learns to approximate the PPD of a given prior in the offline stage to yield predictions on a new dataset in a single forward pass in the online stage. Plots based on [24].

determined by its prior probability and the likelihood of the observations given this hypothesis:

$$p(y|x, D) \propto \int p(y|x, D, \phi)p(D|\phi)p(\phi)d\phi. \tag{1}$$

While obtaining the PPD is generally intractable, PFNs approximate the PPD, given any prior one can sample from. A PFN that minimizes its training objective to the global minimum, provably approximates exactly the PPD, as shown in [24].

Architecture PFNs rely on a Transformer [42] that encodes each feature vector and label as a token, allowing token representations to attend to each other, as depicted in Figure 1a. They accept a variable length training set D_{train} of feature and label vectors as well as a variable length query set of feature vectors $x_{test} = \{x_{(test;1)}, \ldots, x_{(test;m)}\}$ and return estimates of the PPD for each query.

Synthetic Prior-fitting During prior-fitting, the PFN is trained to approximate Bayesian inference given a prior which is specified by a prior sampling scheme $p(D) = \mathbb{E}_{\sim p(\ )}[p(D|\phi)]$, which first samples hypotheses (generating mechanisms) with $\phi \sim p(\phi)$ and then synthetic datasets with $D \sim p(D|\phi)$. We repeatedly sample such synthetic datasets $D := (x_i, y_i)_{i \in \{0;\ldots;n\}}$ and optimize the PFN's parameters θ to make predictions for $D_{test} \subset D$ with held-out labels y_{test} and conditioned on the rest of the dataset $D_{train} = D \setminus D_{test}$. This minimizes the cross entropy loss over sampled datasets w.r.t to the approximated PPD q :

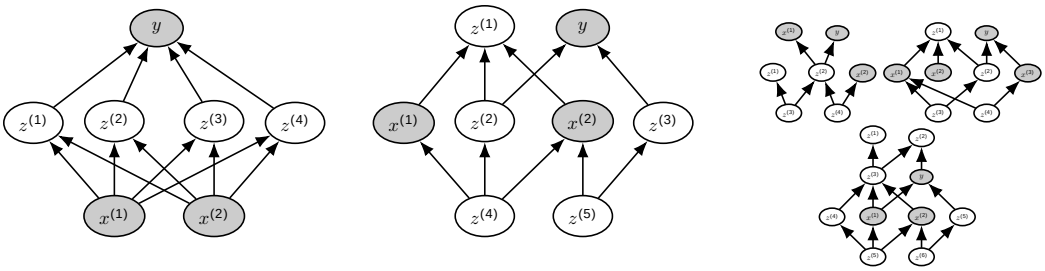
$$\mathcal{L}_{PFN} = -\mathop{\mathbb{E}}_{\{D_{test} \cup D_{train}\} \sim p(D)} [q\ (y_{test}|x_{test}, D_{train})] \tag{2}$$

As demonstrated in [24], minimizing this loss approximates exact Bayesian posterior prediction. We visualize this in Figure 1b and detail the full training setup in Algorithm 1 in the appendix. Crucially, this *synthetic prior-fitting* phase is performed only once for a given prior $p(D)$, learning to do *real-world inference* on any new dataset.

Real-World Inference During inference, the trained model is applied to arbitrary real-world datasets. For a novel dataset with training samples D_{train} and test features x_{test}, feeding it as an input to the model trained above yields the PPD $q\ (y|x_{test}, D_{train})$ in a single forward-pass. The PPD class probabilities are then used as predictions for our real-world task. Thus, PFNs perform training and prediction in one step (similar to prediction with Gaussian Processes) and do not use gradient-based learning for predicting novel datasets.

3 A Prior for Tabular Data

The performance of our method crucially depends on the specification of a suitable prior, as the PFN approximates the PPD for this prior. Section 3.1 outlines a fundamental technique for our prior: we use distributions instead of point-estimates for almost all hyperparameters of our prior. Section 3.2 motivates simplicity in our prior, while Sections 3.3 and 3.4 describe, how we use Structural Causal Models (SCMs) and Bayesian Neural Networks (BNNs) as fundamental mechanisms generating diverse data in our prior. Since our SCM and BNN priors only yield regression tasks, we show how to convert them to classification tasks in Section 3.5. We describe additional refinements to our prior to reflect tabular data better in Appendix D.2.



(a) A BNN (b) An SCM (c) Sample SCMs from the prior

Figure 2: Visualization of graphs generating data in our prior. The inputs x (grey) are mapped to the output y (grey) with unobserved nodes z (white). Independent noise variables ϵ (not visualized) are added to all nodes.

3.1 Fundamentally Probabilistic Models

Fitting a model typically requires finding suitable hyperparameters, e.g. the embedding size, number of layers and activation function for NNs. Commonly, resource-intensive searches need to be employed to find suitable hyperparameters [23, 50, 9]. The result of these searches, though, is only a point estimate of the hyperparameter choice. Ensembling over multiple architectures and hyperparameter settings can yield a rough approximation to a distribution over these hyperparameters and has been shown to improve performance [48, 44]. This, however, scales linearly in cost with the number of choices considered.

PFNs allow us to be fully Bayesian about the hyperparameters of our models in a single forward pass. By defining a probability over the space of hyperparameters in the prior, such as BNN architectures, the PPD approximated by TabPFN jointly integrates over this space of hyperparameters and respective model weights. We extend this approach to a mixture not only over hyperparameters but distinct priors: we mix a BNN and a SCM prior, each of which again entails a mixture of architectures and hyperparameters.

3.2 Simplicity

We base our priors on a notion of simplicity, such as stated by Occam's Razor or the Speed Prior [35]. When considering competing hypotheses, the simpler one, e.g. the one requiring fewer parameters, is to be preferred. Any notion of simplicity, however, depends on choosing a particular criterion that defines simplicity. In the following, we introduce priors based on SCMs and BNNs where we , in which simplicity materializes as graphs with few nodes and parameters.

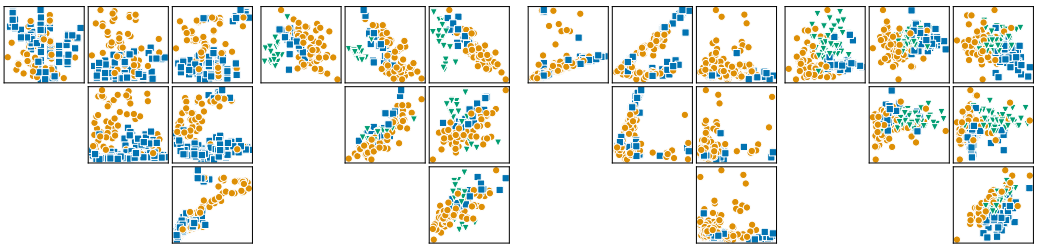
3.3 Structural Causal Prior

It has been demonstrated that causal knowledge can facilitate various ML tasks, including semi-supervised learning, transfer learning and out-of-distribution generalization [37, 17, 34]. Tabular data often exhibits causal relationships between columns, and causal mechanisms have been shown to be a strong prior in human reasoning [43, 45]. Thus, we base our TabPFN prior on SCMs that model causal relationships [27, 31]. An SCM consists of a collection $Z := (\{z_1, \ldots, z_k\})$ of structural assignments (called mechanisms)

$$z_i = f_i(z_{\mathrm{PA}_\mathcal{G}(i)}, \epsilon_i), \tag{3}$$

where $\mathrm{PA}_\mathcal{G}(i)$ is the set of parents of z_i (its direct causes) in an underlying DAG $\mathcal{G}$ (the causal graph), f_i is a (potentially non-linear) deterministic function and ϵ_i a noise variable. Causal relationships in $\mathcal{G}$ are represented by directed edges pointing from causes to effects and each mechanism z_i is assigned to a node in $\mathcal{G}$, as visualized in Figure 2.

Relation to Prior Work There are two major ways of incorporating causal reasoning into predictions, causal inference and causal discovery. Here, we outline both and explain how our approach differs. Causal inference seeks to identify causal relations between the components of a system by the use of interventions and observational data [28]. Causal discovery seeks to reveal causal information



(a) Synthetic datasets (b) Actual datasets

Figure 3: Each plot shows two features, where each point represents a sample with colors indicating the class label. (a) Two synthetic datasets generated by our causal tabular prior. Numeric SCM outputs are mapped to classes as described in Section 3.5. (b) Two datasets from our validation datasets: Parkinsons (Left) and Wine (Right).

by analyzing purely observational data [49]. These causal representations are then used to make predictions on novel samples or to provide explainability. Most existing work focuses on determining a single causal graph to use for downstream prediction. This can be problematic since most kinds of SCMs are non-identifiable without interventional data, and the number of compatible DAGs explodes due to the combinatorial nature of the space of DAGs. While [1] extend this representation to the space of discrete DAGs, this work is limited to a few subfamilies of SCMs while the computational complexity is high. Our TabPFN effectively reasons over the full space of hypotheses in our prior, i.e. a broad family of SCMs, and their respective weights in a single forward pass. We achieve this by skipping any explicit graph representation in our inference step and approximate the PPD directly. Thus, we do not perform causal inference or discovery but solve the downstream prediction task directly. The use of this implicit assumption of causality can be explained by Pearl's "ladder of causation", an abstraction of inference categories, where each higher rung represents a more involved notion of inference [29]. At the lowest rung lies association, which includes most of ML. At the second rung, we find predicting the effect of interventions, i.e. what happens when we influence features directly. Our work can be considered as "rung 1.5", similar to [21, 22]: we do not seek to predict the effect of interventions, but make more informed predictions on observational data using the assumption that causal mechanisms generate the data.

Method To create a PFN prior based on SCMs, we have to define a sampling procedure of supervised learning tasks (i.e. datasets). Here, each dataset is based on one randomly sampled SCM (inducing DAG structure, activation functions, and a deterministic functions f_i). Given an SCM, we sample a set z_X of nodes in $\mathcal{G}$, one for each feature in our synthetic dataset, and a node z_y from the causal graph $\mathcal{G}$. These nodes are observed nodes: values of z_X will be included in the set of features, while values from z_y will act as targets. For each such SCM and list of nodes z_X and z_y, n samples are generated by sampling all noise variables in the SCM n times, propagating these through the graph and retrieving the values z_X and z_y for all n instances. Figure 2 depicts an SCM with observed feature- and target-nodes in grey. In Figure 3b we showcase scatter plots of samples generated by two distinct SCMs. The resulting features and targets are correlated through the generating DAG structure. This leads to features conditionally dependent through forward and backward causation, i.e. targets might be a cause or an effect of features.

In this work, we instantiate a subfamily of DAGs and deterministic functions f_i that can be efficiently sampled from, as described in Appendix D.1. And since efficiently sampling SCMs is the only requirement we have, the instantiated subfamily is very general, including multiple activation functions and potentially non-Gaussian noise.

When we go back to Equation 1, which describes the PPD, we can understand it in the context of the SCM prior as follows: The PPD integrates predictions over the space of SCM graph hypotheses, where the weight of each hypothesis is determined by the likelihood of observing data given this hypothesis and the likelihood of this hypothesis in our prior.

3.4 BNN Prior

We also consider a BNN prior as introduced by Müller et al. [24] and mix it with the SCM prior described above by randomly sampling datasets during PFN training from either one or the other prior (weighted by hyperparameter *Sample SCM vs BNN*, see Appendix F.4). To sample a dataset from the BNN prior, we first sample a NN architecture and its weights. Then, for each data point in the to-be-generated dataset, we sample an input, x feed it through the BNN with sampled noise variables and use the output y as a target (see Figure 2).This is a more general setup than in standard BNNs, as the posterior also considers a distribution over architectures, not only the standard distribution over a fixed architecture's weights. The details of this approach can be found in prior work by Müller et al. [24].

3.5 Multi-class Prediction

So far, the described priors return scalar labels. In order to generate synthetic classification labels, we need to transform our scalar labels $\hat{y}$ to discrete class labels y. We do so by simply splitting the values of $\hat{y}$ into intervals that map to class labels:

1. We sample the number of classes $N_c \sim p(N_c)$, where $p(N_c)$ is a distribution over integers.
2. We sample $N_c - 1$ bounds $B_c \sim p(B_c|N_c, \hat{y})$, where $p(B_c|N_c, \hat{y})$ samples a random value from $\hat{y}$.
3. We map each scalar label $\hat{y}_i$ to the index of the unqiue interval that contains it: $y_i \leftarrow \sum_j [B_j < \hat{y}_i]$, where $[\cdot]$ is the indicator function.

For example, with $N_c = 3$ classes the bounds $B_c = \{-0.1, 0.5\}$ would define three intervals $\{(-\infty, -0.1], (-0.1, 0.5], (0.5, \infty)\}$. Any $\hat{y}_i$ would be mapped to the label 0 if it is smaller than -0.1, to 1 if lies in $(-0.1, 0.5]$ and to 2 otherwise. Finally, we shuffle the labels of classes, i.e. we remove the ordering of class labels w.r.t. the ranges.

4 Experiments

4.1 Evaluation on Toy Problems

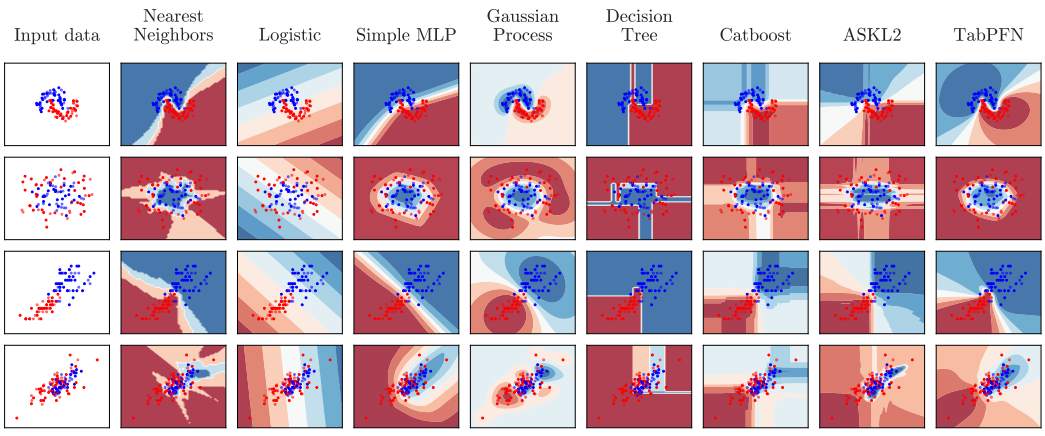


Figure 4: Decision boundaries on toy datasets generated with *scikit-learn* [30].

We qualitatively compare standard classifiers without hyperparameter tuning to our TabPFN in Figure 4. The top row shows the *moons* dataset with noise. The TabPFN accurately models the decision boundary between samples; also, similar to Gaussian processes, its uncertainties are large for points far from observed samples. The second row shows the *circles* dataset with noise: TabPFN accurately models the circle's shape with high confidence anywhere outside the region where samples are mixed. The third row shows two classes and features from the iris dataset, while the third rows shows two classes and features from the wine dataset (both included in *scikit-learn*).

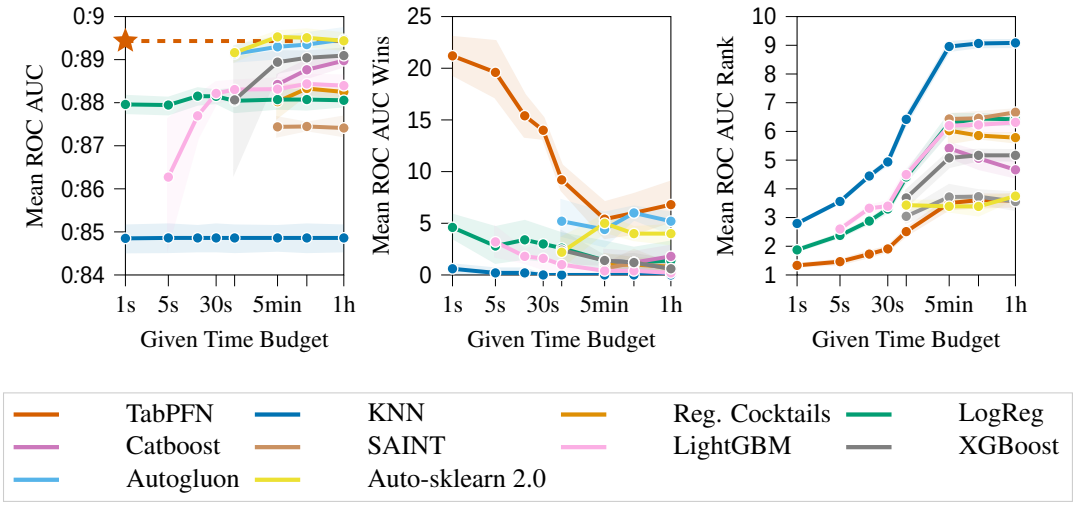


Figure 5: ROC AUC performance over time. We report mean ROC, mean wins and rank along with the 95% confidence interval across 5 repetitions for different time budgets (Unlabelled ticks: 1min, 15min).

4.2 Evaluation on Tabular AutoML Tasks

Now, we turn to an empirical analysis of our method for real-world classification tasks. We compare our method against state-of-the-art ML and AutoML methods for tabular classification.

Datasets As test datasets, we used all datasets from the curated open-source OpenML-CC18 benchmark suite [4] that contain less than 2 000 samples, 100 features and 10 classes. The resulting set comprises 30 datasets. Additionally, we considered a disjoint set of 150 validation datasets from `OpenML.org` [41, 13]. We used the validation datasets to guide the development of our method and the choice of hyperparameter distributions. We did so by qualitatively comparing the data genereated by our prior to real data (See Figure 3b) and by quantitatively comparing feature correlations, feature variance and class distributions. We give a full description of all datasets and how we collected them in Appendix G.3.

Baselines We compare against five standard ML methods and three state-of-the-art AutoML systems for tabular data. As ML models we considered two simple and fast baselines, *K-nearest-neighbors (KNNs)* and *Logistic Regression (LogReg)*. Additionally, we considered *Gaussian Processes (GPs)* [33] and three popular tree-based boosting methods, *XGBoost* [7], *LightGBM* [**?**] and *CatBoost* [32]. For each ML model, we used 5-fold cross-validation to evaluate randomly drawn configurations until a given budget was exhausted or 1 000 configurations were evaluated (for the search spaces, see Appendix G.2). We then refit the configuration with the lowest ROC AUC OVO error on the whole training set. Where necessary, we imputed missing values with the mean, one-hot encoded categorical inputs and normalized features. As more complex but also potentially powerful baselines, we chose three AutoML systems: *AutoGluon* [8], which combines ML models including neural networks and tree-based models into a stacked ensemble, *Auto-sklearn 2.0* [10, 12] which uses Bayesian Optimization and combines the evaluated models into a weighted ensemble. [2] We note that previous works have found that DL baselines do not outperform or match the performance of GBDT or AutoML methods for small to medium sized tabular data [5, 16, 38]. Furthermore, DL methods such as TabNet, SAINT, Regularization Cocktails, Non-parametric transformers [2, 39, 18, 20] are evaluated on much larger datasets, require considerably longer training times and often use custom parameter tuning and preprocessing. We still evaluate two prominent DL methods: *Regularization Cocktails* and *SAINT* [39, 18].

[2] Auto-sklearn 2.0 optimizes ROC AUC for binary classification and cross-entropy for multi-class classification (as multi-class ROC AUC is not implemented).

TabPFN We follow the original PFN architecture [24] which uses a Transformer encoder [42] with a specific attention mask and without positional encodings. We make slight modifications to these attention masks, yielding faster inference at equal performance. Additionally, we allow our model to work with datasets that have different numbers of features by zero-padding and scaling the features linearly. See Appendix F.2 for more information.

In the prior-fitting phase, we train TabPFN once on the prior described in Section 3 for 20 hours on 8 GPUs, yielding a single network that is used for all our evaluations. While this step is expensive, it is done offline, in advance and only once for the TabPFN, as part of the algorithm development. During inference, we apply z-normalization and ensemble predictions, with randomly rotated feature dimensions and permuted labels. Further details and the used hyperparameters can be found in Appendix F.

Evaluation Protocol For each dataset and method, we evaluated 5 repetitions, each with a different random seed and a different train- and test split (all methods used the same split given a seed). To aggregate results across datasets, we report the ROC AUC (one-vs-one (OVO) for multi-class classification) average, ranks and wins including the 95% confidence interval and compare to the performance of the baselines with a budget of $[30, 60, 300, 900, 3600]$ seconds[3]. Our TabPFN is run using 32 data permutations for ensembling as described above; we also evaluate TabPFN without permutations, which we label as TabPFN n.e. in Table 1.

Results We present the achieved AUC ROC as a function of budget in Figure 5, demonstrating that TabPFN is dramatically faster than all the other methods: it makes predictions within a single second on one GPU that tie with the performance of the best competitors (the AutoML systems) after training one hour, and that dominate the performance achieved by competitors in 5 minutes. Unsurprisingly, the simple baselines (*LogReg*, *KNN*) already yield results with a small budget but perform worst overall for larger budgets. Boosted trees (*XGBoost*, *CatBoost*) perform better but are clearly outperformed by the state-of-the-art AutoML systems, Auto-sklearn 2.0 and Autogluon. The AutoML systems are the best-performing baselines at higher budgets.

The TabPFN without ensembling requires 0.0187s on a GPU or 0.87s on a CPU to predict one dataset. It performs comparable to the strongest baselines at one minute, yielding a $70\times$ speedup on CPU and a $3\,200\times$ speedup using a GPU. Given more budget (≥ 5 minutes), the strongest baseline achieves competitive performance with the original TabPFN (including ensembling), which uses a time budget of 0.42 seconds on a GPU or 16.5s on a CPU. If we compare TabPFN against the 5 minutes required for the baseline, it yields an $18\times$ speedup on a CPU and a $710\times$ speedup using a GPU.

These times assume that our trained model is in memory (and possibly moved to the GPU) already, which otherwise required 0.2s for us. We report averaged results in Table 1 and provide further detailed results, including results per dataset in Appendix C.1.

5 Conclusions & Future Work

We have shown how a single Transformer, the TabPFN, can be trained to do the work of a full AutoML framework for tabular data and can yield predictions in 0.4 seconds that are competitive with the performance that the best available AutoML frameworks achieve even after one hour. This result could disrupt the field of data science since it allows for real-time predictions. It also slashes the computational expense of AutoML, allowing for affordable, green, AutoML.

TabPFN still has important limitations: the underlying transformer architecture only scales to small datasets; our evaluations were done on classification datasets with under 1000 training samples, 100 features and 10 classes only, detailed in Appendix A, which motivates work on (1) scaling up to large datasets. Also, our work motivates a multitude of exciting follow-ups regarding the (2) integration of TabPFN into existing AutoML frameworks; (3) ensembling to continue making improvements given more time; (4) studies of out-of-distribution robustness; (5) dataset-dependent choices of the prior; (6) generalizations to non-tabular data and (7) regression tasks. The almost instant state-of-the-art predictions of TabPFN are also likely to give rise to (8) novel exploratory data analysis methods, (9) novel feature engineering methods and (10) novel active learning methods. Our advances in causal

[3]When comparing methods to each other for a given time budget in Figure 5, we drop methods that take more than 200% of the requested time budget; some methods also do not use their full budget.

reasoning warrant follow-ups on (11) approximating the effects of interventions and counterfactuals considering a distribution of SCMs.

Acknowledgments and Disclosure of Funding

Robert Bosch GmbH is acknowledged for financial support. We acknowledge funding by European Research Council (ERC) Consolidator Grant "Deep Learning 2.0" (grant no. 101045765). Funded by the European Union. Views and opinions expressed are however those of the author(s) only and do not necessarily reflect those of the European Union or the ERC. Neither the European Union nor the ERC can be held responsible for them.

This research was funded by the Deutsche Forschungsgemeinschaft (DFG, German Research Foundation) under grant number 417962828.

References

[1] Y. Annadani, J. Rothfuss, A. Lacoste, N. Scherrer, A. Goyal, Y. Bengio, and S. Bauer. Variational causal networks: Approximate Bayesian inference over causal structures. *arXiv:2106.07635 [cs.LG]*, 2021. 5

[2] S. Arik and T. Pfister. TabNet: Attentive interpretable tabular learning. In Q. Yang, K. Leyton-Brown, and Mausam, editors, *Proceedings of the Thirty-Fifth Conference on Artificial Intelligence (AAAI'21)*, pages 6679–6687. AAAI Press, 2021. 1, 7

[3] I. Beltagy, M. Peters, and A. Cohan. Longformer: The long-document transformer. *arXiv:2004.05150 [cs.CL]*, 2020. 14

[4] B. Bischl, G. Casalicchio, M. Feurer, P. Gijsbers, F. Hutter, M. Lang, R. Mantovani, J. van Rijn, and J. Vanschoren. OpenML benchmarking suites. In J. Vanschoren, S. Yeung, and M. Xenochristou, editors, *Proceedings of the Neural Information Processing Systems Track on Datasets and Benchmarks*, 2021. 7, 19

[5] V. Borisov, T. Leemann, K. Seßler, J. Haug, M. Pawelczyk, and G. Kasneci. Deep neural networks and tabular data: A survey. *arXiv:2110.01889 [cs.LG]*, 2021. 7, 16

[6] T. Chen and C. Guestrin. Xgboost: A scalable tree boosting system. In B. Krishnapuram, M. Shah, A. Smola, C. Aggarwal, D. Shen, and R. Rastogi, editors, *Proceedings of the 22nd ACM SIGKDD International Conference on Knowledge Discovery and Data Mining (KDD'16)*, pages 785–794. ACM Press, 2016. 2

[7] T. Chen, S. Kornblith, K. Swersky, M. Norouzi, and G. Hinton. Big self-supervised models are strong semi-supervised learners. *arXiv:2006.10029v2 [cs.LG]*, 2020. 7

[8] N. Erickson, J. Mueller, A. Shirkov, H. Zhang, P. Larroy, M. Li, and A. Smola. Autogluon-tabular: Robust and accurate automl for structured data. *arXiv:2003.06505 [stat.ML]*, 2020. 2, 7

[9] M. Feurer and F. Hutter. Hyperparameter Optimization. In F. Hutter, L. Kotthoff, and J. Vanschoren, editors, *Automated Machine Learning: Methods, Systems, Challenges*, chapter 1, pages 3–38. Springer, 2019. Available for free at http://automl.org/book. 4

[10] M. Feurer, A. Klein, K. Eggensperger, J. Springenberg, M. Blum, and F. Hutter. Efficient and robust automated machine learning. In C. Cortes, N. Lawrence, D. Lee, M. Sugiyama, and R. Garnett, editors, *Proceedings of the 28th International Conference on Advances in Neural Information Processing Systems (NeurIPS'15)*, pages 2962–2970. Curran Associates, 2015. 7

[11] M. Feurer, K. Eggensperger, S. Falkner, M. Lindauer, and F. Hutter. Auto-sklearn 2.0: The next generation. *arXiv:2007.04074[cs.LG]*, 2020. 2

[12] M. Feurer, K. Eggensperger, S. Falkner, M. Lindauer, and F. Hutter. Auto-sklearn 2.0: Hands-free AutoML via meta-learning. *arXiv:2007.04074 [cs.LG]*, 2021. 7

[13] M. Feurer, J. van Rijn, A. Kadra, P. Gijsbers, N. Mallik, S. Ravi, A. Müller, J. Vanschoren, and F. Hutter. OpenML-Python: an extensible Python API for OpenML. *Journal of Machine Learning Research*, 22(100):1–5, 2021. 7

[14] J. Friedman. Greedy function approximation: a gradient boosting machine. *Annals of statistics*, pages 1189–1232, 2001. 1

[15] Y. Gal. *Uncertainty in Deep Learning*. PhD thesis, University of Cambridge, 2016. 2

[16] L. Grinsztajn, E. Oyallon, and G. Varoquaux. Why do tree-based models still outperform deep learning on tabular data? *arXiv:2207.08815 [cs.LG]*, 2022. 7

[17] D. Janzing. Causal regularization. In H. Larochelle, M. Ranzato, R. Hadsell, M.-F. Balcan, and H. Lin, editors, *Proceedings of the 33rd International Conference on Advances in Neural Information Processing Systems (NeurIPS'20)*. Curran Associates, 2020. 4

[18] A. Kadra, M. Lindauer, F. Hutter, and J. Grabocka. Well-tuned simple nets excel on tabular datasets. In M. Ranzato, A. Beygelzimer, K. Nguyen, P. Liang, J. Vaughan, and Y. Dauphin, editors, *Proceedings of the 34th International Conference on Advances in Neural Information Processing Systems (NeurIPS'21)*, pages 23928–23941. Curran Associates, 2021. 1, 7

[19] G. Ke, Q. Meng, T. Finley, T. Wang, W. Chen, W. Ma, Q. Ye, and T.-Y. Liu. Lightgbm: A highly efficient gradient boosting decision tree. In I. Guyon, U. von Luxburg, S. Bengio, H. Wallach, R. Fergus, S. Vishwanathan, and R. Garnett, editors, *Proceedings of the 30th International Conference on Advances in Neural Information Processing Systems (NeurIPS'17)*. Curran Associates, 2017. 2

[20] J. Kossen, N. Band, C. Lyle, A. Gomez, T. Rainforth, and Y. Gal. Self-attention between datapoints: Going beyond individual input-output pairs in deep learning. In M. Ranzato, A. Beygelzimer, K. Nguyen, P. Liang, J. Vaughan, and Y. Dauphin, editors, *Proceedings of the 34th International Conference on Advances in Neural Information Processing Systems (NeurIPS'21)*, pages 28742–28756. Curran Associates, 2021. 1, 7

[21] T. Kyono, Y. Zhang, and M. van der Schaar. CASTLE: Regularization via auxiliary causal graph discovery. In H. Larochelle, M. Ranzato, R. Hadsell, M.-F. Balcan, and H. Lin, editors, *Proceedings of the 33rd International Conference on Advances in Neural Information Processing Systems (NeurIPS'20)*, pages 1501–1512. Curran Associates, 2020. 5

[22] T. Kyono, Y. Zhang, A. Bellot, and M. van der Schaar. Miracle: Causally-aware imputation via learning missing data mechanisms. In M. Ranzato, A. Beygelzimer, K. Nguyen, P. Liang, J. Vaughan, and Y. Dauphin, editors, *Advances in Neural Information Processing Systems*, pages 23806–23817. Curran Associates, 2021. 5

[23] C. Liu, B. Zoph, M. Neumann, J. Shlens, W. Hua, L. Li, L. Fei-Fei, A. Yuille, J. Huang, and K. Murphy. Progressive neural architecture search. In V. Ferrari, M. Herbert, C. Sminchisescu, and Y. Weiss, editors, *14th European Conference on Computer Vision (ECCV'18)*, pages 19–35. Springer, 2018. 4

[24] S. Müller, N. Hollmann, S. Arango, J. Grabocka, and F. Hutter. Transformers can do bayesian inference. In *Proceedings of the International Conference on Learning Representations (ICLR'22)*, 2022. URL `https://openreview.net/forum?id=KSugKcbNf9`. Published online: `iclr.cc`. 2, 3, 6, 8, 17, 18

[25] V. Nair and G. Hinton. Rectified linear units improve restricted Boltzmann machines. In J. Fürnkranz and T. Joachims, editors, *Proceedings of the 27th International Conference on Machine Learning (ICML'10)*. Omnipress, 2010. 16

[26] R. Neal. *Bayesian Learning for Neural Networks*. Lecture Notes in Computer Science. Springer, 1996. 2

[27] J. Pearl. *Causality*. Cambridge University Press, 2 edition, 2009. 2, 4

[28] J. Pearl. Causal inference. *Causality: objectives and assessment*, pages 39–58, 2010. 4

[29] J. Pearl and D. Mackenzie. *The Book of Why*. Basic Books, 2018. 5

[30] F. Pedregosa, G. Varoquaux, A. Gramfort, V. Michel, B. Thirion, O. Grisel, M. Blondel, P. Prettenhofer, R. Weiss, V. Dubourg, J. Vanderplas, A. Passos, D. Cournapeau, M. Brucher, M. Perrot, and E. Duchesnay. Scikit-learn: Machine learning in Python. *Journal of Machine Learning Research*, 12:2825–2830, 2011. 6, 19

[31] J. Peters, D. Janzing, and B. Schölkopf. *Elements of causal inference: foundations and learning algorithms*. The MIT Press, 2017. 2, 4

[32] L. Prokhorenkova, G. Gusev, A. Vorobev, A. Dorogush, and A. Gulin. CatBoost: unbiased boosting with categorical features. In S. Bengio, H. Wallach, H. Larochelle, K. Grauman, N. Cesa-Bianchi, and R. Garnett, editors, *Proceedings of the 31st International Conference on Advances in Neural Information Processing Systems (NeurIPS'18)*. Curran Associates, 2018. 2, 7

[33] C. Rasmussen and C. Williams. *Gaussian Processes for Machine Learning*. The MIT Press, 2006. 7

[34] D. Rothenhäusler, N. Meinshausen, P. Bühlmann, and J. Peters. Anchor regression: heterogeneous data meets causality. *arXiv:1801.06229 [stat.ME]*, 2018. 4

[35] J. Schmidhuber. The speed prior: A new simplicity measure yielding near-optimal computable predictions. In J. Kivinen and R. Sloan, editors, *Computational Learning Theory*, pages 216–228. Springer, 2002. 4

[36] R. Schwartz, J. Dodge, N. Smith, and Etzioni. Green AI. *Communications of the ACM*, 63(12): 54–63, 2020. 14

[37] B. Schölkopf, D. Janzing, J. Peters, E. Sgouritsa, K. Zhang, and J. Mooij. On causal and anticausal learning. In J. Langford and J. Pineau, editors, *Proceedings of the 29th International Conference on Machine Learning (ICML'12)*, pages 459–466. Omnipress, 2012. 4

[38] R. Shwartz-Ziv and A. Armon. Tabular data: Deep learning is not all you need. *Information Fusion*, 81:84–90, 2022. 1, 7, 18, 20

[39] G. Somepalli, M. Goldblum, A. Schwarzschild, C. Bruss, and T. Goldstein. SAINT: Improved neural networks for tabular data via row attention and contrastive pre-training. *arXiv:2106.01342 [cs.LG]*, 2021. 1, 7

[40] N. Srivastava, G. Hinton, A. Krizhevsky, I. Sutskever, and R. Salakhutdinov. Dropout: A simple way to prevent neural networks from overfitting. *Journal of Machine Learning Research*, 15: 1929–1958, 2014. 2

[41] J. Vanschoren, J. van Rijn, B. Bischl, and L. Torgo. OpenML: Networked science in machine learning. *SIGKDD Explorations*, 15(2):49–60, 2014. 7, 19

[42] A. Vaswani, N. Shazeer, N. Parmar, J. Uszkoreit, L. Jones, A. Gomez, L. Kaiser, and I. Polosukhin. Attention is all you need. In I. Guyon, U. von Luxburg, S. Bengio, H. Wallach, R. Fergus, S. Vishwanathan, and R. Garnett, editors, *Proceedings of the 30th International Conference on Advances in Neural Information Processing Systems (NeurIPS'17)*. Curran Associates, Inc., 2017. 1, 3, 8, 18

[43] M. Waldmann and Y. Hagmayer. Causal reasoning. *The Oxford handbook of cognitive psychology*, pages 733—-752, 2013. 4

[44] F. Wenzel, J. Snoek, D., Tran, and R. Jenatton. Hyperparameter ensembles for robustness and uncertainty quantification. In H. Larochelle, M. Ranzato, R. Hadsell, M.-F. Balcan, and H. Lin, editors, *Proceedings of the 33rd International Conference on Advances in Neural Information Processing Systems (NeurIPS'20)*, pages 6514–6527. Curran Associates, 2020. 4

[45] Z. Wojtowicz and S. DeDeo. From probability to consilience: How explanatory values implement bayesian reasoning. *Trends in Cognitive Sciences*, 24(12):981–993, 2020. 4

[46] I. Yeo and R. Johnson. A new family of power transformations to improve normality or symmetry. *Biometrika*, 87(4):954–959, 2000. 16

[47] M. Zaheer, G. Guruganesh, K. Dubey, J. Ainslie, C. Alberti, S. Ontanon, P. Pham, A. Ravula, Q. Wang Qifan, L. Yang, and A. Amr. Big bird: Transformers for longer sequences. In H. Larochelle, M. Ranzato, R. Hadsell, M.-F. Balcan, and H. Lin, editors, *Proceedings of the 33rd International Conference on Advances in Neural Information Processing Systems (NeurIPS'20)*, pages 17283–17297. Curran Associates, 2020. 14

[48] S. Zaidi, A. Zela, T. Elsken, C. Holmes, F. Hutter, and Y. Teh. Neural ensemble search for uncertainty estimation and dataset shift. In M. Ranzato, A. Beygelzimer, K. Nguyen, P. Liang, J. Vaughan, and Y. Dauphin, editors, *Proceedings of the 34th International Conference on Advances in Neural Information Processing Systems (NeurIPS'21)*, pages 7898–7911. Curran Associates, 2021. 4

[49] K. Zhang, B. Schölkopf, P. Spirtes, and C. Glymour. Learning causality and causality-related learning: some recent progress. *National science review*, 5(1):26–29, 2018. 5

[50] B. Zoph and Q. V. Le. Neural Architecture Search with reinforcement learning. In *Proceedings of the International Conference on Learning Representations (ICLR'17)*, 2017. Published online: `iclr.cc`. 4

A Limitations

The runtime and memory usage of the Transformer-based PFN architecture used in this work scales quadratically with the number of inputs, i.e. training samples passed. Thus, inference on larger sequences ($> 100\,000$) is hard on current consumer GPUs. A growing number of methods seek to tackle this issue and report similar performances while scaling linearly with the number of inputs [47, 3]. These methods can be integrated in the PFN architecture and thus into TabPFN. Furthermore, in our experiments we limit the number of features to 100 and the number of classes to 10 as described in Section 4. While this choice is flexible, our fitted TabPFN can not work with datasets that go beyond these limits.

B Societal Implications

In terms of broader societal impact of this work, we do not see any foreseeable strongly negative impacts. However, this paper could positively impact the carbon footprint and accessibility of learning algorithms. The computations required for machine learning research have been doubling every few months, resulting in a large carbon footprint [36]. Moreover, the financial cost of the computations can make it difficult for academics, students, and researchers to apply these methods. The decreased computational time shown by TabPFN translates to gains in CO2 emissions and cost, making it available to an audience that does not have access to larger scale computing.

C Additional Results

C.1 Detailed Tabular Results

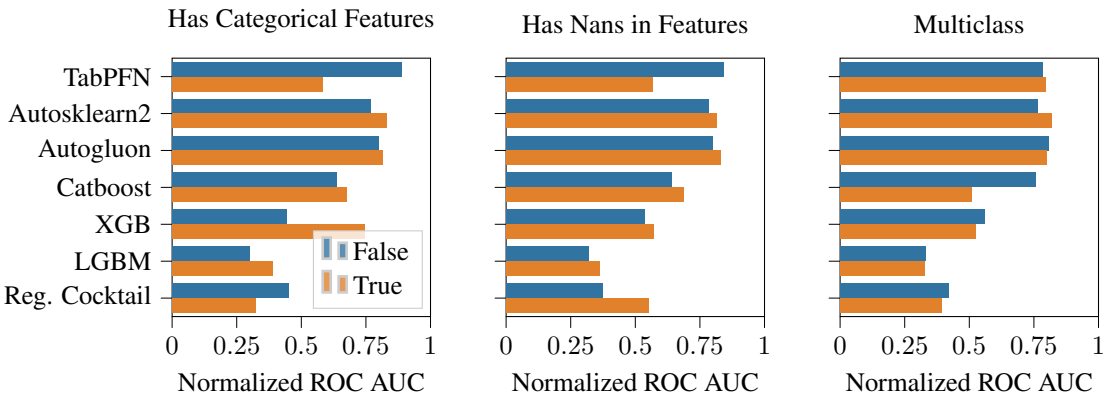


Figure 6: Normalized ROC AUC performance on datasets from the OpenML-CC18 Benchmark, divided by dataset characteristics. For each plot, we split the datasets into two groups. Left: Blue bars indicate the performance on datasets, that have categorical features. Middle: Blue bars indicate more than 20 features, while others have less. Right: Blue bars indicate multiclass datasets, while others are binary.

In Figure 6, we explore how the kind of dataset evaluated affects the performance of TabPFN, compared to our baselines. We find, that TabPFN performs much better, when no categorical features are present. This warrants the extension of our prior, to make it more customized towards categorical data. We also find, that TabPFN performs better in a multiclass setting and when a relatively large number of features is provided.

In addition to the results in the main paper in Section 4.2, we report a wide range of performance values and per dataset results in Table 1.

Table 1: ROC AUC OVO results on the small OpenML-CC18 for 60 minutes requested time per dataset and per split. If available, all baselines are given ROC AUC optimization as an objective, others optimize CE. Overall each method got a time budget of 150 hours, but not all methods used the full budget.

	LightGBM	CatBoost	XGBoost	ASKL2.0	AutoGluon	TabPFN$_{n.e.}$	TabPFN	TabPFN + AutoGluon
balance-scale	0.9938	0.9245	0.9939	0.997	0.9919	0.9965	**0.9973**	0.9958
mfeat-fourier	0.9786	0.9816	0.9803	0.9826	**0.9843**	0.9767	0.9811	0.9838
breast-w	0.991	0.9931	0.9896	0.9939	0.9933	0.9931	0.9934	**0.994**
mfeat-karhunen	0.9979	0.9986	0.9983	0.9975	**0.9987**	0.9939	0.9978	0.9985
mfeat-morphologica..	0.9601	0.9629	0.9612	0.9671	0.9698	0.9657	0.9669	**0.9722**
mfeat-zernike	0.9716	0.9759	0.9735	0.9812	**0.9908**	0.9812	0.9823	0.9901
cmc	0.7288	0.7256	0.7299	**0.7378**	0.7331	0.7233	0.7276	0.7336
credit-approval	0.9415	0.9389	**0.9422**	0.9406	0.9415	0.9253	0.9322	0.9394
credit-g	0.7684	0.7852	0.7853	0.793	0.7941	0.7894	0.7894	**0.7948**
diabetes	0.8247	0.8383	0.8378	0.8343	0.8391	0.8412	0.841	**0.8427**
tic-tac-toe	0.9988	0.9992	**1**	0.9943	**1**	0.9547	0.9759	0.9992
vehicle	0.9232	0.9302	0.9282	0.9504	0.9416	0.9568	**0.9589**	0.9538
eucalyptus	0.8931	0.8979	0.9004	0.9132	0.9204	0.9218	0.9245	**0.9278**
analcatdata_author..	0.9999	0.9999	0.9997	**1**	0.9993	**1**	**1**	**1**
analcatdata_dmft	0.5461	0.5589	0.5743	0.5752	0.5657	0.5643	**0.579**	0.5756
pc4	0.9301	0.9413	0.9291	0.9331	0.9428	0.9298	0.9383	**0.944**
pc3	0.8178	0.8247	0.8288	0.8265	0.8282	0.8308	**0.8373**	0.836
kc2	0.8141	0.8323	0.8227	0.8311	0.8242	0.8322	**0.8346**	0.8321
pc1	0.8321	0.86	0.8489	0.8527	0.8578	**0.877**	0.8761	0.8739
banknote-authentic..	**1**	**1**	**1**	**1**	**1**	**1**	**1**	**1**
blood-transfusion-..	0.7144	0.7403	0.7312	0.7504	0.7364	0.753	**0.7549**	0.7469
ilpd	0.6917	0.7279	0.7171	0.7212	0.723	**0.7412**	0.7379	0.7326
qsar-biodeg	0.9126	0.9217	0.9191	0.9247	0.9276	**0.9345**	0.9336	0.9336
wdbc	0.9904	0.9931	0.9904	0.9947	0.9956	0.996	**0.9964**	0.996
cylinder-bands	0.8556	0.8757	0.8782	0.8718	**0.8878**	0.8314	0.8336	0.8751
dresses-sales	0.5593	0.5696	**0.5823**	0.5705	0.5507	0.5333	0.5376	0.5509
MiceProtein	0.9997	0.9999	0.9998	0.9999	**1**	0.9997	0.9999	**1**
car	0.9925	0.9955	0.9948	**0.998**	0.997	0.9926	0.995	0.9972
steel-plates-fault..	0.9626	0.9655	0.9656	**0.9694**	0.9666	0.9619	0.9655	0.9687
climate-model-simu..	0.9286	0.9344	0.9255	0.9291	0.9391	**0.9426**	0.9415	0.9421
Wins AUC OVO	0	0	2	2	2	4	**5**	**5**
Wins Acc.	0	2	2	3	3	0	6	**8**
Wins CE	0	1	3	1	7	1	6	**9**
M. rank AUC OVO	6.6167	4.9667	5.4167	4.05	3.7833	4.65	3.7	**2.8167**
Mean rank Acc.	6.5333	4.9833	5.1833	4.8667	3.8167	4.5333	3.6167	**2.4667**
Mean rank CE	5.7333	5.6	5.4667	5.8	2.8667	4.6167	3.5333	**2.3833**
Win/T/L AUC vs Tab..	5/4/21	9/4/17	6/5/19	10/6/14	13/4/13	4/8/18	–/–/–	15/7/8
Win/T/L Acc vs Tab..	6/0/24	9/1/20	11/0/19	11/2/17	12/0/18	6/3/21	–/–/–	19/3/8
Win/T/L CE vs TabP..	6/0/24	8/0/22	8/0/22	8/0/22	20/0/10	1/4/25	–/–/–	23/0/7
Mean AUC OVO	0.884±.012	0.89±.011	0.891±.011	0.894±.01	0.895±.01	0.891±.01	0.894±.01	**0.898**±.0097
Mean Acc.	0.815±.014	0.818±.011	0.821±.013	0.821±.016	0.83±.012	0.82±.013	0.825±.012	**0.834**±.011
Mean CE	0.782±.074	0.767±.061	0.758±.047	0.815±.06	**0.72**±.015	0.742±.021	0.732±.018	0.721±.015
Time Tune + Train (s)	3241	3718	3304	3601	3127	**0.0187**	0.4197	3127
Predict (s)	0.0815	0.0168	0.0685	1.224	21.18			

D Details of the TabPFN Prior

D.1 SCM Prior

The Sampling Algorithm We instantiate a subfamily of DAGs that can be efficiently sampled from by starting with a MLP architecture and dropping weights from it. That is, to sample a dataset with k features and n samples from our prior we perform the following steps for each dataset:

(1) We sample the number of MLP layers $l \sim p(l)$ and nodes $h \sim p(h)$ and sample a graph $\mathcal{G}(Z, E)$ structured like an l-layered MLP with hidden size h.

(2) We sample weights for each Edge E_{ij} as $W_{i;j} \sim p_w(\cdot)$.

(3) We drop a random set of edges $e \in E$ to yield a random DAG.

(4) We sample a set of k feature nodes N_x and a label node N_y from the nodes Z.

(5) We sample the noise distributions $p(\epsilon) \sim p(p(\epsilon))$ from a meta-distribution.This yields an SCM, with all f_i's instantiated as random affine mappings followed by an activation. Each z_i corresponds to a sparsely connected neuron in the MLP.

With the above parameters fixed, we perform the following steps for each member of the dataset:

(1) We sample noise variables ϵ_i from their specific distributions.

(2) We compute the value of all $z \in Z$ with $z_i = a((\sum_{j \in \mathrm{PA}_{G(i)}} E_{ij} z_j) + \epsilon_i)$.

(3) We retrieve the values at the feature nodes N_x and the output node N_y and return them.

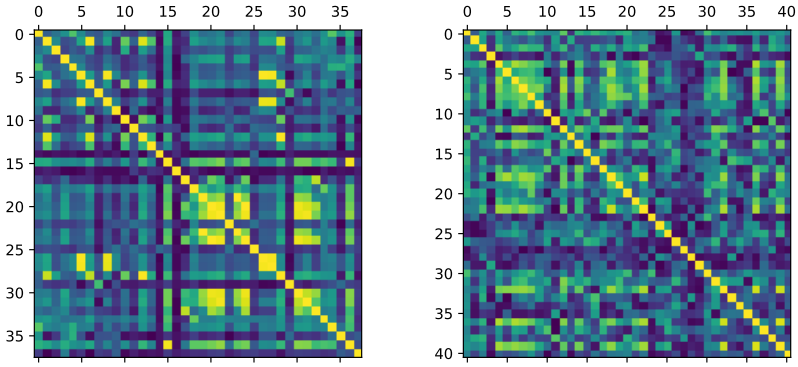


Figure 7: Feature correlation matrices for a real-world ("PC4 Software defect prediction", left) and a synthetic (right) dataset, where brighter colors indicate higher correlation.

We sample one activation function a per dataset from $\{ReLU, Tanh, Identity\}$ [25]. The sampling scheme for the number of layers $p(l)$ and nodes $p(h)$ is designed to follow a log-normal distributions, the dropout rate follows a beta distribution and $p(p(\epsilon))$ samples normal distributions with normally distributed mean and standard deviation.

D.2 Tabular Data Refinements

Tabular datasets comprise a range of peculiarities, e.g. feature types can be numerical, ordinal, or categorical and feature values can be missing leading to sparse features. We seek to reflect these peculiarities in the design of our prior as described in the following sections.

D.2.1 Preprocessing

During meta-training, input data is normalized to zero mean and variance and we apply the same step when evaluating on real data. Since tabular data frequently contains exponentially scaled data, which is not might not be present during meta-training, we apply power scaling during inference [46]. Thus, during inference on real tabular datasets the features more closely match those seen during meta-training. We use only training samples for calculating z-statistics, power transforms and all other preprocessing. We take this preprocessing time is into account when reporting the inference time of our method.

D.2.2 Correlated Features

Feature correlation in tabular data varies between datasets and ranges from independent to highly correlated. This poses problems to classical deep learning methods [5]. When considering a large space of SCMs correlated features of varying degrees naturally arise in our priors. Furthermore, in real-world tabular data the ordering of features is often unstructured, however adjacent features are often more highly correlated than others. We use "Blockwise feature sampling" to reflect the correlation structure between ordered features. Our generation method of SCMs naturally provides a way to do this. The first step in generating our SCMs is generating a unidirectional layered network structure in which nodes in one layer can only receive inputs from the preceding layer. Thus, features in one layer tend to be higher correlated. We use this by sampling adjacent nodes in the layered network structure in blocks and using these ordered blocks in our set of features. In Figure 7, we visualize the correlations of such a generated dataset (right) and compare them to a real-world dataset (left), demonstrating that our prior yields correlation structures similar to those of real datasets.

D.2.3 Generating irregular functions

In real-world data, some features are consistently more important than others. While a random network weight initialization leads to slightly different feature importances, the average effect of input features regresses to the mean when the hidden dimensionality increases. We amplify differences

by sampling a weight parameter for each input feature and multiplying all outgoing weights by this factor. In the prior, we randomly sparsify connections of the graph. Thus hidden variables and the output node are influenced by fewer parameters per step, yielding more irregular patterns, as a larger number of parameters once again regresses to the mean. Here, we extend sparsification to blocks of variables, leading to some groups of variables interacting more strongly. We also extend the way noise variables are sampled. Instead of sampling Gaussian noise at each node from the same distribution, we first sample noise means and standard deviations for each node and then sample from this distribution. Also, we generate input data x, which is distributed non-uniformly, as observed in real-world data. We sample the input variables x, that are propagated through our network, from a mix of distributions such as the Gaussian, Zipfian and Multivariate Distribution.

D.2.4 Nan Handling

To model missing values in our prior, we introduce missing values probabilistically during prior training. For each dataset, we sample a binary decision variable M, indicating whether it will contain missing features and next sample a fraction f_m of missing values. If M is positive, we drop out a fraction f_m of feature values uniformly at random. We introduce a binary missing value mask, indicating the positions of missing values, and pass it to our model alongside the feature embedding. We append this mask during meta-training on synthetic data and inference on actual datasets.

D.2.5 Categorical Features

Tabular data often includes not only numeric features but also discrete categorical ones. A categorical feature can be ordered, i.e. the categories represent binned degrees of some underlying variable, or shuffled. We introduce categorical features by picking a random fraction p_{cat} (a hyperparameter) of categorical features per dataset. Analogous to transforming numeric class labels to discrete multiclass labels, we convert dense features to discrete ones. Also analogous to multiclass labels, we pick a shuffling fraction of categorical features p_{scat} where we reshuffle categories. For details see 3.5. Our experiments regarding categorical features, however, showed no significant improvement. Finding more appropriate ways of handling categorical features could be a line of future research.

E Details of Prior-Data Fitted Network Algorithm

Algorithm 1 describes the training method proposed by Müller et al. [24] for PFNs.

Algorithm 1: Meta-Training of a PFN [24]

Input : A prior distribution over datasets $p(D)$, from which samples can be drawn and the number of samples K to draw
Output : A model q that will approximate the PPD
Initialize the neural network q ;
for $j \leftarrow 1$ **to** K **do**
  Sample $D \cup \{(x_i, y_i)\}_{i=1}^{m} \sim p(D)$;
  Compute stochastic loss approximation $\bar{\ell} = \sum_{i=1}^{m}(-\log q\ (y_i|x_i, D))$;
  Update parameters θ with stochastic gradient descent on $\nabla\ \bar{\ell}$;
end

F Setup of our method

F.1 Transformer Hyperparameters

We considered only PFN Transformers with 12 layers, embeddings size 512, hidden size 1024 in feed-forward layers, and 4-head attention. For each training we tested a set of 3 learning rates, $\{.001, .0003, .0001\}$, and used the one with the lowest final training loss. The resulting model contains 25.82 M parameters.

F.2 PFN Architecture Adaptations

Attention Adaption The original PFN architecture [24] uses a single multi-head self-attention module [42] to compute the attention between all the training examples, as well as, the attention from validation examples to training examples. We replaced this, with two modules that share weights, one which computes self-attention among the training examples and the other that only compute cross-attention from validation examples to training examples. Conceptually, that is equivalent to using a slightly different self-attention mask than the original architecture, which allowed everything to attend to itself (the diagonal is 1), like in this example

$$\begin{bmatrix} 1 & 1 & 1 & 0 & 0 \\ 1 & 1 & 1 & 0 & 0 \\ 1 & 1 & 1 & 0 & 0 \\ 1 & 1 & 1 & 1 & 0 \\ 1 & 1 & 1 & 0 & 1 \end{bmatrix}. \tag{4}$$

We remove the attention to themselves for validation examples. In terms of the example above:

$$\begin{bmatrix} 1 & 1 & 1 & 0 & 0 \\ 1 & 1 & 1 & 0 & 0 \\ 1 & 1 & 1 & 0 & 0 \\ 1 & 1 & 1 & 0 & 0 \\ 1 & 1 & 1 & 0 & 0 \end{bmatrix}. \tag{5}$$

Information about the state of the current position, does still flow through the residual branch, though.

Flexible Encoder Datasets have unequal numbers of input dimensions (features), while PFNs use an encoder layer that accepts fixed dimensional inputs. Here we explain how datasets with different numbers of dimensions can be modelled with a single PFN: We draw the number of dimensions of a dataset during training uniformly at random up to 100. Our encoder changes to accomodate this training and inference with different numbers of features by zero-padding datasets where the number of features k is smaller than the maximum number of features K and scaling these features by $\frac{K}{k}$, s.t. the magnitude stays the same.

F.3 TabPFN Training

We trained our final model for $18\,000$ steps with a batch size of 512 datasets. That is our TabPFN is trained on $9\,216\,000$ synthetically generated datasets. This training takes 20 hours on 8 GPUs (Nvidia RTX 2080 Ti). Each dataset had a fixed size of 1024 and we split it into training and validation uniformly at random. We generally saw that learning curves tended to flatten after around 10 million datasets and where generally very noisy. Likely, this is because our prior generates a wide variety of different datasets.

F.4 Prior Hyperparameters

G Details for Tabular Experiments

Here we provide additional details for the experiments conducted in Section 4 in the main paper.

G.1 Hardware Setup

All evaluations, including the baselines, ran on a compute cluster equipped with Intel(R) Xeon(R) Gold 6242 CPU @ 2.80GHz using 1 CPU with up to 6GB RAM. For evaluation using our TabPFN, we additionally use a RTX 2080 Ti.

G.2 Baselines

We provide the search space used to tune our baselines in Table 3. For *CatBoost* and *XGBBoost*, we used the same ranges as Shwartz-Ziv and Armon [38] with the following exceptions: For *CatBoost* we removed the hyperparameter max_depth since it is not in documentation, for *CatBoost* and *XGBBoost*, we set the range for n_estimators to be in $[100, 4000]$. The search spaces for the KNN, GP and

Table 2: Overview of our prior hyperparameter distribution.

	Sampling distribution $p(\ )$	Minimum	Maximum		
MLP weight dropout	Beta	0.1	5.0	0.9	
		Choices			
Sample SCM vs BNN	Choice	[True, False]			
Share Noise mean for nodes	Choice	[True, False]			
Input feature scaling enabled	Choice	[True, False]			
Sample y from last MLP layer	Choice	[True, False]			
MLP Activation Functions	Choice	[Tanh, ReLU, ELU]			
MLP Activation Functions	Choice	, Identity, Threshold]			
Blockwise Dropout	Choice	[True, False]			
Keep SCM feature order	Choice	[True, False]			
Sample feature nodes blockwise	Choice	[True, False]			
GP noise	Choice	[1e-05, 0.0001, 0.01]			
		Max Mean	Min Mean	Round HP	Min
MLP #layers	Trunc. Normal	6	1	True	2
MLP #hidden nodes per layer	Trunc. Normal	130	5	True	4
Gaussian Noise Std.	Trunc. Normal	0.3	0.0001	False	0.0
MLP Weights Std.	Trunc. Normal	10.0	0.01	False	0.0
SCM #nodes at layer 1	Trunc. Normal	12	1	True	1
GP outputscale	Trunc. Normal	10.0	0.00001	False	0
GP lengthscale	Trunc. Normal	10.0	0.00001	False	0

Logistic Regression baselines were designed from scratch and we used the respective implementation from *scikit-learn* [30]. For *CatBoost* and *AutoSklearn* we pass the position of categorical features to the classifier (*AutoGluon* automatically detects categorical feature columns). We normalize inputs for Logistic Regression, GP and KNN to the range [0, 1] using MinMax Scaling.

G.3 Used Datasets

To construct and evaluate our method, we use three disjoint sets of datasets. Our test-set (see Table 4), a subset of the OpenML-CC18 benchmark suite [4] and a meta-set (see Table 5 and 6), which we collected from `OpenML.org` [41]. These are licensed under the BSD 3-Clause license.

For the test-set, we considered all datasets in the OpenML-CC18 benchmark suite with less than $2\,000$ samples, 100 features or 10 classes, which leaves us with 30 datasets that represent small, tabular datasets. For the meta-set, we considered all datasets on `OpenML.org` and applied the following filtering procedure: We dropped all datasets that are in the test-set and all datasets with more than $1\,000$ samples, 100 features or 10 classes. We also manually checked for overlaps and removed datasets where the number of features, classes and samples was identical to a dataset in the test-set. Furthermore, we manually dropped FOREX (since it is a time series dataset) and artificially created datasets, such as Univ and Friedman datasets. The remaining meta-set then contains of 150 datasets.

H Code

Our code alongside notebooks to reproduce our experiments and pretrained models are available at `https://anonymous.4open.science/r/TabPFN-2AEE`.

We also created two demos. One to experiment with the TabPFNs predictions (`https://huggingface.co/spaces/TabPFN/TabPFNPrediction`) and one to check cross-validation ROC AUC scores on new datasets (`https://huggingface.co/spaces/TabPFN/TabPFNEvaluation`) both of them run on a weak CPU, thus it can require a little bit of time. Both demos are based on a scikit-learn interface that makes using the TabPFN as easy as an scikit-learn SVM.

Table 3: Hyperparameter spaces for baselines. All, except LightGBM, adapted from Shwartz-Ziv and Armon [38].

baseline	name	type	log	range
LogReg	penalty	cat	(l1, l2, none)	-
	max_iter	int	[50, 500]	-
	fit_intercept	cat	(True, False)	-
	C	float	[e^{-5}, 5]	-
KNN	n_neighbors	int	[1, 16]	-
GP	params_y_scale	float	[0.05, 5.0]	yes
	params_length_scale	float	[0.1, 1.0]	yes
CatBoost	learning_rate	float	[e^{-5}, 1]	yes
	random_strength	int	[1, 20]	-
	l2_leaf_reg	float	[1, 10]	yes
	bagging_temperature	float	[0, 1.0]	yes
	leaf_estimation_iterations	int	[1, 20]	-
	iterations	int	[100, 4000]	-
XGBoost	learning_rate	float	[e^{-7}, 1]	yes
	max_depth	int	[1, 10]	-
	subsample	float	[0.2, 1]	-
	colsample_bytree	float	[0.2, 1]	-
	colsample_bylevel	float	[0.2, 1]	-
	min_child_weight	float	[e^{-16}, e^{5}]	yes
	alpha	float	[e^{-16}, e^{2}]	yes
	lambda	float	[e^{-16}, e^{2}]	yes
	gamma	float	[e^{-16}, e^{2}]	yes
	n_estimators	int	[100, 4000]	-
LightGBM	num_leaves	int	[5, 50]	yes
	max_depth	int	[3, 20]	yes
	learning_rate	float	[e^{-3}, 1]	-
	n_estimators	int	50, 2000	-
	min_child_weight	float	[e^{-5}, e^{4}]	yes
	reg_alpha	float	[0, 1e-1, 1, 2, 5, 7, 10, 50, 100]	yes
	reg_lambda	float	[0, 1e-1, 1, 5, 10, 20, 50, 100]	yes
	subsample	float	[0.2, 0.8]	-

Table 4: Datasets used for the evaluation.

Name	#Feat.	#Cat.	#Inst.	#Class.	#NaNs	Minor. Class Size	OpenML ID
balance-scale	5	1	625	3	0	49	11
mfeat-fourier	77	1	2000	10	0	200	14
breast-w	10	1	699	2	16	241	15
mfeat-karhunen	65	1	2000	10	0	200	16
mfeat-morphological	7	1	2000	10	0	200	18
mfeat-zernike	48	1	2000	10	0	200	22
cmc	10	8	1473	3	0	333	23
credit-approval	16	10	690	2	67	307	29
credit-g	21	14	1000	2	0	300	31
diabetes	9	1	768	2	0	268	37
tic-tac-toe	10	10	958	2	0	332	50
vehicle	19	1	846	4	0	199	54
eucalyptus	20	6	736	5	448	105	188
analcatdata_auth...	71	1	841	4	0	55	458
analcatdata_dmft	5	5	797	6	0	123	469
pc4	38	1	1458	2	0	178	1049
pc3	38	1	1563	2	0	160	1050
kc2	22	1	522	2	0	107	1063
pc1	22	1	1109	2	0	77	1068
banknote-authenti...	5	1	1372	2	0	610	1462
blood-transfusion-...	5	1	748	2	0	178	1464
ilpd	11	2	583	2	0	167	1480
qsar-biodeg	42	1	1055	2	0	356	1494
wdbc	31	1	569	2	0	212	1510
cylinder-bands	40	22	540	2	999	228	6332
dresses-sales	13	12	500	2	835	210	23381
MiceProtein	82	5	1080	8	1396	105	40966
car	7	7	1728	4	0	65	40975
steel-plates-fault	28	1	1941	7	0	55	40982
climate-model-simu...	21	1	540	2	0	46	40994

Table 5: Meta-Datasets used for tuning the prior.

Name	#Feat.	#Cat.	#Inst.	#Class.	#NaNs	Minor. Class Size	OpenML ID
breast-cancer	10	10	286	2	9	85	13
colic	27	20	368	2	1927	136	25
dermatology	35	34	366	6	8	20	35
sonar	61	1	208	2	0	97	40
glass	10	1	214	6	0	9	41
haberman	4	2	306	2	0	81	43
tae	6	3	151	3	0	49	48
heart-c	14	8	303	2	7	138	49
heart-h	14	8	294	2	782	106	51
heart-statlog	14	1	270	2	0	120	53
hepatitis	20	14	155	2	167	32	55
vote	17	17	435	2	392	168	56
ionosphere	35	1	351	2	0	126	59
iris	5	1	150	3	0	50	61
wine	14	1	178	3	0	48	187
flags	29	27	194	8	0	4	285
hayes-roth	5	1	160	3	0	31	329
monks-problems-1	7	7	556	2	0	278	333
monks-problems-2	7	7	601	2	0	206	334
monks-problems-3	7	7	554	2	0	266	335
SPECT	23	23	267	2	0	55	336
SPECTF	45	1	349	2	0	95	337
grub-damage	9	7	155	4	0	19	338
synthetic_control	61	1	600	6	0	100	377
prnn_crabs	8	2	200	2	0	100	446
analcatdata_lawsuit	5	2	264	2	0	19	450
irish	6	4	500	2	32	222	451
analcatdata_broadwaymult	8	5	285	7	27	21	452
analcatdata_reviewer	8	8	379	4	1418	54	460
backache	32	27	180	2	0	25	463
prnn_synth	3	1	250	2	0	125	464
schizo	15	3	340	2	834	163	466
profb	10	5	672	2	1200	224	470
analcatdata_germangss	6	5	400	4	0	100	475
biomed	9	2	209	2	15	75	481
rmftsa_sleepdata	3	1	1024	4	0	94	679
diggle_table_a2	9	1	310	9	0	18	694
rmftsa_ladata	11	1	508	2	0	222	717
pwLinear	11	1	200	2	0	97	721
analcatdata_vineyard	4	2	468	2	0	208	724
machine_cpu	7	1	209	2	0	56	733
pharynx	11	10	195	2	2	74	738
auto_price	16	2	159	2	0	54	745
servo	5	5	167	2	0	38	747
analcatdata_wildcat	6	3	163	2	0	47	748
pm10	8	1	500	2	0	246	750
wisconsin	33	1	194	2	0	90	753
autoPrice	16	1	159	2	0	54	756
meta	22	3	528	2	504	54	757
analcatdata_apnea3	4	3	450	2	0	55	764
analcatdata_apnea2	4	3	475	2	0	64	765
analcatdata_apnea1	4	3	475	2	0	61	767
disclosure_x_bias	4	1	662	2	0	317	774
bodyfat	15	1	252	2	0	124	778
cleveland	14	8	303	2	6	139	786
triazines	61	1	186	2	0	77	788
disclosure_x_tampered	4	1	662	2	0	327	795
cpu	8	2	209	2	0	53	796
cholesterol	14	8	303	2	6	137	798
chscase_funds	3	1	185	2	0	87	801
pbcseq	19	7	1945	2	1133	972	802
pbc	19	9	418	2	1239	188	810
rmftsa_ctoarrivals	3	2	264	2	0	101	811
chscase_vine2	3	1	468	2	0	212	814
chatfield_4	13	1	235	2	0	93	820
boston_corrected	21	4	506	2	0	223	825
sensory	12	12	576	2	0	239	826
disclosure_x_noise	4	1	662	2	0	329	827
autoMpg	8	4	398	2	6	189	831
kdd_el_nino-small	9	3	782	2	466	274	839
autoHorse	26	9	205	2	57	83	840
stock	10	1	950	2	0	462	841
breastTumor	10	9	286	2	9	120	844
analcatdata_gsssexsurvey	10	6	159	2	6	35	852
boston	14	2	506	2	0	209	853
fishcatch	8	3	158	2	87	63	854
vinnie	3	1	380	2	0	185	860
mu284	11	1	284	2	0	142	880
no2	8	1	500	2	0	249	886
chscase_geyser1	3	1	222	2	0	88	895
chscase_census6	7	1	400	2	0	165	900
chscase_census5	8	1	400	2	0	193	906
chscase_census4	8	1	400	2	0	194	907
chscase_census3	8	1	400	2	0	192	908
chscase_census2	8	1	400	2	0	197	909
plasma_retinol	14	4	315	2	0	133	915
visualizing_galaxy	5	1	323	2	0	148	925
colleges_usnews	34	2	1302	2	7830	614	930

Table 6: Meta-Datasets used for tuning the prior (continued).

Name	#Feat.	#Cat.	#Inst.	#Class.	#NaNs	Minor. Class Size	OpenML ID
disclosure_z	4	1	662	2	0	314	931
socmob	6	5	1156	2	0	256	934
chscase_whale	9	1	228	2	20	111	939
water-treatment	37	16	527	2	542	80	940
lowbwt	10	8	189	2	0	90	941
arsenic-female-bladder	5	2	559	2	0	80	949
analcatdata_halloffame	17	2	1340	2	20	125	966
analcatdata_birthday	4	3	365	2	30	53	968
analcatdata_draft	5	3	366	2	1	32	984
collins	23	3	500	2	0	80	987
prnn_fglass	10	1	214	2	0	76	996
jEdit_4.2_4.3	9	1	369	2	0	165	1048
mc2	40	1	161	2	0	52	1054
mw1	38	1	403	2	0	31	1071
jEdit_4.0_4.2	9	1	274	2	0	134	1073
PopularKids	11	5	478	3	0	90	1100
teachingAssistant	7	5	151	3	0	49	1115
lungcancer_GSE31210	24	3	226	2	0	35	1412
MegaWatt1	38	1	253	2	0	27	1442
PizzaCutter1	38	1	661	2	0	52	1443
PizzaCutter3	38	1	1043	2	0	127	1444
CostaMadre1	38	1	296	2	0	38	1446
CastMetal1	38	1	327	2	0	42	1447
KnuggetChase3	40	1	194	2	0	36	1448
PieChart1	38	1	705	2	0	61	1451
PieChart3	38	1	1077	2	0	134	1453
parkinsons	23	1	195	2	0	48	1488
planning-relax	13	1	182	2	0	52	1490
qualitative-bankruptcy	7	7	250	2	0	107	1495
sa-heart	10	2	462	2	0	160	1498
seeds	8	1	210	3	0	70	1499
thoracic-surgery	17	14	470	2	0	70	1506
user-knowledge	6	1	403	5	0	24	1508
wholesale-customers	9	2	440	2	0	142	1511
heart-long-beach	14	1	200	5	0	10	1512
robot-failures-lp5	91	1	164	5	0	21	1520
vertebra-column	7	1	310	3	0	60	1523
Smartphone-Based...	68	2	180	6	0	30	4153
breast-cancer-...	10	10	277	2	0	81	23499
LED-display-...	8	1	500	10	0	37	40496
GAMETES_Epistasis...	21	21	1600	2	0	800	40646
calendarDOW	33	21	399	5	0	44	40663
corral	7	7	160	2	0	70	40669
mofn-3-7-10	11	11	1324	2	0	292	40680
thyroid-new	6	1	215	3	0	30	40682
solar-flare	13	13	315	5	0	21	40686
threeOf9	10	10	512	2	0	238	40690
xd6	10	10	973	2	0	322	40693
tokyo1	45	3	959	2	0	346	40705
parity5_plus_5	11	11	1124	2	0	557	40706
cleve	14	9	303	2	0	138	40710
cleveland-nominal	8	8	303	5	0	13	40711
Australian	15	9	690	2	0	307	40981
DiabeticMellitus	98	1	281	2	2	99	41430
conference_attendance	7	7	246	2	0	31	41538
CPMP-2015-...	23	1	527	4	0	78	41919
TuningSVMs	81	1	156	2	0	54	41976
regime_alimentaire	20	17	202	2	17	41	42172
iris-example	5	1	150	3	0	50	42261
Touch2	11	1	265	8	0	27	42544
penguins	7	3	344	3	18	68	42585
titanic	8	5	891	2	689	342	42638

Table 7: Evaluation datasets for model generalization experiments.

Name	#Feat.	#Cat.	#Inst.	#Class.	#NaNs	Minor. Class Size	OpenML ID
KDDCup09_appetency	231	39	50000	2	8024152	890	1111
airlines	8	5	539383	2	0	240264	1169
bank-marketing	17	10	45211	2	0	5289	1461
nomao	119	30	34465	2	0	9844	1486
adult	15	9	48842	2	6465	11687	1590
covertype	55	45	581012	7	0	2747	1596
numerai28.6	22	1	96320	2	0	47662	23517
connect-4	43	43	67557	3	0	6449	40668
jungle_chess_2pcs:.	7	1	44819	3	0	4335	41027
APSFailure	171	1	76000	2	1078695	1375	41138
albert	79	53	425240	2	2734000	212620	41147
MiniBooNE	51	1	130064	2	0	36499	41150
guillermo	4297	1	20000	2	0	8003	41159
riccardo	4297	1	20000	2	0	5000	41161
volkert	181	1	58310	10	0	1361	41166
dionis	61	1	416188	355	0	878	41167
jannis	55	1	83733	4	0	1687	41168
helena	28	1	65196	100	0	111	41169