HyperFast: Instant Classification for Tabular Data

David Bonet[1,2], **Daniel Mas Montserrat**[1], **Xavier Giró-i-Nieto**[3*], **Alexander G. Ioannidis**[1]

[1]Stanford University, Stanford, CA, USA
[2]Universitat Politècnica de Catalunya, Barcelona, Spain
[3]Amazon, Barcelona, Spain
ioannidis@stanford.edu

Abstract

Training deep learning models and performing hyperparameter tuning can be computationally demanding and time-consuming. Meanwhile, traditional machine learning methods like gradient-boosting algorithms remain the preferred choice for most tabular data applications, while neural network alternatives require extensive hyperparameter tuning or work only in toy datasets under limited settings. In this paper, we introduce HyperFast, a meta-trained hypernetwork designed for instant classification of tabular data in a single forward pass. HyperFast generates a task-specific neural network tailored to an unseen dataset that can be directly used for classification inference, removing the need for training a model. We report extensive experiments with OpenML and genomic data, comparing HyperFast to competing tabular data neural networks, traditional ML methods, AutoML systems, and boosting machines. HyperFast shows highly competitive results, while being significantly faster. Additionally, our approach demonstrates robust adaptability across a variety of classification tasks with little to no fine-tuning, positioning HyperFast as a strong solution for numerous applications and rapid model deployment. HyperFast introduces a promising paradigm for fast classification, with the potential to substantially decrease the computational burden of deep learning. Our code, which offers a scikit-learn-like interface, along with the trained HyperFast model, can be found at https://github.com/AI-sandbox/HyperFast.

Introduction

Many different machine learning (ML) methods have been proposed for the task of supervised classification (Duda, Hart, and Stork 2000), following a traditional two-stage methodology. The initial stage involves the optimization of a model using the training portion of a dataset. Several tuning iterations are performed with the aim of finding the hyperparameter configuration of the model that yields the best performance on the specific task. In the second stage, the model with the chosen hyperparameter setup is used for evaluation and inference on the test set. Training and tuning models for classification tasks is time-consuming, and it often requires extensive data pre-processing, expertise in selecting hyperparameters that could fit the task at hand, and a validation process. Further, the computational and temporal costs of the traditional process can be prohibitive, particularly in real-time and large-scale applications, such as healthcare (Esteva et al. 2019), or applications where rapid model deployment is necessary (Deiana et al. 2022), including data streaming, where models need to be updated or re-trained frequently. In this work, we propose HyperFast, a novel method to solve classification tasks from multiple domains with a single forward pass of a hypernetwork. We substitute the slow training stage of a classification network with a fixed hypernetwork that has been pre-trained (meta-trained) to predict the weights of a smaller neural network (i.e. main network) that can instantly solve the classification task with state-of-the-art performance. Recently, TabPFN (Hollmann et al. 2023) has been proposed, introducing a pre-trained Transformer that is able to perform classification without training. However, it is constrained to ≤ 1000 training examples, 100 features and 10 classes, which limits its application to most real-world scenarios. In this study, we are particularly interested in ensuring adaptability to large dataset sizes, filling the gap present in the current landscape of pre-trained models for instant tabular data classification. Our model is designed to work with both large and small datasets, while also providing adaptability to different numbers of samples, features, and categories.

During the meta-training stage, the hypernetwork parameters are learnt and the parameters of a main model are inferred, that is, we are "learning to learn" from a wide variety of datasets (meta-training datasets) from different modalities for which HyperFast generates a smaller neural model that performs the actual classification. During the meta-testing or inference stage, HyperFast receives a "support set" of an unseen dataset (both features and labels), and predicts a set of weights for the main model, which classifies the test samples of the dataset. In this way, the process of adapting the model to a new dataset is accelerated, and the model that does the meta-learning is decoupled from the model that does the actual inference on the data. In other words, we train a high-capacity meta-model to encode task-specific characteristics in the weights of a smaller model. Model size is also decoupled, which means that a large meta-learner can be trained just once, while many lightweight models generated by the meta-learner can be used for deployment in different applications such as edge computing, IoT devices, and mo-

*Work done prior to joining Amazon.

bile devices, where computational resources are constrained, and fast inference is indispensable. These properties are also helpful to accelerate production, improve privacy aspects, or for federated learning (Yang et al. 2019). The meta-learner can instantly generate a model that is ready for deployment, but the generated weights might not be optimal. Thus, we also explore further improvements to quickly boost the performance before deployment and leverage all the power of the framework. For example, ensembles of multiple generated models can be used, or the generated weights can be used as an initial point for fine-tuning. More detail on many of the possibilities to improve model performance and obtain a stronger predictor can be found in the Appendix.

The hypernetwork is trained on a wide range of datasets with different data distributions, allowing it to learn relevant and general meta-features, such that during testing the hypernetwork can adapt and predict an accurate set of weights for new unseen datasets. We evaluate the performance of HyperFast across a set of 15 tabular datasets, including genomics datasets and a standardized suite of datasets from OpenML (Bischl et al. 2021). We also analyze the performance of HyperFast on larger time budgets by ensembling main networks generated with multiple forward passes and fine-tuning on inference. We compare our model to similar approaches and classical methods, both in terms of performance and time. Our method achieves competitive results compared to standard ML and AutoML algorithms tuned for up to one hour for each test dataset.

Related Work

Hypernetworks. Building from evolutionary algorithms, HyperNEAT (Stanley, D'Ambrosio, and Gauci 2009) evolves Compositional Pattern-Producing Networks (CPPNs) to augment the weight structure for a larger main network. Based on this idea, (Ha, Dai, and Le 2017) propose hypernetworks, where one neural network is used to generate weights for another neural network. The hypernetwork is trained end-to-end jointly with the main network to solve the task, producing weights in a deterministic way. (Krueger et al. 2018) and (Louizos and Welling 2017) propose variational approximations for weight generation using normalizing flows, (Deutsch 2018) use multilayer perceptrons (MLPs) and convolutions, and (Ratzlaff and Fuxin 2019) use generative adversarial networks (GANs). (Schürholt et al. 2022) explores unsupervised weight generation through model datasets. (Ashkenazi et al. 2022) use neural representations similar to NeRF (Mildenhall et al. 2020) to reconstruct weights of a pre-trained network leveraging knowledge distillation. The HyperTransformer (Zhmoginov, Sandler, and Vladymyrov 2022) is a few-shot learning hypernetwork based on the Transformer architecture that generates weights of a convolutional neural network (CNN). Unlike our method, the HyperTransformer is only designed for image classification and also requires training image and activation feature extractors. HyperFast presents a novel approach by introducing hypernetworks for instant tabular classification. HyperFast solves the classification task by taking a set of labeled datapoints (support set) and generating the weights of a neural model that can be directly used to classify new unseen datapoints. Previous work (Gidaris and Komodakis 2018; Qiao et al. 2018) considered generating weights for specific layers (e.g., the last classification layer), while training the rest of the feature extractor. Here, we go one step further and consider generating all the weights of the model that performs the classification in a single forward pass. Our hypernetwork design includes initial transformation modules, retrieval-based components, and different pooling operations in a unique architecture, offering feature permutation invariance and providing scalability and adaptability to new datasets while ensuring efficiency and speed.

Meta-learning. In the context of rapidly adapting to new tasks using limited data, meta-learning methods have emerged as powerful techniques. These approaches "learn to learn" by quickly integrating information at test time to make predictions for new, unseen tasks. A model $P_\theta(y|x, \mathcal{S})$ is learned for every new task, where y is the target, x is the test input, and $\mathcal{S} = \{X, Y\}$, is the support set. Metric-based learning methods such as Matching Networks (Vinyals et al. 2016) and Prototypical Networks (Snell, Swersky, and Zemel 2017) map a labelled support set $\mathcal{S}$ into an embedding space, where a distance is computed with the embedding of an unlabelled query sample to map it to its label. As in kernel-based methods, the model P_θ can be obtained through $P_\theta(y|x, \mathcal{S}) = \sum_{x_i, y_i \in \mathcal{S}} K_\theta(x, x_i) y_i$. Optimization-based methods such as Model-agnostic meta-learning (MAML) (Finn, Abbeel, and Levine 2017) learn an initial set of model parameters and perform an additional optimization through a function $f_{\theta(\mathcal{S})}$, where model weights θ are adjusted with one or more gradient updates given the support set of the task $\mathcal{S}$, i.e., $P_\theta(y|x, \mathcal{S}) = f_{\theta(\mathcal{S})}(x, \mathcal{S})$. Finally, model-based approaches such as Neural Processes (NPs) (Garnelo et al. 2018b,a) first process both support samples and query samples independently as in Deep Sets (Zaheer et al. 2017), and the predicted embeddings are aggregated with a permutation-invariant pooling operation, resulting in a dataset-level summary that is fed to a second stage network that predicts the output for the query sample. The overall model is defined by a function f and the process can be mathematically described as $P_\theta(y|x, \mathcal{S}) = f_\theta(x, \mathcal{S})$. Similarly, TabPFN (Hollmann et al. 2023) learns to learn Bayesian inference by using a Transformer network. In contrast, our method directly obtains the model weights θ in a single forward step through an independent network, i.e., the hypernetwork h, such that $P_\theta(y|x, \mathcal{S}) = f_{h(\mathcal{S})}(x)$.

Deep Learning for Tabular Data. Although deep learning (DL) models achieve state-of-the-art results in many domains (e.g., language, computer vision, audio), this is not the case for tabular data. Tree-based models such as XGBoost (Chen and Guestrin 2016), LightGBM (Ke et al. 2017) or CatBoost (Prokhorenkova et al. 2018) are still the preferred choice in some tabular data applications (Grinsztajn, Oyallon, and Varoquaux 2022; Shwartz-Ziv and Armon 2022). AutoML methods (He, Zhao, and Chu 2021; Feurer et al. 2020; Erickson et al. 2020) are also a popular alternative, automatically selecting the most appropriate ML algorithm and its hyperparameter configuration. However, it has been

shown that there is not a universal superior solution (Gorishniy et al. 2021; McElfresh et al. 2023), and many deep learning approaches for tabular data have been proposed (Kadra et al. 2021; Arik and Pfister 2021; Somepalli et al. 2021; Kossen et al. 2021; Gorishniy et al. 2021; Yan et al. 2023; Chen et al. 2022a; Katzir, Elidan, and El-Yaniv 2020; Popov, Morozov, and Babenko 2019; Hollmann et al. 2023; Chen et al. 2022b, 2023; Zhu et al. 2023; Zhang et al. 2023). (Kadra et al. 2021) introduced Regularization Cocktails, where different regularization techniques are applied to simple MLPs to boost performance. Recent work has explored using attention mechanisms to improve performance on tabular data. TabNet (Arik and Pfister 2021) adopts sequential attention on subsets of features, SAINT (Somepalli et al. 2021) applies attention over rows and columns in a BERT-style fashion and uses contrastive pre-training with data augmentation, NPT (Kossen et al. 2021) introduces attention between data points, ExcelFormer (Chen et al. 2023) models feature interaction and feature representation alternately, FT-Transformer (Gorishniy et al. 2021) adapts a Transformer with embeddings for categorical and numerical features, and T2G-Former (Yan et al. 2023) includes a graph estimator to guide tabular feature interaction. TabCaps explore capsule networks (Chen et al. 2022a), Net-DNF (Katzir, Elidan, and El-Yaniv 2020) disjunctive normal formulas, NODE (Popov, Morozov, and Babenko 2019) combines ensembles of differential oblivious decision trees with multi-layer hierarchical representations, DANets (Chen et al. 2022b) learn groups of correlative input features to generate higher-level features, and other works explore large language models (LLM) for tabular data pre-training (Zhu et al. 2023; Zhang et al. 2023). Nevertheless, most of the proposed DL models for tabular data require slow training and custom hyperparameter tuning for every new dataset. In contrast, we focus on off-the-shelf models that do not need extensive tuning for a new task. In this direction, TabPFN (Hollmann et al. 2023) pre-trains a Transformer on synthetic data given a prior to perform tabular data classification in a single forward pass with no hyperparameter tuning. However, TabPFN can only be applied to small tabular datasets, i.e., ≤ 1000 training examples, 100 features and 10 classes.

Background

Meta-Learning Problem Setting

In our meta-learning experiments, we train a model h (i.e., the hypernetwork) that is able to quickly adapt to new tasks given some observations, and generate the weights of a main model f that solves the task for unseen datapoints. We consider a set of classification tasks $\mathcal{T}$ where each task $t \in \mathcal{T}$ is associated with a *support set* $\mathcal{S}_t$ of examples that are sufficient to find the optimal model f that solves the task, a loss function $\mathcal{L}_t$, and a *query set* $\mathcal{Q}_t$ to define $\mathcal{L}_t$. The first phase is the *meta-training*, where in each step a different training task $t \in \mathcal{T}_{\text{meta-train}}$ is selected. We compile a set of meta-training datasets $\mathcal{D}_{\text{meta-train}}$, where each dataset $d \in \mathcal{D}_{\text{meta-train}}$ is composed of a training set d_{train} and a test set d_{test}, as in the common machine learning setup. In each meta-training step, a task $t \in \mathcal{T}_{\text{meta-train}}$ is sampled by first randomly choosing a meta-training dataset d. Then, $\mathcal{S}_t$ and $\mathcal{Q}_t$ are generated by sampling examples from d_{train} and d_{test}, respectively. Meta-validation is also performed intermittently through meta-training, where a separate set of meta-validation datasets $\mathcal{D}_{\text{meta-val}}$ is used to generate validation tasks $\mathcal{T}_{\text{meta-val}}$ to evaluate our algorithm and select the best performing model.

Once HyperFast is trained, an independent set of meta-testing datasets $\mathcal{D}_{\text{meta-test}}$ are used to create the evaluation tasks $\mathcal{T}_{\text{meta-test}}$ in which the selected model is evaluated. This approach allows us to extend the classical "n-way-k-shot" few-shot learning setting to handle multiple datasets with varying distributions and categories, testing the robustness and generalization of our model on new data.

As opposed to the training tasks, where each $t \in \mathcal{T}_{\text{meta-train}}$ is randomly generated at every meta-training step, $\mathcal{T}_{\text{meta-val}}$ and $\mathcal{T}_{\text{meta-test}}$ are sets of partially fixed tasks, as the query set $\mathcal{Q}_t$ always covers all d_{test} samples, in order to evaluate and compare with other methods equally, which also tests their performance on the entire test subset d_{test} of a dataset d.

HyperFast

The traditional training process can be seen as a function $f(X, Y) = \theta$, that receives training instances $X \in \mathbb{R}^{N \times D}$ and corresponding labels $Y \in \mathbb{R}^{N}$, and produces a set of trained weights θ of a model. In this work, we substitute the training process with HyperFast, a pre-trained meta-model based on a hypernetwork (Ha, Dai, and Le 2017) h, that takes as input a subset of the training data (i.e. support set $\mathcal{S}_t$) for a task $t \in \mathcal{T}_{\text{meta-train}}$ and predicts the weights of a main neural network f_θ for the given task t as $\theta^* = h(\mathcal{S}_t)$. The target model $f_{h(\mathcal{S}_t)}$ directly uses the predicted weights and makes predictions for test data points $x \in \mathcal{Q}_t$ in a single forward pass, such that $P_\theta(y|x, \mathcal{S}) = f_{h(\mathcal{S})}(x)$.

The meta-model is learnt by observing a set of tasks $t \in \mathcal{T}_{\text{meta-train}}$ and minimizing $\mathcal{L}_t(f_{h(\mathcal{S})}(x))$. In this section, we detail the design and architecture of h, named HyperFast in analogy to Hypernetworks (Ha, Dai, and Le 2017), and the ability to instantly adapt to new datasets in a single forward pass. Figure 1 illustrates the HyperFast framework and the main building blocks of the architecture. HyperFast is a multi-stage model with initial transformation layers that allows variable input size and permutation invariance, and a combination of linear layers and pooling operations that take both support samples and their associated labels to directly predict the weights θ_{main_l} (weight matrix and bias) of linear layers $l \in [1, L]$ of a target neural network. All trainable modules of HyperFast are learnt end-to-end by optimizing the classification loss of the main network evaluated on $\mathcal{Q}_t$.

The framework and HyperFast architecture proposed in this paper is a specific instance of a more general framework that could be easily extended to predict convolutional layers, batch normalization layers, recurrent layers, or deeper networks, for example. However, the architecture design depicted in Figure 1 selection has been driven by a global and simple approach to handle a wide range of multi-domain data, while seeking efficiency and speed.

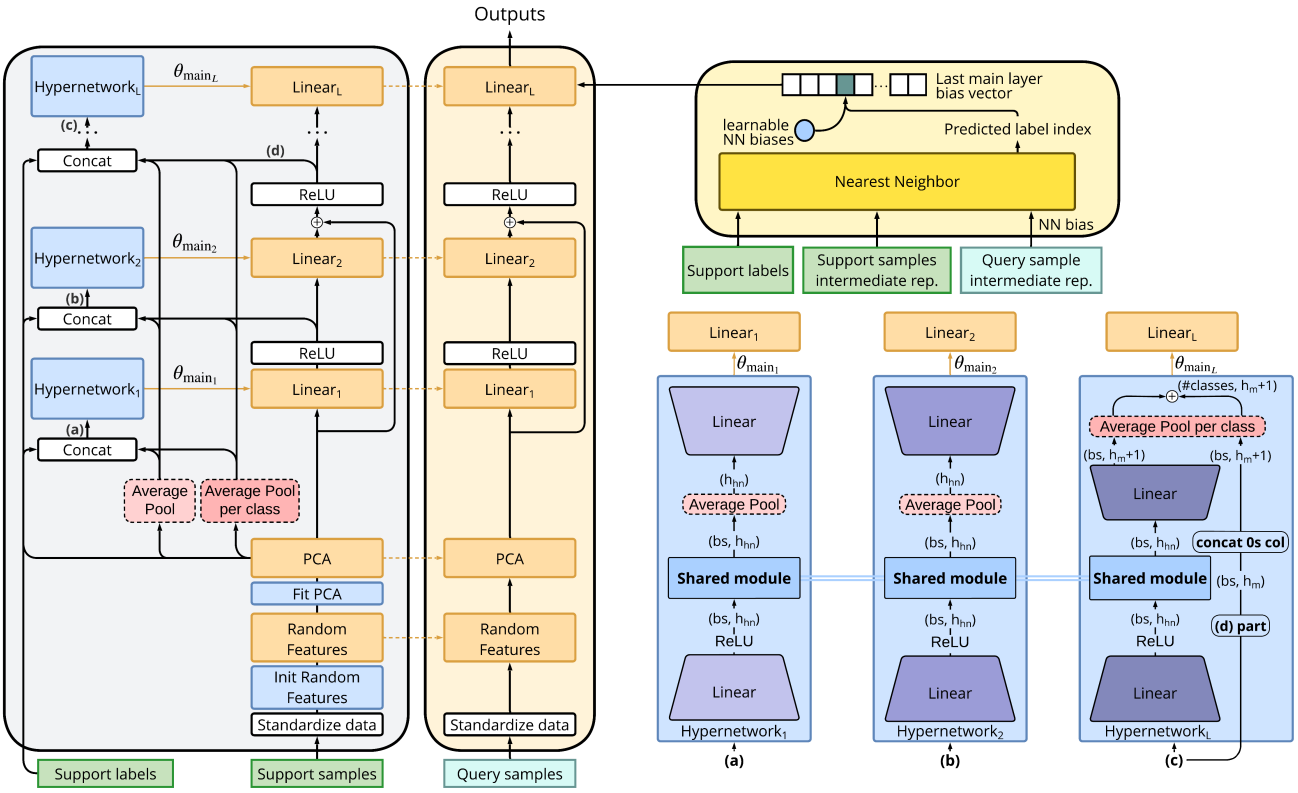

Figure 1: (left) HyperFast framework. (right) Architecture detail. Each hypernetwork module receives representations of the support set of *batch size* (bs) samples. The modules $l \in [1, L-1]$ compress the representations into a single embedding of *hypernetwork hidden size* (h_{hn}) to then generate the main network weights θ_{main_l}. Module L summarizes the representations per class with embeddings of *main hidden size* $(h_{\text{m}}) + 1$, directly obtaining the weights of the last classification layer θ_{main_L}.

Initial Transformation Layers

Properly dealing with datasets of differing dimensionality is a challenge, and one common solution is to apply padding (Hollmann et al. 2023) or to keep a subset of selected features up to a fixed size. We first perform a general data standardization stage by one-hot encoding categorical features, mean imputing missing numerical features, mode imputing missing categorical features, and feature-wise transforming to zero mean and unit variance. Then, HyperFast comprises initial layers that project datasets of different dimensionality to fixed-size and feature-permutation invariant representations. The kernel trick can be used to project data to a Reproducing Kernel Hilbert Space (RKHS) (Aronszajn 1950) when the number of dimensions tends to infinity. However, this would require computing all pairwise kernel distance in every step of the training process. Instead, we use random features (RF) (Rahimi and Recht 2007) to approximate a kernel with a fixed and finite number of dimensions. Random features are computed as $\phi(x) = a(Wx)$, where $a(\cdot)$ is a non-linearity, and W is a random projection matrix that follows a pre-defined distribution. The approximated kernel depends on the distribution of W and the selected non-linearity. In our case, we sample W from a Gaussian distribution and use the ReLU activation as non-linearity, approximating an arc-cosine kernel. We choose to approximate the arc-cosine kernel because it captures sparse, neural network-like feature representations in a non-parametric kernel setting (Cho and Saul 2009). In contrast, polynomial kernel's features are neither sparse nor non-negative, and radial basis function (RBF) kernels capture localized similarities. In each forward step, the random features projection matrix is re-initialized and sampled. The number of rows is adjusted to match the dimensionality of the input dataset, while the number of columns remains fixed, determining the output size.

The combination of random features with Principal Component Analysis (PCA) provides an efficient low-rank randomized approximation of Kernel PCA (Sriperumbudur and Sterge 2017; Lopez-Paz et al. 2014). We estimate the PCA parameters ψ using the support set and project the data onto a specified number of components. Subsequently, both ϕ and ψ are applied to the query samples to transform the data. This transformed data is then forwarded through the L generated linear layers of the main network.

Hypernetwork Modules

The process of generating the weights of the main network is done layer-by-layer, by multiple hypernetwork modules with both shared and layer-specific parameters, see Figure 1.

The hypernetwork module that generates the weights for the main network layer l receives as input the representations of the support samples in the previous stage, concatenated with the one-hot encoded support labels, the global average, and the class average of the low-rank Kernel PCA projection of the support set. Note that each sample is concatenated with the class average corresponding to its associated label.

For predicting θ_{main_1}, the representations are the low-rank Kernel PCA projection of the support samples. For $\theta_{\text{main}_l} \in [2, L]$, the hypernetwork module receives the intermediate representations of the support set in the main network at the output of layer $l-1$, after non-linearities and residual connections are applied. Figure 1 represents a specific multi-layer perceptron (MLP) architecture with ReLU activations and residual connections, which we use for our experiments. However, the HyperFast framework can be easily extended to generate weights for other main network architectures.

The hypernetwork modules that predict the layers $l \in [1, L-1]$ are composed of MLPs with shared middle layers that take the support set representations and labels, and output embeddings for each sample in the support set. Then, permutation-invariant weights are obtained averaging all support embeddings in a similar fashion to Deep Sets (Zaheer et al. 2017), to obtain a single dataset embedding that is passed to a final linear layer which outputs the final weights θ_{main_l} of l. θ_{main_l} is then reshaped as weight matrix and bias vector to forward the data through the main network.

Layer L is the classification layer of the main layer that outputs the logits for the final prediction. In this case, the intermediate representations after the layer $L-1$ and labels information are encoded through a MLP hypernetwork but the weights θ_{main_L} are not directly predicted from a global embedding. Instead, we leverage the fact that the rows of the classification layer weight matrix correspond to the different categories of the task. We perform an average pooling per class, and obtain the rows of the classification weight matrix (and bias) as the average of representations for each category. This also allows a much lightweight implementation, instead of directly predict the weight matrix. Additionally, we add a residual connection (He et al. 2016) from the previous layer representations for which we also perform a per class average, which helps in retaining category information from the input. Finally, we consider a module based on Nearest Neighbors to add learnable parameters (NN biases) to the classification layer bias vector of the main network. The label of a query sample is predicted with NN using the support set and the intermediate representations of the data across the main network, such that $P_\theta(y|x, \mathcal{S}) = f_{h(\mathcal{S})}(x, \mathcal{S})$. We consider the representations after the PCA projection, and after each linear layer. The NN biases are added to the position of the bias of the last main classification layer corresponding to the predicted label.

Once the main network is fully generated, query samples can be forwarded to make predictions. During meta-training, the predictions for the query samples $\mathcal{Q}_t$ of $t \in \mathcal{T}_{\text{meta-train}}$ are used to compute the cross-entropy loss $\mathcal{L}_t$ and learn the parameters of HyperFast end-to-end. In evaluation, all hypernetwork parameters are frozen and generate weights for a main network in a single forward pass.

Experiments

In this section, we compare HyperFast to many standard ML methods, AutoML systems and DL methods for tabular data on a wide variety of tabular classification tasks, listed in the Appendix. We do not perform any hyperparameter tuning to HyperFast, as it can be used as an off-the-shelf hypernetwork ready to generate networks to perform inference on new datasets. We then compare the performance and runtime of the generated model in a single forward pass, as well as the combination of multiple generated networks by increasing the ensemble size and fine-tuning on inference.

Baselines We compare HyperFast to standard ML methods, AutoML systems and state-of-the-art DL methods for tabular data. We first consider simple and fast ML methods as K-Nearest Neighbors (KNN) and Logistic Regression (Log. Reg.), and a MLP matching the architecture of the target network. We also evaluate against tree-based boosting methods: XGBoost (Chen and Guestrin 2016), LightGBM (Ke et al. 2017), and CatBoost (Prokhorenkova et al. 2018). As AutoML methods we incorporate Auto-Sklearn 2.0 (ASKL 2.0) (Feurer et al. 2020), which uses Bayesian Optimization to efficiently discover a top-performing ML model or a combination of models by ensembling, and AutoGluon (Erickson et al. 2020), which uses a selection of models such as neural networks, KNN, and tree-based models, combining them into a stacked ensemble. Finally, we include popular tabular DL methods: SAINT (Somepalli et al. 2021), TabPFN (Hollmann et al. 2023), NODE (Popov, Morozov, and Babenko 2019), FT-Transformer (Gorishniy et al. 2021), and T2G-Former (Yan et al. 2023). All standard ML models, gradient boosting methods and SAINT are evaluated using 5-fold cross validation for hyperparameter adjustment. Hyperparameter configurations are drawn from search spaces (detailed in the Appendix) unitl 10 000 configurations are explored, a specified time budget is reached, or more than 32 GB of memory are required if GPU training is possible for the model. Then, the model is trained on the full training set with the best configuration between the hyperparameter search result and the default. For the AutoML methods, the time budget is given. Finally, both TabPFN and our HyperFast are pre-trained models with no hyperparameter tuning requirements, but with ensembling capabilities. Thus, we perform ensembling for each method until a given number of members are used (detailed in the Appendix) or until 32 GB of GPU memory are overloaded.

Data We collect a wide variety of datasets from different modalities. We use the 70 tabular datasets from the OpenML-CC18 suite (Bischl et al. 2021) which, to the best of our knowledge, is the *largest* and most used standardized tabular dataset benchmark, composed of standard classification datasets (e.g., Breast Cancer, Bank Marketing). The collection of OpenML datasets is randomly shuffled and divided into meta-training, meta-validation and meta-testing sets, with a 75%-10%-15% split, respectively. We also include tabular genomics datasets sourced from distinct biobanks. Specifically, we utilize genome sequences of dogs (Bartusiak et al. 2022) for dog clade (group of breeds) prediction in meta-training, European (British) humans from

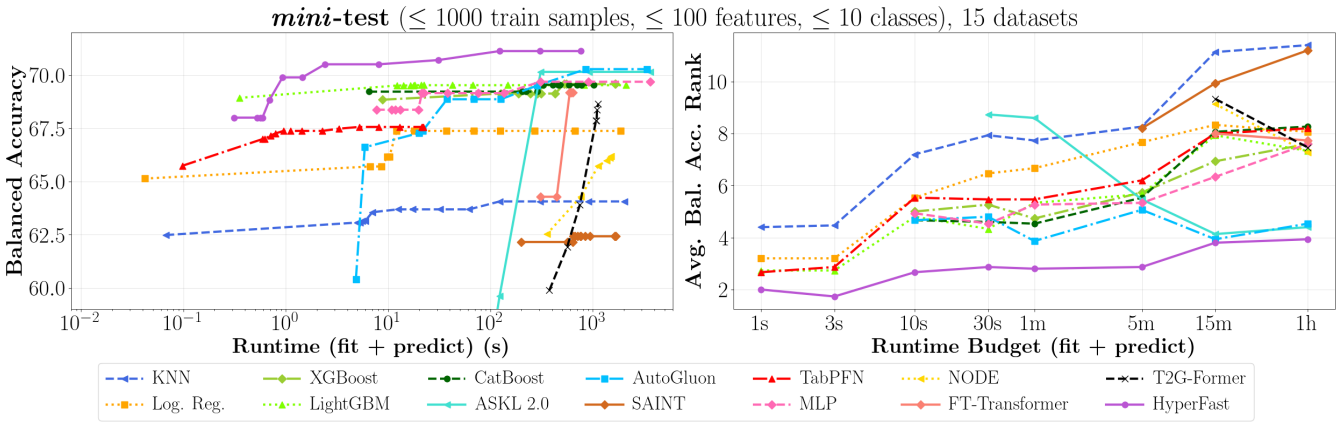

Figure 2: Runtime (fit + predict) vs. performance and average rank for given runtime budgets on the *mini*-test (small-sized version of the 15 meta-test datasets with $\leq$ 1000 training examples, $\leq$ 100 features and $\leq$ 10 classes restrictions).

the UK Biobank (UKB) (Sudlow et al. 2015) for phenotype prediction in meta-validation, and HapMap3 (Consortium et al. 2010) for subpopulation prediction in the meta-test. This strict separation ensures we meta-learn and evaluate on substantially different distributions and tasks. More details on the processing of these datasets are provided in the Appendix. The simple ML methods, implemented with scikit-learn (Pedregosa et al. 2011), and the MLP, receive the numerical features standardized with zero mean and unit variance, and the categorical features are one-hot encoded. For the missing values, we perform mean imputation for numerical features and mode imputation for categorical features, as it was the configuration that yielded the best performance. We also perform imputation of missing values for SAINT, NODE, and FT-Transformer. Boosting methods, AutoML systems, and TabPFN receive the raw data and the indices of categorical features when needed, as their documentation states that they pre-process inputs internally.

Apart from the large-sized original test datasets, we create a secondary small-sized tabular data version (*mini*-test) of the meta-testing datasets to compare to TabPFN, as it is only able to handle $\leq$ 1000 training examples, $\leq$ 100 features and $\leq$ 10 classes. We randomly select a subset of $\leq$ 1000 training samples and $\leq$ 100 features for each dataset. We do not perform any downsizing in terms of number of classes as the highest number of classes appearing in the meta-testing set is 10. However, HyperFast is pre-trained with datasets with higher number of classes and can be used in inference for datasets with $>$10 classes. Only models that can complete the runs for all 15 datasets in less than 48 hours in their default configuration are included in our large-scale experiments. The experiments, which are conducted for all models and both size versions of the 15 meta-testing datasets, considering all time budgets shown in Figure 2, require a total of 2 months to complete. Therefore, we show additional results with 10 repetitions of the experiments for a specific time budget of 5 minutes for each dataset in the Appendix.

Experimental setup We perform supervised classification with HyperFast and all other baselines on the *mini*-test, a small-sized version of the meta-test datasets $\mathcal{D}_{\text{meta-test}}$, and in the original large-scale datasets. To train HyperFast, we use a different set of meta-training datasets, $\mathcal{D}_{\text{meta-train}}$, and select the model with the best average performance on the meta-validation datasets, $\mathcal{D}_{\text{meta-val}}$. We report balanced accuracy, which is the mean of sensitivity and specificity. Balanced accuracy provides a more objective and robust evaluation across classes, especially in the context of imbalanced datasets. In contrast, standard accuracy can be misleading, often masking poor performance in minority classes. We evaluate the models on a time budget (including tuning, training, and prediction) to correctly assess computational complexity and performance. The average rank is also reported.

In order to transform the data to a fixed-size and permutation invariant representation, we apply Random Features and Principal Component Analysis to both support samples and query samples. We set a Random Features projection to 32 768 (2^{15}) features, sampled from a normal distribution following the He initialization (He et al. 2015), followed by a ReLU activation. Note that the random linear layer that computes the random features is not trained, and re-initialized in each HyperFast forward step. Then, we keep the principal components (PCs) associated to the 784 largest eigenvalues, as many of the datasets considered have this dimensionality, and it is a more than sufficient number of dimensions to retain the important information of higher dimensional datasets while preserving efficiency. After the PCA projection, most genomics datasets resemble a similar histogram distribution (i.e., zero mean, small deviation and no outliers). However, it is not the case for some OpenML datasets, which are also centered around zero but present many outliers. Thus, we clip the data after PCA at 4σ.

The hypernetwork modules receive a concatenation of intermediate representations of the support samples, and the support labels. Given that each dataset features a different number of categories and linear layers require a fixed input size, we one-hot encode the labels and apply zero padding up to the maximum number of categories considered in the

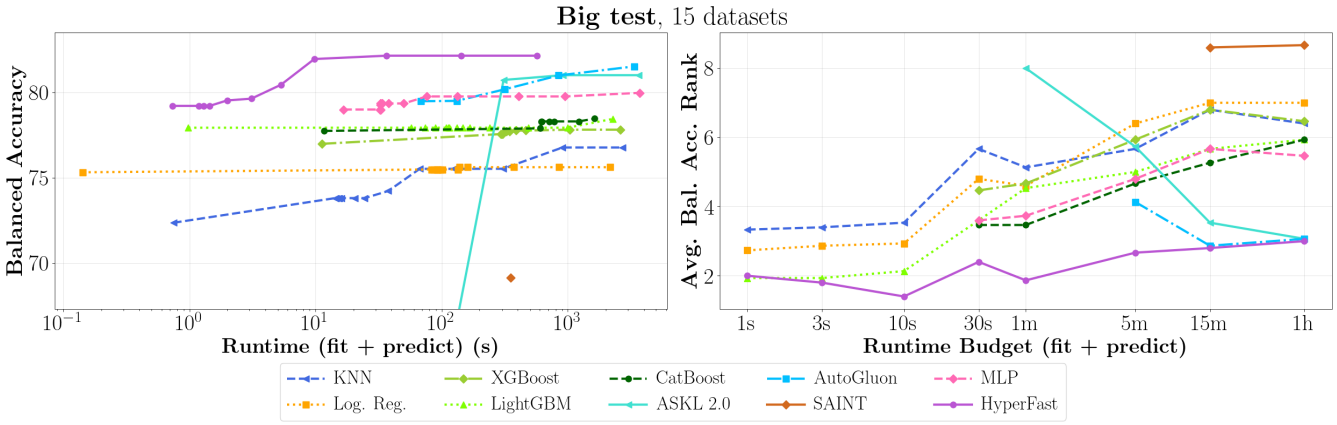

Figure 3: Runtime (fit + predict) vs. performance and average rank for given runtime budgets on the big test: 15 large/medium-sized meta-datasets.

experiments. It is important to note that the number of categories that HyperFast can handle is easily extendable by expanding the input size of HyperFast and zero padding the remaining input dimensions. Such modifications have a negligible impact on efficiency or memory requirements, up to a reasonable number of categories. As a shared module we use 2 feed-forward layers with a hidden size of 1024 and ReLU activations. For the main network, we consider a 3-layer MLP with a residual connection (He et al. 2016), and a main network hidden size equal to the number of PCs (784 dimensions). We select this simple architecture to be able to obtain competitive performance on a wide variety of datasets with a single trained model while preserving efficiency. Other alternatives include predicting weights for CNN layers for only-image datasets, or recurrent layers, for sequential data. Instead, we create a general and simple meta-learning framework to perform fast lightweight inference. In the NN bias module, we randomly select a subset of a maximum of 2048 support samples, since computing all pairwise distances for a large number of datapoints causes high inefficiency and GPU memory overload. A maximum batch size of 2048 samples is used for training, and we make sure to have a sufficient number of samples per category in every case.

In evaluation, we show prediction results with HyperFast in a single forward pass, as well as predictions by ensembling main networks generated in multiple forward passes. We also experiment with performing gradient steps with the training data of the meta-testing datasets on the generated main networks, including the random features projection matrix, PCA parameters and linear layers.

Results on small-sized datasets. We first compare HyperFast to the other methods on a small-sized setting with datasets having ≤ 1000 training samples, ≤ 100 features and ≤ 10 classes, in order to compare to TabPFN. As shown in Figure 2, HyperFast delivers superior results in both performance and runtime, with better prediction capabilities up to 3 orders of magnitude faster than competing methods. Simple ML methods such as KNN and Log. Reg. also deliver instant predictions, but do not achieve remarkable performance. Interestingly, an MLP (with an architecture identical to the network generated by HyperFast, including the including the initial transformation layers) performs on par with XGBoost. However, HyperFast surpasses gradient-boosting techniques in both runtime and performance. LightGBM stands out as the only boosting machine that achieves a higher balanced accuracy in a similar runtime to a single forward pass by HyperFast. Yet, an ensemble of networks generated by HyperFast outperforms all fine-tuned boosting machines in under 3 seconds. TabPFN is noted for its rapid predictions and outperforms NODE and SAINT. But on average, it falls behind gradient boosting machines and neural models, including HyperFast. FT-Transformer, T2G-Former, NODE, and SAINT are DL tabular models with very time-consuming training, and FT-Transformer obtains the highest performance among them, similar to that of gradient-boosting machines. AutoML systems are superior to the other baselines when given higher runtime budgets. However, HyperFast still outperforms both AutoGluon and ASKL 2.0 for runtimes up to 1h, obtaining the lowest rank throughout all the budgets in the mini test.

Results on medium/large-scale datasets. Figure 3 benchmarks the algorithms on large real-world datasets. We observe that HyperFast is able to obtain predictions in less than a second, and achieves the overall best performance in a wide range of runtime budgets, ranging from 1 second to 5 minutes. For more extended budgets, up to 1h per dataset, HyperFast's performance is on par with other AutoML systems. Specifically, HyperFast, ASKL 2.0, and AutoGluon all achieve an average rank of approximately 3.0. In comparison, gradient-boosting machines plateau at a balanced accuracy of 78.4% and rank above 5.9, being outperformed by the MLP. SAINT obtains the lowest performance, using the hyperparameter configuration that the authors implement for the biggest datasets they consider in their benchmark. No hyperparameter optimization is performed for SAINT in the big test since larger architectures do not fit in GPU memory for the larger datasets. Additional

Variation	Bal. acc. (%)	Bal. acc. diff.	Fit time (s)	Pred. time (s)	HF size	Model size
Base model (784 PCs)	81.496	-	0.600	0.125	1.27 B	52.65 M
No RF	75.387	-6.108	0.126	0.114	1.27 B	1.85 M
No RF-PCA	73.704	-7.792	0.029	0.109	1.26 B	1.23 M
First 512 PCs only	81.347	-0.149	0.625	0.125	547 M	43.03 M
First 256 PCs only	81.235	-0.261	0.640	0.042	140 M	34.25 M
d_{RF}=16 384 (2^{14})	81.059	-0.436	0.510	0.116	1.27 B	26.95 M
No concat PCA to hypern. modules	80.727	-0.769	0.583	0.125	1.26 B	52.65 M
1 linear layer in shared module	80.790	-0.706	0.637	0.125	1.27 B	52.65 M
No residual conn. in hypern.$_L$	77.835	-3.660	0.620	0.125	1.27 B	52.65 M
No residual con. in main model	80.318	-1.178	0.633	0.125	1.27 B	52.65 M
No NN bias using PCA features	81.305	-0.191	0.625	0.125	1.27 B	52.65 M
No NN bias using interm. act.	80.703	-0.793	0.628	0.125	1.27 B	52.65 M
No NN biases	79.714	-1.781	0.628	0.125	1.27 B	52.65 M
Random init. linear layers main	72.229	-9.267	0.437	0.125	-	52.65 M

Table 1: Ablation studies on HyperFast performing a single forward pass. Time results are shown for a single GPU. *HF size* denotes the number of trainable parameters of HyperFast, i.e., the meta-model, while *Model size* denotes the size of the generated model.

experiments with very high-dimensional genomic datasets can be found in the appendix.

Ablation Experiments In Table 1, we present ablation studies for the HyperFast framework, exploring variations affecting both hypernetwork modules and the generated model. First, we consider removing the RF and both RF and PCA modules, obtaining a fixed-sized input by keeping the first 784 features or applying zero padding. The weight generation time is reduced from 0.6s to 0.12s and 0.03s, since the main time bottleneck is the RF matrix multiplication and SVD to obtain the PCs. Also, the main model size is greatly reduced as RFs account for most parameters, but the drop in performance is one of the most significant. This is because RF and PCA not only allow transforming any dataset to a fixed number of features, but also homogenize the input data to HyperFast and the generated network across datasets. For example, the first feature post RF-PCA holds the most variance, with subsequent features capturing the maximum variance that is orthogonal to the previous dimensions, with minimal information loss. Also, histogram distributions are similar across datasets with zero mean. These properties help in learning important meta-features across different dataset distributions. If we scale down RF-PCA by reducing the number of PCs used and the RF dimensionality, we observe that model size is significantly reduced while the drop in performance is not critical, which shows that most dataset relevant information is preserved, even using 512 or 256 PCs. These observations can help create even more efficient HyperFast desings in the future. In addition, PCA representations concatenated to hypernetwork inputs retain key information without a major parameter increase. We also observe that reducing the shared hypernetwork module from 2 to 1 layer degrades performance, and residual connections in both the hypernetwork and main model are key to retain post-PCA and per class information, while not increasing model size and runtime. We also analyze the retrieval-based component of HyperFast. We observe that NN biases in the last classification layer improve predictions while maintaining model size, especially using the intermediate activations of the main network as features. Finally, if we replace the weights produced by HyperFast by random weights, and base the prediction solely on the Nearest Neighbor-based component, we observe the biggest drop in performance.

Conclusion

We present HyperFast, a meta-trained hypernetwork designed to perform rapid classification of tabular data by encoding task information in the prediction of the weights of a target network in a single forward pass. Our experiments show that HyperFast consistently improves performance over traditional ML methods and tabular-specific DL architectures in a matter of seconds. Remarkably, it is able to replace the traditional training of a neural network, and achieves competitive results with state-of-the-art AutoML frameworks trained for 1h. HyperFast eliminates the necessity for time-consuming hyperparameter tuning, making it a highly accessible, off-the-shelf model that can be specially useful for fast classification tasks. We also explore how we can leverage all training data by creating ensembles of generated networks and fine-tuning them on inference, significantly boosting performance at almost no additional computational cost. Future work should consider expanding this framework to a general architecture or multi-hypernetwork setting that is able to handle regression tasks, multi-domain and high-dimensional non-tabular settings such as audio streams, 3D, and video.

Acknowledgments

This work was partially supported by a grant from the Stanford Institute for Human-Centered Artificial Intelligence (HAI) and by NIH under award R01HG010140. This research has been conducted using the UK Biobank Resource under Application Number 24983.