AutoForecast: Automatic Time-Series Forecasting Model Selection

Mustafa Abdallah
mabdall@iu.edu
Indiana University-Purdue University
Indianapolis
Indianapolis, IN, USA

Ryan Rossi
ryrossi@adobe.com
Adobe Systems
San Jose, CA, USA

Kanak Mahadik
mahadik@adobe.com
Adobe Systems
San Jose, CA, USA

Sungchul Kim
sukim@adobe.com
Adobe Systems
San Jose, CA, USA

Handong Zhao
hazhao@adobe.com
Adobe Systems
San Jose, CA, USA

Saurabh Bagchi
sbagchi@purdue.edu
Purdue University
West Lafayette, IN, USA

ABSTRACT

In this work, we develop techniques for fast automatic selection of the best forecasting model for a new unseen time-series dataset, without having to first train (or evaluate) all the models on the new time-series data to select the best one. In particular, we develop a forecasting meta-learning approach called AutoForecast that allows for the quick inference of the best time-series forecasting model for an unseen dataset. Our approach learns both forecasting models performances over time horizon of same dataset and task similarity across different datasets. The experiments demonstrate the effectiveness of the approach over state-of-the-art (SOTA) single and ensemble methods and several SOTA meta-learners (adapted to our problem) in terms of selecting better forecasting models (i.e., 2X gain) for unseen tasks for univariate and multivariate testbeds.

CCS CONCEPTS

• **Computing methodologies** → **Machine learning**; **Feature selection**;

KEYWORDS

Time-series forecasting, Model selection, AutoML, Meta-learning

ACM Reference Format:
Mustafa Abdallah, Ryan Rossi, Kanak Mahadik, Sungchul Kim, Handong Zhao, and Saurabh Bagchi. 2022. AutoForecast: Automatic Time-Series Forecasting Model Selection. In *Proceedings of the 31st ACM International Conference on Information and Knowledge Management (CIKM '22), October 17–21, 2022, Atlanta, GA, USA.* ACM, New York, NY, USA, 10 pages. https://doi.org/10.1145/3511808.3557241

1 INTRODUCTION

Accurate time-series forecasting at scale is critical for a wide range of industrial domains such as cloud computing [37], supply chain [1], energy [11], and finance [33]. Most of the current time-series forecasting solutions are built by experts and require significant manual effort in model construction, feature engineering, and hyper-parameter tuning [6]. Hence, they do not scale to generate high-quality forecasts for a wide variety of applications. Moreover, there is no learning scheme that is uniformly better than all other learning schemes for all problem instances. For example, from our experiments (see Figure 2), we find empirically that no single forecasting model triumphs in more than 0.7% of the datasets in our two training testbeds comprising 625 time series (details in Section 6), *i.e.,* there is no unique single model that works well on all datasets. A naïve approach would be, given a new dataset, we evaluate the performance of thousands of available models on the dataset to select the best forecasting model for the problem at hand. However, this approach is practically infeasible due to the untenable time burden for every new problem.

In this work, we formulate the problem of automatic and fast selection of the best time-series forecasting model as a meta-learning problem. Our solution avoids the infeasible burden of first training each of the models and then evaluating each one to select the best model for a new unseen time-series dataset, or even a new time window within a non-stationary dataset. A practically important desideratum for any solution to this problem is that once the meta-learner $\mathcal{L}$ is trained in an offline manner using a large corpus of time-series data, then we can use it to *quickly* infer the best forecasting model. The quick inference requirement of this new problem, makes it challenging to solve, yet practically important. Our meta-learner $\mathcal{L}$ is trained on the models' performances on historical datasets and the time-series meta-features of these datasets.

We emphasize that our *time-series forecasting model selection meta-learning problem* has several unique characteristics and challenges compared to previous related meta-learning problems, e.g., [16, 40, 54]. *First*, existing time-series forecasting models have different designs and different assumptions around the characteristics of time-series (e.g., probabilistic, seasonal, traditional, etc.). Therefore, different models perform differently depending on the characteristics that each dataset exhibits. Thus, capturing the similarity among different datasets needs careful selection of representative time-series meta-features. *Second*, the new meta-learning approach should capture the temporal variations of the models' performances over different time windows of the dataset. This is borne out of our observation that

CIKM '22, October 17–21, 2022, Atlanta, GA, USA
© 2022 Association for Computing Machinery.
ACM ISBN 978-1-4503-9236-5/22/10…$15.00
https://doi.org/10.1145/3511808.3557241

the best time-series forecasting model for time window w_t is not necessarily the best model for a subsequent time window w_{t+k} (see Figure 3 in Section5.3). *Third*, the number of available time-series forecasting models is large (in thousands) and thus training each forecasting model and then evaluating the suitability of each in inference leads to an unacceptable time burden for most real-world scenarios. These challenges motivate the need for our approach.

Our solution. To solve the problem of *automatic* time-series forecasting model selection, we propose a temporal meta-learning approach, called AutoForecast that selects the best time-series forecasting model without a heavy evaluation burden. The schematic of AutoForecast with the main components and their interactions is shown in Figure 1. There are two key intuitions behind our approach. *First*, we learn the similarity across datasets through meta-features that capture key characteristics of the datasets and then developing our “general meta-learner” that learns to predict the performance of a model for a time window within a dataset. *Second*, we learn a model’s performance evolution over successive time windows for the same dataset via our “temporal meta-learner”. We train our meta-learner using a large model space which has over 320 forecasting models (Section 5.1). We also generate more than 800 meta-features that represent five different types of meta-features (simple, statistical, information theoretic, spectral-based, and landmarker), which reflect various characteristics of the time-series datasets (Section 3). We also consider diverse datasets so our meta-learning model becomes generalizable to new time series datasets (Section 5.1). To stimulate reproducible research on this topic, we publicly release the corpus of datasets, along with their meta-features and the performances across hundreds of models, plus our source codes for training and evaluation[1]. Given a new (unseen) dataset, AutoForecast automatically determines, using the meta-features and the meta-learners, the best forecasting model among a large space of models, without the need to train and evaluate any of the different forecasting models on this new dataset.

The experiments demonstrate the effectiveness of our proposed approach where we validate our meta-learning approach on both univariate and multivariate testbeds. In particular, we show the superiority of our approach over the state-of-the-art (SOTA) time series forecasting models [27, 30, 41, 48, 52] (including DeepAR [41], DeepFactors [52], and Prophet [48]) and different meta-learning approaches [20, 32, 56] (including simple and optimization-based meta-learners). Across all datasets, AutoForecast is at least 2× better in selecting the best forecasting model, compared to the closest baseline. Moreover, AutoForecast yields a significant reduction in inference time over the naïve approach — AutoForecast has a 42× median inference time reduction averaged across all datasets.

Summary of Main Contributions. The key contributions of this work are as follows:

(1) **Problem Formulation**: We formulate time-series forecasting model selection in a novel light, as a meta-learning problem.

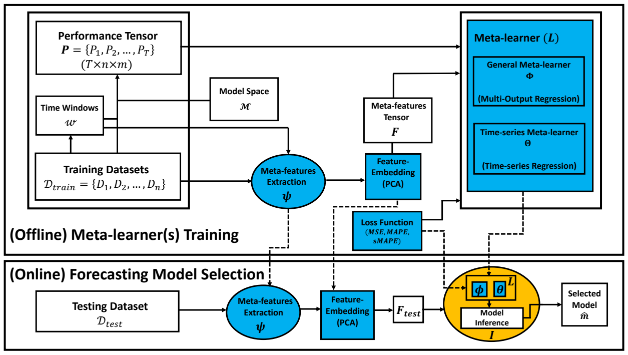

Figure 1: An overview of AutoForecast; components that transfer from offline to online (model selection) phase are shown in blue. Given the two main inputs, the performance tensor P and the meta-features tensor F, the meta-learner $\mathcal{L}$ learns two main components: general meta-learner (Φ) and time-series meta-learner (Θ). These are then used online to quickly predict the performance of available models on the new test dataset and pick the expected best model.

(2) **Temporal Learning of Performances**: We propose a meta-learner that learns the models’ performances evolution over time windows of the datasets. Our meta-learner has two sub-learners — the time-series meta-learner and the general meta-learner that are designed for different data types with different time dependencies.

(3) **Specialized Meta-features for Time-series Forecasting**: We design novel time-series landmarker meta-features to capture the unique characteristics of a time-series dataset toward effectively capturing task similarity.

(4) **Efficiency and Effectiveness**: Given a new time-series dataset, AutoForecast selects the best performing forecasting algorithm and its associated hyperparameters without requiring any model evaluations, incurring negligible run-time overhead. Through extensive experiments on our benchmark testbeds, we show that selecting a model by AutoForecast outperforms SOTA meta-learners and popular forecasting models.

(5) **Benchmark Data:** We release our meta-learning database corpus (348 datasets), performances of the 322 forecasting models, meta-features, and source codes for the community to access it for forecasting model selection and to build on it with new datasets and models. As part of this, we are unveiling new traces of Adobe’s computing cluster usage for production workloads.

2 RELATED WORK

Meta-learning in Time-series Forecasting: There are few works that considered meta-learning for time-series analysis [19, 24, 35, 39, 51]. Among these, a few works considered simple ranking-based [24, 29] and rule-based [4, 51] meta-learners. The works [19, 39] applied a neural network time-series forecasting model trained on a source (energy) dataset and fine-tuned it on the target (energy) dataset. However, these works did not consider using meta-learning for the general problem of forecasting model selection that we consider in our current work. There also exist few works that have explored

[1] The URL for our database and source codes is:
https://drive.google.com/drive/folders/1K1w1Ida5Cr15b5Fhidax-i-fNpWZjvet.
The Adobe traces are available from:
https://github.com/adobe-research/AutoForecast_ResourceUsageData.

model selection problem using ensemble learning [15, 46, 49]. However, their problem domain of ensemble learning is different from our problem of model selection. This is due to the fact that ensemble learning constitutes building multiple models for the same task and does not in itself involve learning from prior experience on other tasks. In contrast to those works, AutoForecast can select among any (heterogeneous) set of methods. Finally, there is a line of work that considered empirical analysis for performance estimation [5, 9, 43] and model selection [10, 47] in time-series forecasting. However, these works have several distinctions from our work: (i) the need for evaluating *all* forecasting models in inference and (ii) providing an analysis of the ranking ability of performance estimators without having a meta-learner. In contrast, our meta-learner learns how to automatically select the best model and can capture the dependence within the same dataset.

Few-shot Learning & Transfer Learning: Few-shot learning has been recently leveraged for automating machine learning pipeline [33, 38, 45, 57]. In particular, the works [38, 45, 57] investigated different problems outside the domain of time-series forecasting. The work [33] applied meta-learning for zero-shot univariate time series forecasting. However, that work has the limitations of focusing on solving the cold start problem (learning model parameter initialization that generalizes better to similar tasks) which is different from our forecasting model selection problem, considering different models from the same N-BEATS architecture [34], and tackling only univariate time-series datasets. We emphasize that our framework can use N-BEATS as one forecasting algorithm in our model space. Finally, there exist few works that applied transfer learning for time series classification (TSC) [2, 13, 31, 53]. These works however have two distinctions from our work. First, they transfer the learned network's weights to another network that is also trained on a target dataset. Second, the TSC problem is different from our forecasting problem.

Hyperparameter Optimization: Automated hyperparameter optimization (HPO) has received a surge of attention in the machine learning domain in the last decade [14]. In particular, decision-theoretic [7], bandit-based [26], meta-heuristic [28] and Bayesian optimization (BO) techniques [44] are various SOTA approaches for doing HPO. We emphasize that all of these approaches rely on multiple model evaluations (i.e., performance queries) which are computationally expensive and typically start from scratch for every new dataset and hence lead to huge overhead when applied to the time-series forecasting model selection problem.

3 PROBLEM FORMULATION

We address the problem of model selection for time-series forecasting via the meta-learning approach.

Meta-learning Components: Our proposed meta-learner AutoForecast depends on:

- A collection of historical time-series forecasting datasets $\mathcal{D}_{train} = \{\boldsymbol{D}_1, \boldsymbol{D}_2, \cdots, \boldsymbol{D}_n\}$, namely, a training database, where n is the number of the historical datasets in $\mathcal{D}_{train}$. Note that $\boldsymbol{D}_i \in \mathbb{R}^{n_i \times v_i}$, where n_i is the number of observations of the dataset $\boldsymbol{D}_i$ and v_i is the number of variables in $\boldsymbol{D}_i$.
- The forecasting models that define the model space (set), denoted as $\mathcal{M} = \{M_1, M_2, \cdots, M_m\}$, where m is the size of the model space.
- For each dataset $\boldsymbol{D}_i \in \mathcal{D}_{train}$, we sample random T windows from $\boldsymbol{D}_i$, where each sample window w_t from dataset $\boldsymbol{D}_i$ has length $|w_t|$ (smaller than the dataset length).

Window Notation: Time window represents a sequence of time observations in the time series. In particular, w_t denotes the t-th time window, and $|w_t|$ is the length of that time window (e.g., $|w_{10}| = 16$ means that the 10th time window of the dataset has a length of 16 observations).

Model Design: Now, we explain the model space design in our solution (AutoForecast). For our forecasting model selection problem, we define our model as follows.

Definition 1. *A model $M_i \in \mathcal{M}$ is given by the tuple $M_i = (a_i, \mathbf{h}_i, g_i(\cdot))$, where a_i is the forecasting algorithm, $\mathbf{h}_i$ is the hyperparameter vector for the forecasting algorithm a_i, and $g_i(\cdot) : \mathbb{R}^{n_i \times v_i} \to \mathbb{R}^{n_i \times v_i}$ is the time-series data representation.*

We emphasize that $\mathbf{h}_i$ consists of hyper-parameters of the forecasting algorithm (e.g., number of RNN layers in DeepAR [41]) and that $g_i(\cdot)$ represents optional transformations of the original time-series data (e.g., exponential smoothing [22]; see Table 1).

Performance Tensor: Now we introduce the performance tensor:

Definition 2. *Given a training database $\mathcal{D}_{train}$ and a model space $\mathcal{M}$, we define the performance tensor $\mathbf{P} \in \mathbb{R}^{T \times n \times m}$ as*

$$\mathbf{P} = \{\boldsymbol{P}_1, \boldsymbol{P}_2, \cdots, \boldsymbol{P}_T\},$$

where $\boldsymbol{P}_k = (p_k^{i,j}) \in \mathbb{R}^{n \times m}$ and the element $p_k^{i,j} = M_j(w_k(\boldsymbol{D}_i))$ denotes the j^{th} model M_j's performance on the time window w_k of the i^{th} training dataset $\boldsymbol{D}_i$. We denote $\boldsymbol{p}_k^i = \begin{bmatrix} p_k^{i,1} & \cdots & p_k^{i,m} \end{bmatrix}$ as the performance vector of all models in $\mathcal{M}$ on time window w_k of $\boldsymbol{D}_i$.

Note that we denote the performance of a model on a time window by the forecasting error (e.g., MSE) of that model on that window. The performance tensor represents the prior experience that the meta-learner will leverage to perform efficiently on the new unseen task (time-series). This motivates us to define our problem.

Definition 3. ***Time-series forecasting model selection problem**. Given a new input task (dataset) $\boldsymbol{D}_{test}$ (i.e., unseen time-series forecasting task), the time-series forecasting model selection problem is then stated as follows: for each time window w_t in $\boldsymbol{D}_{test}$, select the best model $\hat{M}_t \in \mathcal{M}$ to employ on that window. Formally, such selection problem is given by*

$$\hat{M}_t \in \arg\max_{M_j \in \mathcal{M}} M_j(w_t(\boldsymbol{D}_{test})), \ t \in \{1, 2, \ldots, T\}. \tag{1}$$

Our problem can be described as follows. We are given a new time series dataset and we have to select the best model to perform forecasting on it. We have a prior bag of models from which we have to select the best candidate for the forecasting task. An additional subtlety is that within the new time series, we may have to select different best models for different time windows.

Time-series Meta-Features: A key component of AutoForecast is the extraction of meta-features that aims to capture the important characteristics of a time-series dataset. To achieve such a goal, we extract meta-features (defined below) for each time-series dataset.

Definition 4. Given a time-series dataset $\boldsymbol{D}_i$, we define the meta-features tensor $\mathbf{F}_i = \{F_1^i, \cdots, F_T^i\} \in \mathbb{R}^{T\times d\times v_i}$, where the meta-features matrix $F_k^i \in \mathbb{R}^{d\times v_i}$ denotes the set of the meta features for the time window w_k of the dataset $\boldsymbol{D}_i$, given by

$$F_k^i \triangleq \{\psi(w_k(\boldsymbol{D}_i)) : \psi : \mathbb{R}^{|w_i|\times v_i} \longrightarrow \mathbb{R}^{d\times v_i}\}, \tag{2}$$

where $\psi(\cdot) : \mathbb{R}^{|w_i|\times v_i} \longrightarrow \mathbb{R}^{d\times v_i}$ defines feature extraction module in AutoForecast and d denotes the number of the meta-features.[2]

Meta-Features Categories: The set of meta features in our work that capture the main characteristics of a dataset can be organized into five categories [50]: simple (general task properties), statistical (properties of the underlying dataset distributions), information-theoretic (entropy measures), spectral (frequency domain properties), and landmarker (forecasting models' attributes on the task) features. The idea of our proposed landmarker features is to apply a few of the fast, easy-to-construct time-series forecasting models on a dataset and extract features from (i) the structure of the estimated forecasting model, and (ii) its output performance scores. The complete meta-features list in AutoForecast are explained in Section 5.1.

4 AUTOFORECAST

AutoForecast consists of two-phases: offline training of the meta-learner and online inference that aims at selecting the appropriate model at test time. We argue that running time of the offline training phase is not critical since it is done only once. On the contrary, forecasting model selection for a new time-series dataset should incur small run-time overhead since it is critical for quick selection of the forecasting model. We now explain our meta-learning approach and its components.

4.1 Meta-Learning Objective and Training

We show the overview of the major components of AutoForecast in Figure 1. We highlight the components transferred from offline to online stage (model selection) in blue; namely, meta-feature extractors ψ, feature embedding, time-series meta-learner $\boldsymbol{\Theta}$, and general meta-learner $\boldsymbol{\Phi}$. The meta-learner $\mathcal{L}$ has three main inputs; the performance tensor $\mathbf{P}$, the meta-features tensor $\mathbf{F}$, and the loss function. In the offline training of the meta-learner $\mathcal{L}$, it learns two components $\boldsymbol{\Theta}$ and $\boldsymbol{\Phi}$. The time-series meta-learner $\boldsymbol{\Theta}$ captures the temporal relationship between the meta-features of the consecutive time windows within the same dataset and the evolution of the performances of the models on these windows. On the other hand, the general meta-learner $\boldsymbol{\Phi}$ predicts the best model for each task (window) without taking into account the temporal relationship among different time windows within the same dataset.

Rationale for Having both General and Time-series Meta-learners: The rationale of having both meta-learners is the fact that the temporal dependency among different time windows depends on the dataset type. Some datasets have strong temporal dependency which would be predicted efficiently by the time-series meta-learner $\boldsymbol{\Theta}$ while others datasets would have weak temporal dependence among different windows performances in which the general meta-learner $\boldsymbol{\Phi}$ is expected to perform better. We show the results of such different datasets for our two testbeds in performance benchmark folder within our anynomized link (in Section 1).

General Meta-learner $\boldsymbol{\Phi}$: We propose *multi-output regression model* for training our general meta-learner $\boldsymbol{\Phi}$. From running all the models in $\mathcal{M}$ on different time windows w_t with $t \in \{1, \ldots, T\}$ for all datasets in the training database $\mathcal{D}_{train}$, we collect a set of $N = T \times n$ distinct training samples of the meta-features matrix and the performance vector $(F_t^i, \boldsymbol{p}_t^i)$, with $t \in [1, T]$ and $i \in [1, n]$. Thus, the multi-output regression model is given by

$$\hat{\boldsymbol{p}}_t^i = \boldsymbol{\Phi}\left(F_t^i, \boldsymbol{\beta}\right); t \in [1, T], i \in [1, n], \tag{3}$$

where $\boldsymbol{\Phi}$ denotes the regression function (e.g., linear, NN) and $\boldsymbol{\beta}$ are the unknown regression parameters. Thus, the general meta-learner's objective, denoted by loss function $L_{\boldsymbol{\Phi}}$, is given by

$$L_{\boldsymbol{\Phi}} = \sum_{t=1}^{T}\sum_{i=1}^{n} L(\hat{\boldsymbol{p}}_t^i, \boldsymbol{p}_t^i), \tag{4}$$

where L is the loss metric (e.g., MSE, MAPE, etc). Therefore, $\boldsymbol{\Phi}$ learns the mapping between the meta-features of a time window in a dataset and the corresponding best model in the model space.

LSTM-based Time-series Meta-learner $\boldsymbol{\Theta}$: The goal of the time-series meta-learner $\boldsymbol{\Theta}$ is to learn how the models' performances evolve with the time-series meta-feature matrices over time. For this purpose, we propose *time-series multi-regression model* to learn such performance evolution. For any dataset $\boldsymbol{D}_i$, given the time-series meta-feature matrices $F_1^i, F_2^i, \ldots, F_t^i$ and the history of the performance vectors $\boldsymbol{p}_1^i, \ldots, \boldsymbol{p}_{t-1}^i$, we aim to predict performance vector $\boldsymbol{p}_t^i$ of current time window w_t. The time-series regression equation would be

$$\hat{\boldsymbol{p}}_t^i = \boldsymbol{\Theta}\left(F_1^i, \ldots, F_{t-1}^i, F_t^i, \boldsymbol{p}_1^i, \ldots, \boldsymbol{p}_{t-1}^i\right), i \in [1, n], t \in [1, T], \tag{5}$$

where $\boldsymbol{\Theta}$ denotes the time-series regression function.

We adapt long-short term memory (LSTM) inputs for our time-series meta-learner $\boldsymbol{\Theta}$. We denote $\boldsymbol{X}_t$ as the input at the time window w_t which is given by $\boldsymbol{X}_t = \left[F_1^i, \boldsymbol{p}_1^i, F_2^i, \boldsymbol{p}_2^i, \cdots, F_{t-1}^i, \boldsymbol{p}_{t-1}^i, F_t^i\right]$. The predicted LSTM's output denoted by $\hat{\boldsymbol{p}}_t^i$ is a function of $\boldsymbol{X}_t$. We now provide the detailed equations of such relation between $\hat{\boldsymbol{p}}_t^i$ and $\boldsymbol{X}_t$. The LSTM cell at time t has two recurrent features, denoted by $\boldsymbol{h}_t^i$ and $\boldsymbol{c}_t^i$, called the hidden state and the cell state, respectively. The LSTM cell consists of three layers (forget gate layer, input gate layer, and output gate layer). The activation of those layers is given by

$$\begin{aligned} \boldsymbol{f}_t^i &= \sigma\left(\boldsymbol{W}_f \cdot [\boldsymbol{h}_{t-1}^i, \boldsymbol{X}_t] + \boldsymbol{b}_f\right), \\ \boldsymbol{l}_t^i &= \sigma\left(\boldsymbol{W}_l \cdot [\boldsymbol{h}_{t-1}^i, \boldsymbol{X}_t] + \boldsymbol{b}_l\right), \\ \boldsymbol{o}_t^i &= \sigma\left(\boldsymbol{W}_o \cdot [\boldsymbol{h}_{t-1}^i, \boldsymbol{X}_t] + \boldsymbol{b}_o\right). \end{aligned}$$

where $\boldsymbol{W}_f, \boldsymbol{W}_l, \boldsymbol{W}_o$ and $\boldsymbol{b}_f, \boldsymbol{b}_l, \boldsymbol{b}_o \in \mathbb{R}^m$ denote the weights matrices and the biases of the three layers, respectively. These are the parameters to be learned during the training of the time-series meta-learner. Moreover, the cell update $\mathbf{u}_t^i$ is constructed with a tanh activation function as follows.

$$\mathbf{u}_t^i = \tanh\left(\boldsymbol{W}_u \cdot [\boldsymbol{h}_{t-1}^i, \boldsymbol{X}_t] + \boldsymbol{b}_u\right),$$

[2] Note that we do feature embedding (PCA), shown in Figure 1, to get the final meta-features tensor $\boldsymbol{F}_i$. Such embedding eliminates redundancy among features.

where $\boldsymbol{W}_u$ and $\boldsymbol{b}_u \in \mathbb{R}^m$ are further weight and bias parameters to be learned. Thus, the new cell and hidden states at time t are

$$\boldsymbol{c}_t^i = \boldsymbol{f}_t^i \cdot \boldsymbol{c}_{t-1}^i + \boldsymbol{l}_t^i \cdot \mathbf{u}_t^i$$
$$\hat{\boldsymbol{h}}_t^i = \boldsymbol{o}_t^i \cdot \tanh(\boldsymbol{c}_t^i)$$

Finally, the output equations of the LSTM cell are given by

$$\mathrm{V}_t^i = \boldsymbol{W}_v \hat{\boldsymbol{h}}_t^i + \boldsymbol{b}_v$$
$$\hat{\boldsymbol{p}}_t^i = \sigma(\mathrm{V}_t^i),$$

with $\boldsymbol{W}_v$ and $\boldsymbol{b}_v \in \mathbb{R}^m$ are learned weight and bias parameters. This gives the relationship between the input $\boldsymbol{X}_t$ and the predicted performance output vector $\hat{\boldsymbol{p}}_t^i$.

During training, $\boldsymbol{\Theta}$ learns the parameters $\boldsymbol{W}_f, \boldsymbol{b}_f, \boldsymbol{W}_l, \boldsymbol{b}_l, \boldsymbol{W}_o, \boldsymbol{b}_o, \boldsymbol{W}_u, \boldsymbol{b}_u, \boldsymbol{W}_v, \boldsymbol{b}_v$ which are the weights and biases of the forget, input, and output layers and cell updates, respectively. Thus, the objective of the LSTM time-series meta-learner $\boldsymbol{\Theta}$, is to minimize the loss denoted by $L_{\boldsymbol{\Theta}}$, given by

$$L_{\boldsymbol{\Theta}} = \sum_{t=1}^{T} \sum_{i=1}^{n} L(\hat{\boldsymbol{p}}_t^i, \boldsymbol{p}_t^i). \tag{6}$$

We emphasize that $\boldsymbol{\Theta}$ learns the appropriate current model (from the model space) given both the history of the meta-features and performance vectors over time windows.

Meta-learner Objective: Having established the two main components, the general meta-learner $\boldsymbol{\Phi}$ and the time-series meta-learner $\boldsymbol{\Theta}$, we now define the objective of our meta-learner $\mathcal{L}$, given by a linear combination of the two components as follows:

$$\min_{\boldsymbol{\beta}, \boldsymbol{W}_f, \boldsymbol{W}_l, \boldsymbol{W}_o, \boldsymbol{b}_f, \boldsymbol{b}_l, \boldsymbol{b}_o, \boldsymbol{W}_u, \boldsymbol{b}_u, \boldsymbol{W}_v, \boldsymbol{b}_t} aL_{\boldsymbol{\Phi}}(\mathbf{F}, \mathbf{P}) + (1-a)L_{\boldsymbol{\Theta}}(\mathbf{F}, \mathbf{P}), \tag{7}$$

The parameter a defines the relative weight of the two meta-learners. In our experiments, we chose $a = 0.5$ for training our meta-learner. The meta-learner $\mathcal{L}$ learns jointly general meta-learner $\boldsymbol{\Phi}$ (Equation 4) and time-series meta-learner $\boldsymbol{\Theta}$ (Equation 6) given meta-learner inputs, the performance tensor $\mathbf{P}$ and the meta-features tensor $\mathbf{F}$. By definition, this meta-learner $\mathcal{L}$ optimizes the loss over all datasets and all time windows.

4.2 Online Inference And Model Selection

In the online mode of AutoForecast, we aim to make use of the trained meta-leaner $\mathcal{L}$ to quickly infer the best model for the current task. Given a new time-series dataset $\boldsymbol{D}_{test}$, AutoForecast first computes the corresponding meta-features tensor $\hat{\mathrm{F}}_{test} = \psi(\boldsymbol{D}_{test})$. Those time-series meta-features are then embedded (using PCA) to obtain the final meta-features tensor $\boldsymbol{F}_{test}$. Then, in the model inference, as shown in Figure 1, the model set performances are predicted for each available model in $\mathcal{M}$. The model $\hat{M}_t$ with the lowest predicted error score by $\mathcal{L}$ on the time window w_t of $\boldsymbol{D}_{test}$ is chosen as the selected model for that window w_t. Such a process is repeated for all time windows $w_0, w_1, \dots, w_T$ of $\boldsymbol{D}_{test}$.

Now, we explain such model selection process for each time window across the time windows $w_0, w_1, \dots, w_T$ of $\boldsymbol{D}_{test}$ as follows. For the first window (w_0), the inference is given by $\hat{M}_0 \in \arg\min_{\bar{M}\in\mathcal{M}} \mathcal{L}(F_0^{test})$. For any other window w_t ($t > 0$), the time-series meta-learner $\boldsymbol{\Theta}$ inference depends on the history of the meta-features and the history of the models' performances as follows $\hat{M}_t^{\Theta} \in \arg\min_{\bar{M}\in\mathcal{M}} \Theta(\boldsymbol{F}_0^{test}, \dots, \boldsymbol{F}_{t-1}^{test}, \boldsymbol{F}_t^{test}, \hat{\boldsymbol{p}}_0^{test}, \dots, \hat{\boldsymbol{p}}_{t-1}^{test})$. On the other hand, the general meta-learner $\boldsymbol{\Phi}$ inference depends on the predicted (regression) output on the meta-features of current time window where $\hat{M}_t^{\Phi} \in \arg\min_{\bar{M}\in\mathcal{M}} \Phi(\boldsymbol{F}_t^{test})$. Thus, the final selected model is given by

$$\hat{M}_t \in \underset{\bar{M}\in\{\hat{M}_t^{\Phi}, \hat{M}_t^{\Theta}\}}{\arg\min} \hat{\boldsymbol{p}}_t^{test}(\bar{M}). \tag{8}$$

We emphasize that tie between models can happen in online inference (i.e., two or more models can have an identical predicted performance). We built upon the several tie breaking techniques that have been examined in the literature [8, 23], but usually such a choice does not have a strong influence on the performance of AutoForecast. For making the decision between the model selected by the general meta-learner $\boldsymbol{\Phi}$ and that selected by the time-series meta-learner $\boldsymbol{\Theta}$, we choose the model with the best performance (i.e., least predicted error score) (Equation 8).

Inference Time Complexity: Recall that the number of meta-features is d and the number of models in our model space $\mathcal{M}$ is m. The time complexity for the inference part of the general meta-learner $\boldsymbol{\Phi}$ is $O(d)$. On the other hand, the time complexity of the time-series LSTM meta-learner $\boldsymbol{\Theta}$ is given by $O(d \times |X_t|)$, where $|X_t|$ is the length of the input sequence. Therefore, AutoForecast's inference time is given by $O(d \times |X_t|)$. We emphasize that the inference times of the naïve method is much larger since it is given by $O(\sum_{i=1}^{m} I_i)$ (summation of inference time of all algorithms), where I_i is the inference times of forecasting algorithm a_i.

5 EXPERIMENTS

We evaluate AutoForecast by designing experiments to answer the following research questions:

(1) Does employing AutoForecast for time-series forecasting model selection yield improved performance, as compared to no model selection, as well as other selection techniques (such as meta-learners adapted from the AutoML domain)?
(2) How much reduction in inference time does AutoForecast give over the naïve method?
(3) How does performance change with different datasets with different temporal dependencies?

5.1 Experimental Setup

Models and Performance Collection: By pairing seven SOTA time-series forecasting algorithms (which are DeepAR [41], Deep Factors [52], Prophet [48], Seasonal Naive [30], Gaussian Process [55], Vector Auto Regression [25], and Random Forest Regressor [27]) and their corresponding hyperparameters, and using different data representation methods, we compose a model set $\mathcal{M}$ with 322 unique models (see Table 1 for the complete list). For our testbeds, we first generate the performance tensor $\mathbf{P}$, by evaluating the models from $\mathcal{M}$ against the benchmark datasets in each testbed. For consistency, all models are built using the GluonTS [3], Scikit-learn [36], and Statsmodels [42] Python libraries on an Intel i7 @2.60 GHz, 16GB RAM, 8-core workstation.

Time-series Meta Features: There are prior works that generated standard time-series features [17], tsfresh [12] (that we used for generating part of our meta-features).

Table 1: Time-Series Forecasting Model Space. See hyperparameter definitions for various algorithms from GluonTS [3] and statsmodels [42]. The number of models (last column) is all possible combinations of hyperparameters and data representations.

Forecasting Algorithm	HyperParameter 1	HyperParameter 2	Data Representation	Total
DeepAR	num_cells = [10,20,30,40,50]	num_rnn_layers = [1,2,3,4,5]	{Exp_smoothing, Raw}	50
DeepFactor	num_hidden_global = [10,20,30,40,50]	num_global_factors = [1,5,10,15,20]	{Exp_smoothing, Raw}	50
Prophet	changepoint_prior_scale = [0.001, 0.01, 0.1, 0.2, 0.5]	seasonality_prior_scale = [0.01, 0.1, 1.0, 5.0, 10.0]	{Exp_smoothing, Raw}	50
Seasonal Naive	season_length = [1,5,7,10,30]	N/A	{Exp_smoothing, Raw}	10
Gaussian Process	cardinality = [2,4,6,8,10]	max_iter_jitter = [5,10,15,20,25]	{Exp_smoothing, Raw}	50
Vector Auto Regression	cov_type= {"HC0","HC1","HC2","HC3","nonrobust"}	trend = {'n', 'c', 't', 'ct' }	{Exp_smoothing, Raw}	40
Random Forest Regressor	n_estimators = [10,50,100,250,500,1000]	max_depth = [2,5,10,25,50,'None']	{Exp_smoothing, Raw}	72
				322

Table 2: Hit-at-k Accuracy (the higher the better) comparison of AutoForecast **against the different baseline meta-learners for both univariate and multivariate testbeds.** AutoForecast **outperforms all baselines for both testbeds.**

Dataset Testbed	k	Global Best	AS	ISAC	MLP	AutoForecast-TSL	AutoForecast
Univariate	1	2.46	2.15	0.82	0.62	2.67	**3.95**
	5	7.18	4.92	2.67	1.13	9.04	**14.57**
	10	11.97	7.89	4.10	4.51	14.15	**21.45**
	50	37.40	28.00	11.45	22.25	35.28	**52.05**
Multivariate	1	**6.78**	2.26	4.19	0.43	5.16	5.87
	5	12.18	4.73	5.69	1.51	9.03	**13.86**
	10	16.21	9.03	7.31	4.06	11.39	**20.91**
	50	41.72	24.73	14.64	20.86	35.06	**51.67**

We now provide details of our meta-features (shared with our database and source codes in the link provided in Section 1). For each dataset, we generate a meta-feature vector that consists of more than 800 meta-features where some of them are based on [50]. Specifically, our meta-features can be categorized into (1) simple features, (2) statistical features, (3) information-theoretic features, (4) Spectral features, and (5) landmarker features. Broadly speaking, the statistical features captures statistical properties of the underlying data distributions; e.g., min, max, variance, skewness, covariance, etc. of the features and feature combinations. The information-theoretic features capture information-theoretic underlying characteristics in the time-series; e.g., entropy, trend, non-linearity, change statistics, etc. Most of those meta-features have been commonly used in the AutoML literature [50]. Our meta-features vector also includes landmarker features, which are problem-specific, and aim to capture the unique characteristics of a dataset. The idea is to apply a few of the fast, easy-to-construct time-series forecasting models on a dataset and extract features from (i) the structure of the estimated forecasting model, and (ii) its output performance scores. We emphasize that our landmarker meta-features are novel and that some components of the spectral meta-features have not been used in any related work.

Training Testbed Sources: Meta-learning works if the new task can leverage prior knowledge. Our testbeds are built to simulate the case when meta-train comes from many different distributions. This diversity enhancing the training of the meta-learning model. Model selection on test data can thus benefit from the prior experience on the train set. We thus have created a repository of 348 forecasting datasets including two hitherto unreleased ones from Adobe's production compute clusters. In particular, most of the datasets are from different application domains (e.g., finance, IoT, energy, storage, etc.) where we use benchmark datasets from Kaggle [21], Adobe real traces, and other open source repositories. The Adobe trace datasets records CPU and Memory usage for 50 different services running in Adobe production clusters collected for 15 days from May 1 to May 15 in 2021. Such traces are shared for the first time in our current work.

Dataset Types in AutoForecast**:** We consider two general types of time-series datasets depending on the number of the variables v_i in the time-series dataset. *(1) Univariate Datasets* with single time-series ($v_i = 1$) and *(2) Multivariate Datasets* with several time-series (variables) (i.e., $v_i > 1$) that need to be predicted. In particular, we collect 308 univariate time-series datasets for the first testbed and 40 multivariate datasets with 317 time-series for the second testbed. In total, we have 625 time-series in our testbeds. For each dataset in the testbeds, we use different time windows selected randomly from the dataset, where each time window has a length of 16 (i.e., $|w_t| = 16 \ \forall t \in \{1, \ldots, T\}$). Note that our approach has no restriction on the length of the time window nor any assumption of the homogeneity of windows duration.

Evaluation: For evaluating AutoForecast, for robustness, we split each testbed into 5 folds for cross-validation. We build the train/test testbed by each time selecting four folds from the datasets for training and the remaining fifth fold for testing (i.e., after training the meta-learning approach, we use it to infer the best forecasting model for the new unseen test datasets of that fifth fold). Finally, we take the average performance of these five folds. We mainly compare the Hit-at-k accuracy of AutoForecast against different meta-learners baselines. This metric indicates whether the selected model fits within the top-k models from ground truth data. We also compare using the metric, mean square error (MSE) and the average rank. For each testbed, the meta-learners are first ranked by the corresponding forecasting MSE under the selected model for each dataset and then the rank is averaged across all datasets.

Time-series Meta-learner Setup: For our time-series meta-learner Θ explained in Section 4, we used LSTM with 4 layers where each layer has 50 units. The training was with 50 epochs with the Adam optimizer with a batch size of 25 and dropout rate of 0.2 to prevent over-fitting. A detailed evaluation of the effect of such parameters on Θ's performance is available and is omitted here due to space constraints.

Table 3: Average rank (the lower the better) comparison of AutoForecast against the different baseline meta-learners for both testbeds. AutoForecast outperforms all baselines.

Dataset Testbed	Global Best	AS	ISAC	MLP	AutoForecast-TSL	AutoForecast
Univariate	2.5161	2.7965	2.9096	3.7072	2.5202	**2.0571**
Multivariate	2.3191	3.0851	2.3191	3.8723	2.3404	**1.3191**

5.2 Baselines

We adapt recent meta-learning approaches to our specific problem setting, and also include a few methods that do not perform model selection.

No model selection: This category always employs either the same single model or the ensemble of all the models:

- **Random Forest (RF) [27]**: is a SOTA tree ensemble that combines the predictions made by many decision trees into a single model. In prediction, the RF regression model takes the average of all the individual decision tree estimates.
- **SOTA Forecasting Algorithms:** We selected seven popular time series forecasting models, including the recent works DeepAR [41], DeepFactors [52], and Prophet [48]. For each model, we generated multiple variants by varying the values of hyperparameters and data representations (10-72 variants, details in Table 1) and we chose the model variant with the best average performance across all training datasets.

Simple meta-learners: Meta-learners in this category pick the generally well-performing forecasting model, globally or locally:

- **Global Best (GB)**: It selects the forecasting model with the largest average performance across all train datasets (across all time windows), without using any meta-features.
- **ISAC [20]**: clusters the training datasets based on meta-features. Given a new test time-series dataset, it identifies its closest cluster and selects the best model with largest average performance on the cluster’s datasets.
- **ARGOSMART (AS) [32]**: finds the closest training time-series dataset to a given test time-series dataset, based on meta-feature similarity, and selects the model with the best performance on that nearest neighbor training dataset.

Optimization-based meta-learners: Meta-learners in this category learn meta-feature by task similarities toward optimizing performance estimates:

- **Multi-layer Perceptron (MLP)**: Given the training datasets and selected time window, the MLP regressor directly maps the meta-features onto model performances by regression. However, this does not learn temporal dependence within datasets.
- **AUTOFORECAST-TSL**: is a variant of our solution in which the meta-learner $\mathcal{L}$ consists only of the time-series learner $\mathbf{\Theta}$.

5.3 Results

5.3.1 Variation of Best Model across Time. Figure 2 shows that no single forecasting model triumphs in more than 0.7% of the datasets. Figure 3 shows the aggregate statistics on all datasets of the univariate testbed. This contradicts the claims that one forecasting algorithm can work best for different datasets and motivates the need for an effective approach for learning such both dimensions, which we propose in our current work.

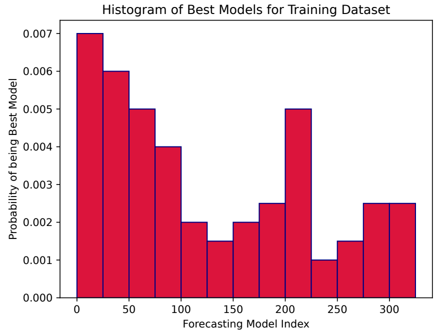

Figure 2: A histogram of the best forecasting model’s probability distribution across datasets of our two training testbeds. Different datasets have different best models and no single model triumphs in more than 0.7% of the datasets.

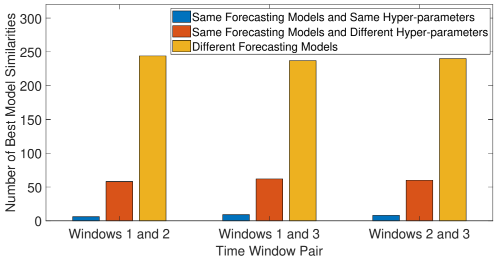

Figure 3: The aggregate statistics for similarity in the best forecasting model across three consecutive time windows for univariate testbed. Most different time windows have different (best) models.

5.3.2 Univariate Testbed Results. To investigate the impact of the train/test similarity on meta-learning performance, we build the univariate testbed that consists of 308 diverse datasets.

Superiority of AutoForecast compared to all baseline methods w.r.t. the Hit-at-k, average rank, and MSE: The different results are provided in Tables 2-4 where the best result for every testbed is highlighted in bold. We observe that AutoForecast outperforms previous SOTA meta-learning methods adapted to our problem. For example, AutoForecast has 79.20%, 171.86%, 423.17%, 375.61%, and 51.59% higher Hit-at-10 accuracy than GB, AS , ISAC, MLP, and AutoForecast-TSL, respectively.

Statistical Significance of AutoForecast: To compare two methods statistically, we use the pairwise Wilcoxon rank test on performances (i.e., MSE of selected models) across datasets (significance level $p < 0.05$). Table 5 shows that AutoForecast is significantly better than most of the baseline meta-learners, i.e., including GB (9.07×10^{-5}), AS (1.07×10^{-37}) and AutoForecast-TSL (8.16×10^{-15}).

Table 4: Results for one-step ahead forecasting (MSE; the lower the better) for both testbeds. The selected model by AutoForecast **yields better performance compared to baseline meta-learners and SOTA methods. For each SOTA, we choose the model with the best average performance from all its model variants.**

Dataset Testbed	Seasonal Naive	DeepAR	Deep Factors	Random Forest	Prophet	Gaussian Process	VAR
Univariate	0.0345 ± 0.0382	0.0164 ± 0.0506	0.0217 ± 0.0415	0.0199 ± 0.0398	0.0155 ± 0.0295	0.1661 ± 0.2104	0.0602 ± 0.1260
	Global Best	AS	ISAC	MLP	AutoForecast-TSL	AutoForecast	
	0.0065 ± 0.0199	0.0158 ± 0.0556	0.0071 ± 0.0145	0.0351 ± 0.1186	0.00463 ± 0.0138	**0.00256 ± 0.0090**	
Multivariate	Seasonal Naive	DeepAR	Deep Factors	Random Forest	Prophet	Gaussian Process	VAR
	0.0149 ± 0.0408	0.0085 ± 0.0197	0.0135 ± 0.0232	0.0071 ± 0.0365	0.0065 ± 0.0153	0.2576 ± 0.1344	0.9865 ± 0.2988
	Global Best	AS	ISAC	MLP	AutoForecast-TSL	AutoForecast	
	0.0046 ± 0.0099	0.0139 ± 0.0563	0.0046 ± 0.0099	0.0121 ± 0.2462	0.00541 ± 0.0186	**0.00124 ± 0.0051**	

Table 5: Pairwise statistical test results between every pair of methods by Wilcoxon signed rank test. Statistically better method ($p = 0.05$) shown in bold (both marked bold if no significance). In the left, Univariate testbed is shown. In the right, Multivariate testbed is shown. For both testbeds, AutoForecast **is statistically better than most of the baseline meta-learners.**

Method 1	**Method 2**	**p-value**
AutoForecast	GB	9.0712×10^{-5}
AutoForecast	AS	1.0726×10^{-37}
AutoForecast	**ISAC**	0.1349
AutoForecast	**MLP**	0.0657
AutoForecast	AutoForecast-TSL	8.1683×10^{-15}
AutoForecast-TSL	GB	2.2611×10^{-8}
AutoForecast-TSL	AS	1.5760×10^{-14}
AutoForecast-TSL	ISAC	2.3843×10^{-16}
AutoForecast-TSL	MLP	1.1658×10^{-26}
GB	AS	9.4952×10^{-33}
GB	ISAC	0.0322
GB	MLP	4.5489×10^{-9}
AS	ISAC	1.7842×10^{-37}
AS	MLP	4.4658×10^{-54}
ISAC	MLP	2.2062×10^{-31}

Method 1	**Method 2**	**p-value**
AutoForecast	**GB**	1.0
AutoForecast	AS	3.9399×10^{-7}
AutoForecast	**ISAC**	0.8240
AutoForecast	MLP	0.0004
AutoForecast	AutoForecast-TSL	0.0025
AutoForecast-TSL	GB	0.00254
AutoForecast-TSL	AS	5.8013×10^{-7}
AutoForecast-TSL	ISAC	1.5598×10^{-5}
AutoForecast-TSL	MLP	3.4572×10^{-8}
GB	AS	3.9399×10^{-7}
GB	**ISAC**	0.8240
GB	MLP	0.0004
AS	ISAC	1.4217×10^{-7}
AS	MLP	6.6612×10^{-8}
ISAC	MLP	3.7789×10^{-8}

Meta-learners perform better than methods without model selection: Table 4 shows that meta-learners outperform almost all models with no model selection. In particular, three meta-learners (AutoForecast, Global Best, ISAC) significantly outperform baseline time-series forecasting models. For instance, AutoForecast has 92.58%, 84.39%, 88.20%, 87.14%, 83.48%, 98.45%, and 95.75% lower MSE over Seasonal Naive, DeepAR, Deep Factors, Random Forest, Prophet, Gaussian Process, and VAR, respectively. These results signify the benefits of using meta-learning for model selection, specifically using AutoForecast.

Optimization-based meta learners generally perform better than simple meta learners: Two of the top-3 meta learners by average rank and MSE (AutoForecast and AutoForecast-TSL) are all optimization-based and significantly outperform simple meta-learners such as ISAC and AS as shown in Table 2 and Table 4. The interpretation is that simple meta-learners weigh meta-features equally for task similarity, whereas optimization-based methods learn which meta-features matter (e.g., time-series regression on meta-features in AutoForecast-TSL), leading to better results.

Dataset-wise Performance: We present the detailed performances for each dataset by comparing AutoForecast with all baseline methods in our anonymized link (provided in the Introduction as footnote). It is noted these results are averaged across the different time windows for each dataset. The results shows that AutoForecast achieves the best average MSE and average rank among all meta-learners. We note that AutoForecast and AutoForecast-TSL have same performance for datasets with higher temporal dependency.

5.3.3 Multivariate Testbed Results. In this testbed, we choose some time series within one dataset for training and a disjoint set of time series within the same dataset for testing. Our multivariate testbed consists of 40 datasets.

For the Multivariate testbed, AutoForecast **still outperforms all baseline methods w.r.t. average rank, MSE, and Hit-at-*k* accuracy** as shown in Tables 2-4. Moreover, Figure 4 shows that for the pool of multivariate datasets (across all time windows), AutoForecast gives a gain of 2X and higher compared to other meta-learning baselines. **Dataset-wise** benchmark performance for the datasets in the multivariate testbed is shown in our anonymized link. AutoForecast has the lowest average MSE on most of the multivariate datasets.

Statistical Significance of AutoForecast**:** Table 5 shows that for Multivariate testbed, AutoForecast is also significantly better than most of the baseline meta-learners, i.e., including AS and MLP while there is no significant statistical difference from GB and ISAC.

5.3.4 Runtime Analysis. **Inference run time statistics of** AutoForecast**:** Table 6 shows that AutoForecast (meta-feature generation and model selection) takes 1.7 seconds on most time series datasets. Moreover, Figure 5 shows that AutoForecast has significant reduction in inference time compared to the naïve approach (i.e., doing inference using all possible models and then selecting the model with the best performance), median is 42× across the two testbeds (i.e., 41× on univariate testbed and 45× on multivariate testbed).

Comparing AutoForecast **with baselines:** In terms of inference, Table 6 shows that most of the meta-learners are fast, taking less than 2 seconds to infer the best forecasting model. Finally, we compare the training cost of AutoForecast against the baseline meta-learners. Table 6 also shows that AutoForecast has comparable computational training cost. While the training process is offline and done only once and hence is less critical, this

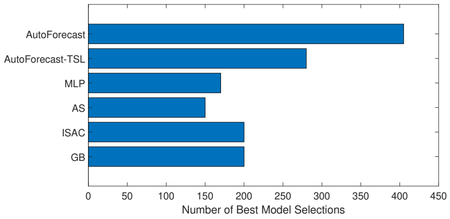

Figure 4: The number of best model selections by each meta-learning approach. AutoForecast **has 2× gain in the selection of best model compared to the closest baselines (ISAC and GB).**

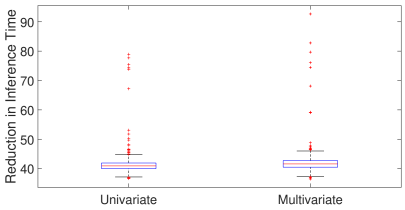

Figure 5: The inference time reduction of AutoForecast **over the naïve approach.** AutoForecast **gives a median reduction of 42X over naïve approach for both testbeds.**

Table 6: Average and standard deviation inference and training runtime performance (in seconds) for both dataset testbeds.

Phase	Dataset Testbed	Naïve	Global Best	AS	ISAC	MLP	AutoForecast-TSL	AutoForecast
Inference	**Univariate**	70.9500 ± 1.7801	0.6259 ± 0.0964	0.8537 ± 0.1438	10.2480 ± 2.7182	1.2745 ± 0.5198	0.7962 ± 0.0436	1.6508 ± 0.0401
	Multivariate	48.6287 ± 5.4051	0.4151 ± 0.0403	1.3055 ± 0.2610	7.037 ± 1.6239	1.1461 ± 0.2176	0.682 ± 0.0372	1.1309 ± 0.1257
Training	**Univariate**	N/A	N/A	308.9301 ± 46.1968	278.8083 ± 57.9900	705.2908 ± 123.3715	334.8091 ± 31.6808	670.5855 ± 31.5465
	Multivariate	N/A	N/A	194.3877 ± 39.7441	182.4753 ± 34.3238	411.9337 ± 41.7406	178.1978 ± 18.0098	376.3956 ± 40.0195

experiment emphasizes that our better model selection performance does *not* entail a prohibitive training cost.

6 DISCUSSION

(1) Reproducibility of AutoForecast**:** To further research into the important problem introduced in our work, we have publicly released our source codes and benchmark data to enable others reproduce our work. In particular, we are publicly releasing, with this submission, our meta-learning database corpus of 348 datasets, containing 625 time series in all, performances of the 322 forecasting models, and meta-features for the datasets. This resource will hopefully encourage the community to standardize efforts at benchmarking time series forecasting model selection. We also encourage the community to expand this resource by contributing their new datasets and models. The anonymized website with our database and source codes is: https://drive.google.com/drive/folders/1K1w1Ida5Cr15b5Fhidax-i-fNpWZjvet. The details of each dataset in the two testbeds and the different categories of meta-features are presented in Section 5.1. This serves as the training data and ground truth evaluation data for AutoForecast.

(2) Usage of Meta-Learning in AutoForecast**:** We emphasize that we use the term "meta-learning" in the context of traditional principle of meta-learning which is building upon prior experience on a set of historical tasks to "do better" on a new task. We build the experience across different datasets using our general meta-learner and build experience on the sequential temporal dependence within the same dataset using the LSTM time-series meta-learner. We also capture task similarity between a new input task (dataset) and historical datasets using the "meta-features". We also emphasize that our proposed method is faster compared to gradient descent-based meta-learners, equivalently pure Deep Learning-based meta-learners, [18, 34], e.g., on our univariate testbed, N-Beats [34] has significantly slower training (average = 3600 seconds) and inference time (average = 101 sec) compared to AutoForecast (average = 670 seconds for training and 1.13 sec for inference). Integrating our meta-learning approach with a deep learning approach is a potential area of future work.

(3) Diversity of Datasets: We acknowledge that diversity of sources makes the meta-learning model learn from such diversity since model selection on test data would benefit from the prior experience on that diverse train set. For that purpose, we use benchmark datasets from Kaggle, Adobe real traces, and other open-source repositories, where the datasets are from different application domains (e.g., finance, IoT, energy, storage, etc.). We have released these benchmark datasets (anonymized link provided earlier in this Section) to help the community build on our work.

7 CONCLUSION

We introduced a meta-learning approach to automate the process of time-series forecasting by automatically inferring the best time-series model on an unseen dataset, without needing exhaustive evaluation of all existing models on this dataset. The problem arises because there are many possible forecasting models with their associated hyperparameters, and different choices are optimal for different datasets. Our proposed solution AutoForecast is a meta-learner, trained on an extensive pool of historical time-series forecasting datasets and models. To effectively capture dataset similarity, we designed novel problem-specific meta-features. AutoForecast generalizes to new datasets since it learns two models from the training corpus: *first*, the mapping between the meta-features vector and the corresponding best-model via the general meta-learner (Φ); and *second*, the evolution of the models' performances in time with the meta-features and previous models' performances via the time-series meta-learner (Θ). Thus, for a new different dataset, we extract its meta-features vector and then determine the best model using the pre-trained (Φ) and (Θ). Extensive experiments on two large testbeds, univariate and multivariate, showed that AutoForecast significantly improves time-series forecasting model selection over directly using some of the most popular models as well as several SOTA meta-learners. We showed that AutoForecast gives a significant improvement in the inference time compared to naïve approaches. We release the benchmark data for the community to contribute new datasets and models to further advance automating time-series forecasting.

REFERENCES

[1] H. Abbasimehr, M. Shabani, and M. Yousefi. An optimized model using lstm network for demand forecasting. *Computers & industrial engineering*, 143:106435, 2020.

[2] M. Abdallah, W. J. Lee, N. Raghunathan, C. Mousoulis, J. W. Sutherland, and S. Bagchi. Anomaly detection through transfer learning in agriculture and manufacturing iot systems. *arXiv preprint arXiv:2102.05814*, 2021.

[3] A. Alexandrov, K. Benidis, M. Bohlke-Schneider, V. Flunkert, J. Gasthaus, T. Januschowski, D. C. Maddix, S. S. Rangapuram, D. Salinas, J. Schulz, et al. Gluonts: Probabilistic and neural time series modeling in python. *J. Mach. Learn. Res.*, 21(116):1–6, 2020.

[4] B. Arinze, S.-L. Kim, and M. Anandarajan. Combining and selecting forecasting models using rule based induction. *Computers & Operations Research*, 24(5):423–433, 1997.

[5] C. Bergmeir and J. M. Benítez. On the use of cross-validation for time series predictor evaluation. *Information Sciences*, 191:192–213, 2012.

[6] J. Bergstra, R. Bardenet, Y. Bengio, and B. Kégl. Algorithms for hyper-parameter optimization. In *25th annual conference on neural information processing systems (NIPS 2011)*, volume 24. Neural Information Processing Systems Foundation, 2011.

[7] J. Bergstra and Y. Bengio. Random search for hyper-parameter optimization. *Journal of machine learning research*, 13(2), 2012.

[8] P. Brazdil, C. G. Carrier, C. Soares, and R. Vilalta. *Metalearning: Applications to data mining*. Springer Science & Business Media, 2008.

[9] V. Cerqueira, L. Torgo, and I. Mozetič. Evaluating time series forecasting models: An empirical study on performance estimation methods. *Machine Learning*, 109(11):1997–2028, 2020.

[10] V. Cerqueira, L. Torgo, and C. Soares. Model selection for time series forecasting: Empirical analysis of different estimators. *arXiv preprint arXiv:2104.00584*, 2021.

[11] B. Chatterjee, D.-H. Seo, S. Chakraborty, S. Avlani, X. Jiang, H. Zhang, M. Abdallah, N. Raghunathan, C. Mousoulis, A. Shakouri, S. Bagchi, D. Peroulis, and S. Sen. Context-aware collaborative intelligence with spatio-temporal in-sensor-analytics for efficient communication in a large-area iot testbed. *IEEE Internet of Things Journal*, 8(8):6800–6814, 2021.

[12] M. Christ, N. Braun, J. Neuffer, and A. W. Kempa-Liehr. Time series feature extraction on basis of scalable hypothesis tests (tsfresh–a python package). *Neurocomputing*, 307:72–77, 2018.

[13] H. I. Fawaz, G. Forestier, J. Weber, L. Idoumghar, and P.-A. Muller. Transfer learning for time series classification. In *2018 IEEE international conference on big data (Big Data)*, pages 1367–1376. IEEE, 2018.

[14] M. Feurer and F. Hutter. Hyperparameter optimization. In *Automated Machine Learning*, pages 3–33. Springer, Cham, 2019.

[15] 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, *Advances in Neural Information Processing Systems*, volume 28. Curran Associates, Inc., 2015.

[16] C. Finn, P. Abbeel, and S. Levine. Model-agnostic meta-learning for fast adaptation of deep networks. In *International Conference on Machine Learning*, pages 1126–1135. PMLR, 2017.

[17] J.-Y. Franceschi, A. Dieuleveut, and M. Jaggi. Unsupervised scalable representation learning for multivariate time series. *arXiv preprint arXiv:1901.10738*, 2019.

[18] H. Hewamalage, C. Bergmeir, and K. Bandara. Recurrent neural networks for time series forecasting: Current status and future directions. *International Journal of Forecasting*, 37(1):388–427, 2021.

[19] A. Hooshmand and R. Sharma. Energy predictive models with limited data using transfer learning. In *Proceedings of the Tenth ACM International Conference on Future Energy Systems*, pages 12–16, 2019.

[20] S. Kadioglu, Y. Malitsky, M. Sellmann, and K. Tierney. Isac-instance-specific algorithm configuration. In *ECAI*, volume 215, pages 751–756. Citeseer, 2010.

[21] Kaggle. Time Series Forecasting Datasets. https://www.kaggle.com/search?q=time+series+forecasting+in%3Adatasets, 2021. [Online; accessed 21-May-2021].

[22] P. S. Kalekar et al. Time series forecasting using holt-winters exponential smoothing. *Kanwal school of information Technology*, 4329008(13):1–13, 2004.

[23] A. Kalousis. *Algorithm selection via meta-learning*. PhD thesis, University of Geneva, 2002.

[24] C. Lemke and B. Gabrys. Meta-learning for time series forecasting and forecast combination. *Neurocomputing*, 73(10-12):2006–2016, 2010.

[25] R. Lewis and G. C. Reinsel. Prediction of multivariate time series by autoregressive model fitting. *Journal of multivariate analysis*, 16(3):393–411, 1985.

[26] L. Li, K. Jamieson, G. DeSalvo, A. Rostamizadeh, and A. Talwalkar. Hyperband: A novel bandit-based approach to hyperparameter optimization. *The Journal of Machine Learning Research*, 18(1):6765–6816, 2017.

[27] A. Liaw, M. Wiener, et al. Classification and regression by randomforest. *R news*, 2(3):18–22, 2002.

[28] P. R. Lorenzo, J. Nalepa, M. Kawulok, L. S. Ramos, and J. R. Pastor. Particle swarm optimization for hyper-parameter selection in deep neural networks. In *Proc. of the Genetic & Evolutionary Computation Conference*, pages 481–488, 2017.

[29] M. Matijaš, J. A. Suykens, and S. Krajcar. Load forecasting using a multivariate meta-learning system. *Expert systems with applications*, 40(11):4427–4437, 2013.

[30] P. Montero-Manso, G. Athanasopoulos, R. J. Hyndman, and T. S. Talagala. Fforma: Feature-based forecast model averaging. *International Journal of Forecasting*, 36(1):86–92, 2020.

[31] J. Narwariya, P. Malhotra, L. Vig, G. Shroff, and T. Vishnu. Meta-learning for few-shot time series classification. In *Proceedings of the 7th ACM IKDD CoDS and 25th COMAD*, pages 28–36. ACM, 2020.

[32] M. Nikolić, F. Marić, and P. Janičić. Simple algorithm portfolio for sat. *Artificial Intelligence Review*, 40(4):457–465, 2013.

[33] B. N. Oreshkin, D. Carpov, N. Chapados, and Y. Bengio. Meta-learning framework with applications to zero-shot time-series forecasting. *arXiv:2002.02887*, 2020.

[34] B. N. Oreshkin, D. Carpov, N. Chapados, and Y. Bengio. N-beats: Neural basis expansion analysis for interpretable time series forecasting. In *International Conference on Learning Representations*, 2020.

[35] Z. Pan, W. Zhang, Y. Liang, W. Zhang, Y. Yu, J. Zhang, and Y. Zheng. Spatio-temporal meta learning for urban traffic prediction. *IEEE Transactions on Knowledge and Data Engineering*, pages 1–1, 2020.

[36] F. Pedregosa, G. Varoquaux, A. Gramfort, V. Michel, B. Thirion, O. Grisel, M. Blondel, P. Prettenhofer, R. Weiss, V. Dubourg, et al. Scikit-learn: Machine learning in python. *the Journal of machine Learning research*, 12:2825–2830, 2011.

[37] A. Poghosyan, A. Harutyunyan, N. Grigoryan, C. Pang, G. Oganesyan, S. Ghazaryan, and N. Hovhannisyan. An enterprise time series forecasting system for cloud applications using transfer learning. *Sensors*, 21(5):1590, 2021.

[38] S. Ravi and H. Larochelle. Optimization as a model for few-shot learning. In *5th International Conference on Learning Representations, ICLR 2017, Toulon, France, April 24-26, 2017, Conference Track Proceedings*. OpenReview.net, 2017.

[39] M. Ribeiro, K. Grolinger, H. F. ElYamany, W. A. Higashino, and M. A. Capretz. Transfer learning with seasonal and trend adjustment for cross-building energy forecasting. *Energy and Buildings*, 165:352–363, 2018.

[40] A. A. Rusu, D. Rao, J. Sygnowski, O. Vinyals, R. Pascanu, S. Osindero, and R. Hadsell. Meta-learning with latent embedding optimization. In *International Conference on Learning Representations*, 2019.

[41] D. Salinas, V. Flunkert, J. Gasthaus, and T. Januschowski. Deepar: Probabilistic forecasting with autoregressive recurrent networks. *International Journal of Forecasting*, 36(3):1181–1191, 2020.

[42] S. Seabold and J. Perktold. Statsmodels: Econometric and statistical modeling with python. In *Proceedings of the 9th Python in Science Conference*, volume 57, page 61. Austin, TX, 2010.

[43] S. Y. Shah, D. Patel, L. Vu, X.-H. Dang, B. Chen, P. Kirchner, H. Samulowitz, D. Wood, G. Bramble, W. M. Gifford, et al. Autoai-ts: Autoai for time series forecasting. In *Proceedings of the 2021 International Conference on Management of Data*, pages 2584–2596, 2021.

[44] B. Shahriari, A. Bouchard-Côté, and N. Freitas. Unbounded bayesian optimization via regularization. In *Artificial intelligence and statistics*, pages 1168–1176. PMLR, 2016.

[45] J. Snell, K. Swersky, and R. Zemel. Prototypical networks for few-shot learning. In I. Guyon, U. V. Luxburg, S. Bengio, H. Wallach, R. Fergus, S. Vishwanathan, and R. Garnett, editors, *Advances in Neural Information Processing Systems*, volume 30. Curran Associates, Inc., 2017.

[46] T. S. Talagala, R. J. Hyndman, G. Athanasopoulos, et al. Meta-learning how to forecast time series. *Monash Econometrics Working Papers*, 6:18, 2018.

[47] T. S. Talagala, F. Li, and Y. Kang. Fformpp: Feature-based forecast model performance prediction. *International Journal of Forecasting*, 2021.

[48] S. J. Taylor and B. Letham. Forecasting at scale. *The American Statistician*, 72(1):37–45, 2018.

[49] E. Vaiciukynas, P. Danenas, V. Kontrimas, and R. Butleris. Meta-learning for time series forecasting ensemble. *arXiv preprint arXiv:2011.10545*, 2020.

[50] J. Vanschoren. Meta-learning: A survey. *arXiv preprint arXiv:1810.03548*, 2018.

[51] X. Wang, K. Smith-Miles, and R. Hyndman. Rule induction for forecasting method selection: Meta-learning the characteristics of univariate time series. *Neurocomputing*, 72(10-12):2581–2594, 2009.

[52] Y. Wang, A. Smola, D. Maddix, J. Gasthaus, D. Foster, and T. Januschowski. Deep factors for forecasting. In *International Conference on Machine Learning*, pages 6607–6617. PMLR, 2019.

[53] T. Wen and R. Keyes. Time series anomaly detection using convolutional neural networks and transfer learning. *arXiv preprint arXiv:1905.13628*, 2019.

[54] M. Wistuba, N. Schilling, and L. Schmidt-Thieme. Scalable gaussian process-based transfer surrogates for hyperparameter optimization. *Machine Learning*, 107(1):43–78, 2018.

[55] W. Yan, H. Qiu, and Y. Xue. Gaussian process for long-term time-series forecasting. In *2009 International Joint Conference on Neural Networks*, pages 3420–3427. IEEE, 2009.

[56] Y. Zhao, R. A. Rossi, and L. Akoglu. Automating outlier detection via meta-learning. *arXiv preprint arXiv:2009.10606*, 2020.

[57] F. Zhou, C. Cao, K. Zhang, G. Trajcevski, T. Zhong, and J. Geng. Meta-gnn: On few-shot node classification in graph meta-learning. In *Proceedings of the 28th ACM International Conference on Information and Knowledge Management*, CIKM '19, page 2357–2360, New York, NY, USA, 2019. Association for Computing Machinery.