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

Noah Hollmann*, Samuel Müller*, Katharina Eggensperger and Frank Hutter

* Equal contribution
Neural Networks for Small Tabular Datasets

Premises

- **Neural Networks** excel for **large amounts of data** or given effective inductive biases (such as CNNs for images), but:
  - Overfit on small datasets unless regularized properly (e.g., [1])
  - For tabular data identifying effective inductive biases is challenging

- **The long tail of Machine learning datasets:**
  Many real-world datasets are small, but most ML research datasets are large (e.g., [2])

**Idea:** Model inductive biases and regularize using Bayesian priors. Bayesian inference is enabled through Prior Fitted Networks.

Bayesian Supervised Learning

Prior $p(t)$:
With latents $t$: parameterizing functions

Posterior:

$$p(t|D) = \frac{p(D|t)p(t)}{\int p(D|t)dt}$$

Posterior predictive distribution:

$$p(y|x, D) = \int p(y|x, t)p(t|D)dt$$

Intractable exactly!
Prior-Data Fitted Networks (PFNs)

A Permutation-Invariant Transformer

Prior work

Prior-Data Fitted Networks (PFNs)

Prior-fitting

\[
\text{Sample prior datasets } D^{(i)} \sim p(D)
\]

\[
D^{(1)} = D^{(1)}_{\text{train}} \cup \{(x^{(1)}_{\text{test}}, y^{(1)}_{\text{test}})\}
\]

\[
\vdots
\]

\[
D^{(K)} = D^{(K)}_{\text{train}} \cup \{(x^{(K)}_{\text{test}}, y^{(K)}_{\text{test}})\}
\]

Train the PFN by minimizing

\[
-\sum_{i=1}^{K} \log q\theta(y^{(i)}_{\text{test}} | x^{(i)}_{\text{test}}, D^{(i)}_{\text{train}})
\]

PFN with parameters \(\theta^*\)

Prior-Data Fitted Networks (PFNs)

Prior-fitting & Inference

Sample prior datasets $D^{(i)} \sim p(D)$

$D^{(1)} = D_{train}^{(1)} \cup \{(x_{test}^{(1)}, y_{test}^{(1)})\}$

$\vdots$

$D^{(K)} = D_{train}^{(K)} \cup \{(x_{test}^{(K)}, y_{test}^{(K)})\}$

Actual training dataset and test input

$(D_{train}, x_{test})$

Train the PFN by minimizing

$$-\sum_{i=1}^{K} \log q_\theta(y_{test}^{(i)}|x_{test}^{(i)}, D_{train}^{(i)})$$

PFN with parameters $\theta^*$

Bayesian inference via the trained PFN, with the actual training data and a test point as input:

$q_\theta^*(y_{test}|x_{test}, D_{train}) \approx p(y_{test}|x_{test}, D_{train})$

Prior-Data Fitted Networks (PFNs)

Prior-fitting & Inference

Sample prior datasets $D^{(i)} \sim p(D)$

$D^{(1)} = D_{train}^{(1)} \cup \{(x_{test}^{(1)}, y_{test}^{(1)})\}$

$\vdots$

$D^{(K)} = D_{train}^{(K)} \cup \{(x_{test}^{(K)}, y_{test}^{(K)})\}$

Actual training dataset and test input

$(D_{train}, x_{test})$

Train the PFN by minimizing

$-\sum_{i=1}^{K} \log q_\theta(y_{test}^{(i)} | x_{test}^{(i)}, D^{(i)}_{train})$

PFN with parameters $\theta^*$

Bayesian inference via the trained PFN, with the actual training data and a test point as input:

$q_{\theta^*}(y_{test} | x_{test}, D_{train}) \approx p(y_{test} | x_{test}, D_{train})$

PFNs allow Bayesian inference for any prior we can sample from
TabPFN Prior: Structural Causal Models

Building a synthetic dataset for training

Initialize:

$y = x_2 x_{1.2} + x_{-0.2} x_{0.1} + x_{-1.1} x_{0.5} + \epsilon_1 x_{0.5} + \epsilon_2 x_{0.1}$

Sample noise per example:

Sample noise per example:

Build dataset:

$\{0.2\} \rightarrow \{((0.33, 0.35), 0.35), (0.22, 0.08, 0.05), (0.0, 1.1, 0.0)\}$
TabPFN Prior: Simplicity Principle

Prior likelihood

Graph Complexity
TabPFN Prior: Pior samples

Data for meta-learning is purely synthetic

```python
def create_random_scm(graph_nodes):
    for node, i in enumerate(g.nodes):
        g.nodes[node]["size"] = random.randint(...)
        g.nodes[node]["is_feature"] = False

nodes_per_layer = list(nx.topological_generations(g))
layers, node_ids_per_layer = [], []
adj = nx.adjacency_matrix(g)

for i, _ in enumerate(nodes_per_layer[:-1]):
    ...
```

Features

Colors indicate target classes
Bayesian Supervised Learning

Prior $p(t)$: SCM prior
With latents $t$: parameterizing SCMs, i.e. weights, graph structure, activation function, etc

Posterior: $p(t|D) = \frac{p(D|t)p(t)}{\int p(D|t)dt}$
Posterior predictive distribution: $p(y|x, D) = \int p(y|x, t)p(t|D)dt$
Simplifying AutoML for Real-time Training

AutoML pipeline

Image credit Biedenkapp et al. (https://www.autml.org/blog-2nd-automl-challenge/)
Simplifying AutoML for Real-time Training

TabPFN pipeline

\[ \{X_{\text{train}}, Y_{\text{train}}, X_{\text{test}}, \text{budget}\} \]

- Power Transform or Identity
- TabPFN Forward Pass
- Ensembling

\[ \hat{Y}_{\text{test}} \]
The TabPFN

Results on OpenML-CC18 suite subset with < 1000 examples
The TabPFN

Results on OpenML-CC18 suite subset with < 1000 examples

Additional evaluation on 149 validation datasets in our work!
Future Work

**Improving predictions:** Categorical/Missing data

**Larger applicability:** Generalizations to non-tabular data and regression tasks, architectural changes to remove quadratic memory scaling

**Application of TabPFN:** Instantaneous tabular predictions for small datasets in production: exploratory data analysis, novel feature engineering, active learning
Conclusions

TabPFN is fully learned: We do not specify the learning algorithm, but only prior assumptions from which the PFN learns a general prediction method.

Strong performance on small datasets: 1000 training samples, 100 features, 10 classes; stronger results for continuous data.

Training and prediction in < 1s: TabPFN can be parallelized on GPU using standard matrix operations.

We thank Eddie Bergman, Ravin Kohli, Matthias Feurer, Robert Bosch GmbH.