Evaluation-Free Time-Series Forecasting Model Selection via Meta-Learning

MUSTAFA ABDALLAH, Indiana University-Purdue University Indianapolis, USA
RYAN A. ROSSI, Adobe Research, USA
KANAK MAHADIK, Adobe Research, USA
SUNGCHUL KIM, Adobe Research, USA
HANDONG ZHAO, Adobe Research, USA
SAURABH BAGCHI, Purdue University, USA

Time-series forecasting models are invariably used in a variety of domains for crucial decision-making. Traditionally these models are constructed by experts with considerable manual effort. Unfortunately, this approach has poor scalability while generating accurate forecasts for new datasets belonging to diverse applications. Without access to skilled domain-knowledge, one approach is to train all the models on the new time-series data and then select the best one. However, this approach is nonviable in practice. 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 the 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. AutoForecast has also significant reduction in inference time compared to the naïve approach (doing inference using all possible models and then selecting the best one), with median of 42X across the two testbeds. We release our meta-learning database corpus (348 datasets), performances of the 322 forecasting models on the database corpus, meta-features, and source codes for the community to access them for forecasting model selection and to build on them with new datasets and models which can help advance automating time-series forecasting problem. In our released database corpus, we unveil new traces of Adobe computing cluster usage for production workloads.

CCS Concepts: • **Computing methodologies → Machine learning**; **Feature selection**;

Additional Key Words and Phrases: Time-series forecasting, Model selection, AutoML, Meta-learning, Inference.

ACM Reference Format:
Mustafa Abdallah, Ryan A. Rossi, Kanak Mahadik, Sungchul Kim, Handong Zhao, and Saurabh Bagchi. 2023. Evaluation-Free Time-Series Forecasting Model Selection via Meta-Learning. *ACM Trans. Knowl. Discov. Data.* 37, 4, Article 1 (January 2023), 40 pages. https://doi.org/10.1145/1122445.1122456

Authors' addresses: Mustafa Abdallah, mabdall@iu.edu, Indiana University-Purdue University Indianapolis, Indianapolis, IN, USA; Ryan A. Rossi, ryrossi@adobe.com, Adobe Research, San Jose, CA, USA; Kanak Mahadik, mahadik@adobe.com, Adobe Research, San Jose, CA, USA; Sungchul Kim, sukim@adobe.com, Adobe Research, San Jose, CA, USA; Handong Zhao, hazhao@adobe.com, Adobe Research, San Jose, CA, USA; Saurabh Bagchi, sbagchi@purdue.edu, Purdue University, West Lafayette, IN, USA.

1556-4681/2023/1-ART1 $15.00
https://doi.org/10.1145/1122445.1122456

1 INTRODUCTION

Accurate time-series forecasting at scale is critical for a wide range of industrial domains such as cloud computing [59], supply chain [1], energy [15], and finance [55]. 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 [8]. 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 [85]. For example, from our experiments (see Figure 3), 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, to 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., [25, 65, 84]. *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 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 4 in Section 5.3). *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 develop 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 (see Table 13 for the full list of our meta-features and

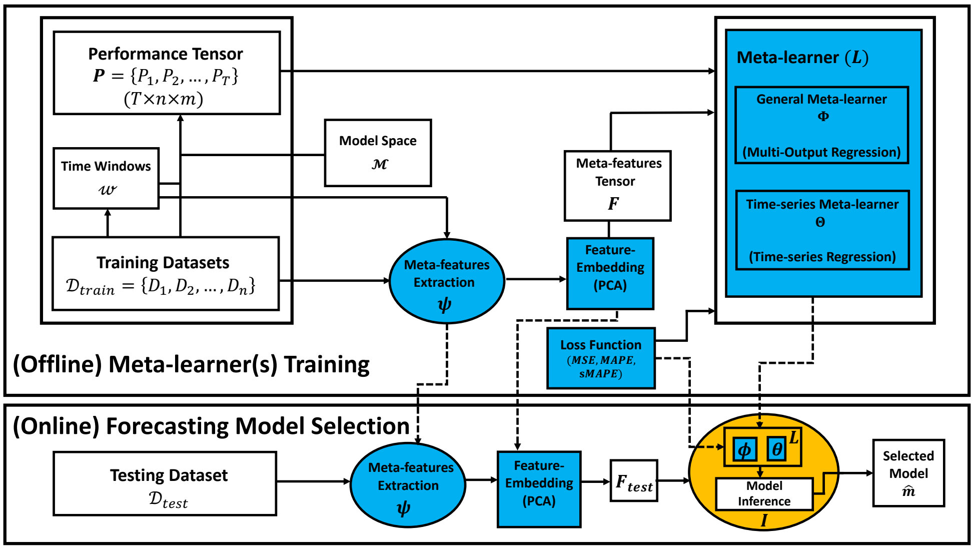

Fig. 1. An overview of AutoForecast; components that transfer from offline to online (model selection) phase are shown in blue. Given the two main inputs, the performance tensor **P** and the meta-features tensor **F**, the meta-learner $\mathcal{L}$ learns two main components: general meta-learner (Φ) 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.

Section 3.4 for categories). 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]. In particular, we are unveiling new traces of Adobe's computing cluster usage for production workloads in this corpus. 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. AutoForecast is an important step towards automating time-series forecasting pipeline.

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 [46, 51, 66, 77, 81] (including DeepAR [66], DeepFactors [81], and Prophet [77]) and different meta-learning approaches [37, 54, 88] (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. This shows the prospect of AutoForecast as a possible solution to the challenging forecasting model selection.

[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.

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.

(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.

Extensions over the Conference Version of the Work: This paper extends the conference version of this work [3] in the following manner:

(1) We perform a detailed data-wise performance evaluation for each dataset in the two studied testbeds (Section 5.3.4). Such evaluation shows the effect of dependency among the dataset on the performance of our meta-learners.

(2) We collect the performances of additional conventional forecasting models (Trigonometric, Box-Cox transform, ARMA errors, Trend, and Seasonal components (TBATS) [20], Holt-Winters [14], Exponential Smoothing [28], and ESRNN [72]) for our meta-learning problem. We also include few AutoML solutions (AutoArima [35], AutoETS [36], and Auto-AI-TS [69]) in our evaluation.

(3) We add several interpretable input meta-feature values for all datasets in our testbed (e.g., mean, median, variance, skeweness, kutosis, absolute energy, and benford correlation) for each of the individual time series in that testbed (Section 5.3.8). Such features values show the diversity in the datasets of our testebds.

(4) We explain in details the generation of our novel landmarker meta-features (Appendix B.3) and provide a full list of the meta-features we used in our work (Table 13). This can help further the replication of our results.

(5) We perform an experiment to evaluate the consistency and stability of AutoForecast against the brute force approach that ranks all forecasting algorithms (Section 5.3.7).

(6) We provide the implementation details of the forecasting models in our model space used to study the meta-learning problem formulated in our current work (Appendix C).

(7) We add the following micro evaluations in our current version

(a) *Tuning of Time-series Meta-learner:* We show the effect of different hyper-parameters used in the training of the time-series meta-learner Θ on the performance of the selected model by Θ (Appendix E.1).

(b) *Time overhead of* AutoForecast *relative to training of selected model:* We show via aggregate statistics that AutoForecast incurs only negligible overhead relative to actual training of the selected model (Appendix E.2).

(c) *Dataset-wise inference time comparison:* We pick several random groups of datasets and show the time (in seconds) that AutoForecast takes versus the naïve approach for forecasting model selection (Appendix E.3).

(8) For better readability of our work, we add the following material
(a) A table summarizing the main notations used in our work (Table 12).
(b) A table summarizing the statistics of our two time-series data testbeds (Table 14).

The rest of the paper is organized as follows. Section 2 presents the related literature. Section 3 introduces the problem formulation. The proposed framework of AutoForecast is presented in Section 4. The experimental setup and experiments on our testbeds is conducted in Section 5. A discussion about the applicability of our work and possible extensions is presented in Section 6. Section 7 concludes our paper.

2 RELATED WORK

2.1 Meta-learning in Time-series Forecasting

There are several works that considered meta-learning for time-series analysis [32, 43, 57, 64, 80]. The work [19] presented 99 rules utilizing 18 time series features to make forecasts for economic and demographic time series. The work [60] introduced the term "meta-learning" in the context of time series model selection. The work [43] explored different meta-learning approaches for time series forecasting where they used ARIMA models, exponential smoothing models, random walk model, and a neural network model as the main forecasting models for performance collection for their meta-learning approaches. The work [83] developed a meta-learning framework for forecast-model selection and reduced time series dimensionality through principal component analysis. The work [42] proposed a neural network-based meta-learning framework for forecast-model selection. Among these, a few works considered simple ranking-based [43, 49] and rule-based [6, 80] meta-learners. The works [32, 64] 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 a few works that have explored model selection problem using model combination [23, 75, 78]. 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 [7, 12, 69] and model selection [13, 76] 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.

There exist several works that have explored model selection problem using ensemble learning [23, 51, 75, 87], parameter-tuning transfer learning [82], and in-sample model selection [61]. In particular, the work [51] (officially known as FFORMA) is the successor of the work [76] (officially known as FFORMS [74]). The main insight leading to developing FFORMA from FFORMS was that model combination is better than model selection. While FFORMS [76] selects a single model, FFORMA learns weights for the models and then combines (or ensembles) the predictions according to these weights. This FFORMA framework earned the second place in the popular M4 competition. However, those works apply only to this specific model class and not for the general model selection case. For example, 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 lines of work, our proposed AutoForecast can select among any (heterogeneous) set of methods. Furthermore, FFORMA considers a base model pool that nowadays seems dated, with only local forecasting methods, and most of them not suitable for series with multiple seasonalities. Recent works have built on FFORMA via using different base model pools and leveraging different meta-learning methods [10, 30]. In particular, the work [30] considered global forecasting methods along with traditional univariate forecasting algorithms. The goal of global forecasting algorithms is to train across different time series, but they have the limitation of lower time efficiency (i.e., they need huge computational time for training and testing).

2.2 Few-shot Learning & Transfer Learning

Few-shot learning has been recently leveraged for automating machine learning pipeline [55, 63, 73, 89]. In particular, the works [63, 73, 89] investigated different problems outside the domain of time-series forecasting. The work [55] 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 [56], 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, 21, 53, 82]. 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.

2.3 Hyperparameter Optimization

Automated hyperparameter optimization (HPO) has received a surge of attention in the machine learning domain in the last decade [22]. In particular, decision-theoretic [9], bandit-based [45], meta-heuristic [47] and Bayesian optimization (BO) techniques [70] 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.

2.4 Meta-learning in ML Pipelines

Meta-learning has recently received significant attention for automating ML pipelines for a variety of different problems outside the domain of time-series forecasting including supervised learning [24, 84], classification and regression [25, 65], unsupervised learning [4], and among other applications [50]. In particular, meta-learning has been leveraged for such automation by designing models for new tasks based on prior experience [62, 79]. These meta-learners are very different from ours and prior works in meta-learning for time series forecasting model selection.

3 PROBLEM FORMULATION

We address the problem of model selection for time-series forecasting via the meta-learning approach. We start by introducing our main meta-learning components, which are historical datasets, forecasting model space, and time windows.

3.1 Meta-learning Components

Our proposed meta-learner AutoForecast depends on:

- **Historical Datasets:** 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 $\boldsymbol{D}_i$.
- **Forecasting Model Space:** 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. We show the details of the implementations of our forecasting models in the model space in Appendix C.
- **Time Windows:** 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).

3.2 Model Design and Performance Tensor

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 $g_i(\cdot) : \mathbb{R}^{n_i \times v_i} \longrightarrow \mathbb{R}^{n_i \times v_i}$ 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 [66]) and that $g_i(\cdot)$ represents optional transformations of the original time-series data (e.g., exponential smoothing [39]; 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 T is the number of the time windows, $\boldsymbol{P}_k = (p_k^{i,j}) \in \mathbb{R}^{n \times m}$ 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 the dataset $\boldsymbol{D}_i$.

Note that $M_j(w_k(\boldsymbol{D}_i))$ denotes the performance of a model $M_j \in \mathcal{M}$ on a time window w_k which is given by the forecasting error (i.e., mean square error "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).

Illustrating Example: Figure 2 illustrates the different components in our Tensor in which each component $\boldsymbol{P}_k$ represents the performance matrix on a time window t_k. For $\boldsymbol{P}_k$, each row represents a different dataset in the training database and each column represents one model in the mode space. Thus, 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 $\boldsymbol{D}_i$. Note that $M_j(w_k(\boldsymbol{D}_i))$ is given by the forecasting error (i.e., mean square error "MSE") of that model on that window.

This motivates us to define our problem formally, as shown below.

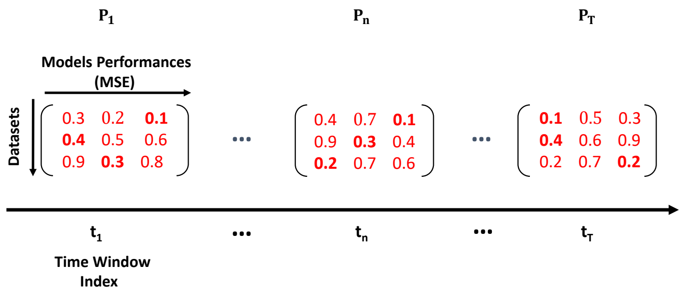

Fig. 2. An illustrating Example of Model Selection and Tensor concept in AutoForecast. The best performance (lowest error) is given in bold text.

3.3 Forecasting Model Selection Problem Statement

We now give the main statement of our time-series forecasting model selection 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*

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

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.

3.4 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 $\mathbf{F}_i = \{\boldsymbol{F}_1^i, \cdots, \boldsymbol{F}_T^i\} \in \mathbb{R}^{T \times d \times v_i}$, where the meta-features matrix $\boldsymbol{F}_k^i \in \mathbb{R}^{d \times v_i}$ denotes the set of the meta features for the time window w_k of the dataset $\boldsymbol{D}_i$, given by

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

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]

The Meta-feature Extractor Ψ: In words, the meta-feature extractor Ψ takes the row data of each time series window w_k, which has length $|w_k|$, as input. Then, it generates all meta-features for that window. For Univariate datasets we have $v_i = 1$, where we have one variable for the time-series. This kind of datasets is the traditional single time-series which usually consists of single variable that need to be predicted. Formally, a time-series dataset $\boldsymbol{D}_i$ is single-variate if $v_i = 1$. For Multivariate Time-series Datasets, we consider two main types (subcategories) of the Multivariate datasets which are (i) *Multivariate Homogeneous dataset* that consists of multiple time-series in which each time-series represents the same metric (*e.g.*,, collection of r time-series

[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.

representing CPU usage of r different machines). In this type, $v_i = r$ (the number of different time-series for that same measurement), (ii) *Multivariate Heterogeneous dataset* in which each time-series column represents a different measurement (*e.g.*,, wind speed, humidity, temperature). In this type, $v_i = |columns(D_i)|$ which is the number of the different variables for the time-series dataset. For the multi-variate dataset, the output of Ψ is a matrix that has d rows and v_i columns.

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 [79]: 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 and Appendix B. In particular, the full list of our meta-features are shown in Table 13.

We summarize our notations in Table 12 (Appendix A).

Having introduced the problem statement and the main components of our meta-learner, we next present our solution framework, AutoForecast.

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 Θ, and general meta-learner Φ. 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 Θ and Φ. The time-series meta-learner Θ 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 Φ 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 Θ while others datasets would have weak temporal dependence among different windows performances in which the general meta-learner Φ is expected to perform better. We show the results of such different datasets for our two testbeds in performance benchmark folder within our anonymized link (provided in Section 1).

General Meta-learner Φ: We propose *multi-output regression model* for training our general meta-learner Φ. 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 $(\boldsymbol{F}_t^i, \boldsymbol{p}_t^i)$, with $t \in [1, T]$ and $i \in [1, n]$.

Thus, the multi-output regression model is given by

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

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_Φ, is given by

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

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 Θ: The goal of the time-series meta-learner Θ 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 $\boldsymbol{F}_1^i, \boldsymbol{F}_2^i, \ldots, \boldsymbol{F}_t^i$ 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

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

where Θ denotes the time-series regression function.

We address such issue of the increasing length of LSTM input in equation 5 via adapting long-short term memory (LSTM) inputs for our time-series meta-learner Θ. In particular, we denote $\boldsymbol{X}_t$ as the input at the time window w_t which is given by

$$\boldsymbol{X}_t = \left[\boldsymbol{F}_1^i, \boldsymbol{p}_1^i, \boldsymbol{F}_2^i, \boldsymbol{p}_2^i, \cdots, \boldsymbol{F}_{t-1}^i, \boldsymbol{p}_{t-1}^i, \boldsymbol{F}_t^i\right]. \tag{6}$$

Note that the length of an LSTM's input (which is $\boldsymbol{X}_t$ in equation 6) will be different depending on the time window index t of such LSTM. For instance, for the LSTM for the first time window w_0 we have $\boldsymbol{X}_0 = \left[\boldsymbol{F}_0^i\right]$. In other words, the input of the LSTM of the first window w_0 is given by the feature vector of that window $\boldsymbol{F}_0^i$. On the other hand, for the second time window w_1 we have

$$\boldsymbol{X}_1 = \left[\boldsymbol{F}_0^i, \boldsymbol{p}_0^i, \boldsymbol{F}_1^i\right].$$

In other words, the input of the LSTM of the second window w_1 is given by the feature vector of that window $\boldsymbol{F}_1^i$, the feature vector of prior window $\boldsymbol{F}_0^i$, and the performance vector of the prior window $\boldsymbol{F}_0^i$. For the later time window (as t advances to T), the input sequence $\boldsymbol{X}_t$ in equation 6 in which both Ft's and pt's keep increasing, and reaching 2*(T-1)+1 at its maximal length.

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 mainly of three layers (the forget gate layer, the input gate layer, and the output gate layer). The activation of those layers is given by

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

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.

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

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

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

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

$$L_{\boldsymbol{\Theta}} = \sum_{t=1}^{T} \sum_{i=1}^{n} L(\hat{\boldsymbol{p}}_t^i, \boldsymbol{p}_t^i). \tag{7}$$

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:

$$\min_{\boldsymbol{\beta}, \boldsymbol{W}_f, \boldsymbol{W}_l, \boldsymbol{W}_o, \boldsymbol{b}_f, \boldsymbol{b}_l, \boldsymbol{b}_o, \boldsymbol{W}_u, \boldsymbol{b}_u, \boldsymbol{W}_v, \boldsymbol{b}_t} \quad aL_{\boldsymbol{\Phi}}(\mathbf{F}, \mathbf{P}) + (1-a)L_{\boldsymbol{\Theta}}(\mathbf{F}, \mathbf{P}), \tag{8}$$

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 7) 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 $\hat{\mathbf{F}}_{test} = \psi(\boldsymbol{D}_{test})$. 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 (by $\mathcal{L}$) error score 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 $w_0, w_1, \ldots, w_T$ of $\boldsymbol{D}_{test}$.

Now, we explain such model selection process for each time window across the time windows $w_0, w_1, \ldots, w_T$ of $\boldsymbol{D}_{test}$ as follows. For the first window (w_0), the inference is given by $\hat{M}_0 \in \arg\min_{\bar{M} \in \mathcal{M}} \mathcal{L}(\boldsymbol{F}_0^{test})$. 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 $\hat{M}_t^{\boldsymbol{\Theta}} \in \arg\min_{\bar{M} \in \mathcal{M}} \boldsymbol{\Theta}(\boldsymbol{F}_0^{test}, \ldots, \boldsymbol{F}_{t-1}^{test}, \boldsymbol{F}_t^{test}, \hat{\boldsymbol{p}}_0^{test}, \ldots, \hat{\boldsymbol{p}}_{t-1}^{test})$. 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 $\hat{M}_t^{\Phi} \in \arg\min_{\bar{M} \in \mathcal{M}} \Phi(F_t^{test})$. Thus, the final selected model is given by

$$\hat{M}_t \in \underset{\bar{M} \in \{\hat{M}_t^{\Phi}, \hat{M}_t^{\Theta}\}}{\arg\min} \; \hat{\boldsymbol{p}}_t^{test}(\bar{M}). \tag{9}$$

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 [11, 40], 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 Φ and that selected by the time-series meta-learner Θ, we choose the model with the best performance (i.e., least predicted error score) (Equation 9).

4.3 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 Φ is $O(d)$. On the other hand, the time complexity of the time-series LSTM meta-learner Θ 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. Notice that we provide quantitative measurements for the difference in inference time between our approach, AutoForecast, and other approaches including naïve method and baseline meta-learners in Section 5.3.5.

4.4 Intuition of Selecting Different Forecasting Models within the Same Dataset

We now provide details about the applicability of our meta-learning approach in which we may select different forecasting models for different time windows within the same dataset. First, we reemphasize that there are temporal variations of the models' performances over different time windows of the same dataset (e.g., Figure 4 in Section 5.3 shows that the best time-series forecasting model for time window w_t is not necessarily the best model for a subsequent time window w_{t+1}). Such point can also be explained due to the fact that each time window of the dataset may have different characteristics (e.g., the time series of the status of a financial company can have such difference in which a healthy condition would be different from financially distressed condition). Therefore, our meta-learning approach capture such variation within the dataset using our time-series meta-learner Θ. On the other hand, sticking to one forecasting model and using it for the entire dataset may lead to higher forecasting error (e.g., we refer to our Average Rank and Hit-at-k accuracy in Section 5.3 where our meta-learner outperforms the performance of Global best and other SOTA forecasting models).

From the user perspective, there are two possible scenarios for implementing our framework on real-world applications which are: (i) The user may not know such low-level best model selection and our algorithm selects the best forecasting model under-the-hood and then generates the required future forecasting for the user, or (ii) The user selects a specific time window in the dataset and then receives from our framework a list which generates a list with the top-k forecasting models along with associated expected error (MSE); generated from our meta-learning inference. The first solution (i) would be more appropriate for high-level users while the second solution (ii) can help scientists in enterprises that have background about different designs of such forecasting models. Note that both solutions can work in real-time (i.e., we refer to our inference time evaluation results in Section 5.3.5).

Table 1. Time-Series Forecasting Model Space. See hyperparameter definitions for various algorithms from GluonTS [5] and statsmodels [68]. The number of models (last column) is all possible combinations of hyperparameters and data representations. It also provides the main hyperparameters tuned for the baseline forecasting algorithms in our evaluation.

Forecasting Algorithm	**HyperParameter 1**	**HyperParameter 2**	**Data Representation**	**Total**
DeepAR (local)	num_cells = [10,20,30,40,50]	num_rnn_layers = [1,2,3,4,5]	{Exp_smoothing, Raw}	50
DeepFactors (local)	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

Baseline Algorithm	**HyperParameter 1**	**HyperParameter 2**	**Data Representation**	**Total**
Auto Arima	n_fits = [10,20,30,40,50]	max_d = [1,2,3,4,5]	{Exp_smoothing, Raw}	50
AutoETS	model = {‘ZMZ’, ‘AZZ’ }	season_length = [1,5,7,10,30]	{Exp_smoothing, Raw}	20
Auto-AI-TS	score_type = [‘rmse’, ’normalized_rmse’]	model_type= [‘best’, ‘prophet’, ‘stats’, ‘ARIMA’, ‘SARIMAX’, ‘VAR’]	{Exp_smoothing, Raw}	24
ESRNN	dilations = [1, 4, 24, 168]	seasonality = [24, 168]	{Exp_smoothing, Raw}	16
Holt-Winters	season_length = [1,2,7,10,30]	error_type = [‘A’,’m’]	{Exp_smoothing, Raw}	20
Exponential_Smoothing	smoothing_level =[0.2,0.5,0.8,1]	smoothing_slope = [0.2,0.5,0.8,1]	{Exp_smoothing, Raw}	32
TBATS	use_box_cox = {False, True}	use_trend= {False, True}	{Exp_smoothing, Raw}	8

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?
(4) How much run-time overhead does AutoForecast incur preceding training of selected model?

5.1 Experimental Setup

Models and Performance Collection: By pairing seven SOTA time-series forecasting algorithms (which are DeepAR [66], Deep Factors [81], Prophet [77], Seasonal Naive [34], Gaussian Process [86], Vector Auto Regression[3] [44], and Random Forest Regressor [46]) 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).[4] 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 [5], Scikit-learn [58], and Statsmodels [68] 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 [26], tsfresh [16] (that we used for generating part of our meta-features).

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 [79]. 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

[3]The Vector Auto Regression was used for Multivariate tesbed while Auto Regression (AR) was used for Univariate testbed.
[4]For ESRNN baseline, we have followed the Pytorch implementation of the ESRNN. After hyper-parameter tuning, the hyperparameters that gave us this performance were as follows: freq_of_test = 1, batch_size = 4, learning_rate = 0.02, per_series_lr_multip = 0.5, lr_decay = 0.5, lr_scheduler_step_size = 7, max_epoch = 5, random_seed = 1, gradient_clipping_threshold = 50, noise_std = 0.001, level_variability_penalty = 30, ensemble = True, seasonality = [24, 168], input_size = 24, output_size = 48, dilations = [1, 4, 24, 168], add_nl_layer=False, cell_type = ‘LSTM’, and state_hsize = 40. We shared the codes (link given in the Introduction) for all baselines in our folder so future works can further tune hyperparameters for different forecasting methods and baseline algorithms and reproduce our results.

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 [79]. 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 enhances 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 [38], 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 (Table 17 in Appendix D) and 40 multivariate datasets with 317 time-series for the second testbed where each multivariate dataset has multiple time-series (e.g., multivariate dataset of Adobe traces have 98 time-series for memory and cpu usages for different production workloads). We refer to our shared datasets for all time-series we used in our work (Table 16 in Appendix D). 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., $|w_t| = 16 \ \forall t \in \{1, \ldots, T\}$).[5] 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 followed standard 5-fold cross validation done in [7]. In particular, we applied blocked cross-validation scheme on the time-series datasets where the datasets are not partitioned randomly, but sequentially into five sets. 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 $\mathbf{\Theta}$ explained in Section 4, we used LSTM with 4 layers where each layer has 50 units. The training was with 50 epochs with

[5] Our choice of such value was related to the "lag parameter", which was suggested to be multiple of four in the "GluonTS" library. However, we set that as a general parameter for the research community in our codes. Moreover, our codes support the case of training using specific window length and testing on another window length.

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 shown in Appendix E.1.

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) [46]**: It 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 [66], DeepFactors [81],[6] and Prophet [77]. 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. We also collected overall performances for Trigonometric, Box-Cox transform, ARMA errors, Trend, and Seasonal components (TBATS) [20], Holt-Winters [14], ESRNN [72], and Exponential Smoothing [28].
  For ESRNN baseline, we have considered two versions of it. The first one which we name "ESRNN (global)" is trained globally on many datasets for each testbed. The second one which we name as "ESRNN (local)" is trained locally for windows of each dataset and tested on the unseen windows.

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 [37]**: This baseline 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) [54]**: It 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 **Θ**.

AutoML solutions: We also compare the performance of AutoForecast with different popular AutoML solutions:

[6]We train DeepAR and DeepFactors in a local manner, i.e., we built a separate model for each window of each series. Recall that our goal is to use all of these baseline time-series forecasting models (with different representations and different hyper-parameters) to design a a meta-learner that learns the models' performances evolution over time windows of the same dataset and across different time series datasets where the meta-learner is designed for different data types with different time dependencies. Thus, in our experimental results we would call them DeepAR (local) and DeepFactors (local).

- **AutoARIMA [35]:** In the basic ARIMA model, we need to provide the p, d, and q values which is time consuming process. In AutoARIMA, the model itself will generate the optimal p, d, and q values which would be suitable for the dataset to provide better forecasting.
- **Automatic Exponential Smoothing (AutoETS) [36]:** This statistical forecasting method represents a variation of the exponential smoothing technique where AutoETS can automatically selects the optimal parameters for the forecasting model.
- **Auto-AI-TS [69]:** the Auto-AI-TS automatically train multiple time series models such as SARIMAX, Prophet, and VAR. Then, it comes up with the best performing model which is suitable for our problem statement. Its performance depends on the size of the dataset.

5.3 Results

5.3.1 Variation of Best Model across Time. Figure 3 shows that no single forecasting model triumphs in more than 0.7% of the datasets. Figure 4 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.

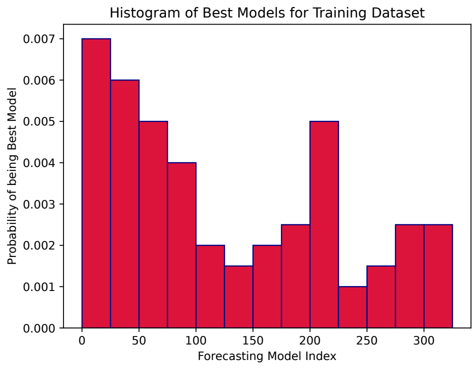

Fig. 3. A histogram of the best forecasting model's probability distribution across datasets of our two testbeds. Different datasets have different best models and no single model triumphs in more than 0.7% of the datasets.

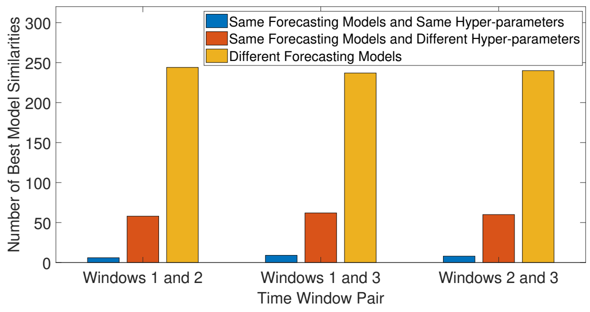

Fig. 4. The aggregate statistics for similarity in the best forecasting model across three consecutive time windows for univariate testbed. Most different time windows have different (best) models.

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

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**

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**

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. Note that with our model selection, some accuracy in selecting best forecasting models is lost to achieve much faster processing at inference time (e.g., see the Hit-at-10 accuracy compared to Hit-at-50 accuracy).

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}). However, there is no significant statistical difference between AutoForecast and FFORMA for both Univariate and Multivariate testbeds (as shown in last row in Table 5).

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 (local), DeepFactors (local), Random Forest, Prophet, Gaussian Process, and VAR, respectively. These results signify the benefits of using meta-learning for model selection, specifically using AutoForecast. We emphasize that one important intuition of our framework is that it generates the meta-features for the test time-series and such meta-features will guide the meta-leaner in doing inference on this test time-series by predicting the best performance via leveraging the performance of different models on similar datasets (that have similar meta-features) on the training.

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

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, autoML, and SOTA methods. For each SOTA, we choose the model with the best average performance from all its model variants.

Method	Univariate Testbed	Multivariate Testbed
Seasonal Naive	0.0345 ± 0.0382	0.0149 ± 0.0408
DeepAR (local)	0.0164 ± 0.0506	0.0085 ± 0.0197
DeepFactors (local)	0.0217 ± 0.0415	0.0135 ± 0.0232
Random Forest	0.0199 ± 0.0398	0.0071 ± 0.0365
Prophet	0.0155 ± 0.0295	0.0065 ± 0.0153
Gaussian Process	0.1661 ± 0.2104	0.2576 ± 0.1344
VAR	0.0602 ± 0.1260	0.9865 ± 0.2988
Global Best	0.0065 ± 0.0199	0.0046 ± 0.0099
AS	0.0158 ± 0.0556	0.0139 ± 0.0563
ISAC	0.0071 ± 0.0145	0.0046 ± 0.0099
MLP	0.0351 ± 0.1186	0.0121 ± 0.2462
AutoArima	0.0563 ± 0.1388	0.0585 ± 0.1340
AutoETS	0.0323 ± 0.0773	0.0273 ± 0.1092
Auto-AI-TS	0.0157 ± 0.0409	0.0211 ± 0.0742
ESRNN (local)	0.0398 ± 0.0444	0.0410 ± 0.0643
ESRNN (global)	0.0401 ± 0.0963	0.0881 ± 0.1319
Holt-Winters	0.0182 ± 0.0436	0.0118 ± 0.0309
Exp. Smoothing	0.0198 ± 0.0417	0.0131 ± 0.0383
TBATS	0.0148 ± 0.1092	0.0118 ± 0.0337
FFORMA	0.0030 ± 0.0116	0.0043 ± 0.0830
AutoForecast-TSL	0.0046 ± 0.0138	0.0054 ± 0.0186
AutoForecast	$\mathbf{0.0026 \pm 0.0090}$	$\mathbf{0.0012 \pm 0.0051}$

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}
AutoForecast	**FFORMA**	0.9868

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}
AutoForecast	**FFORMA**	0.7764

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.

Performance of FFORMA: For FFORMA, we followed the Python implementation of FFORMA in [17]. We emphasize that after producing FFORMA predictions using trained ensemble method, we report the MSE on test data to be consistent with our other experimental metrics. Table 4 shows that the FFORMA method has better performance compared to most other baseline methods.
Performance of ESRNN: Surprisingly, our evaluation results show that ESRNN (local) has lower MSE compared to ESRNN (global). Table 4 shows such a finding in our evaluation where the difference in MSE of ESRNN (local) and ESRNN (global) is higher for the multivariate testbed compared to that for the univariate testbed.
Dataset-wise Performance: We present the detailed performances for each dataset by comparing AutoForecast with all baseline methods in our link (provided in the Introduction as footnote). We note that these results are averaged across the different time windows for each dataset. The results show 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 5 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.

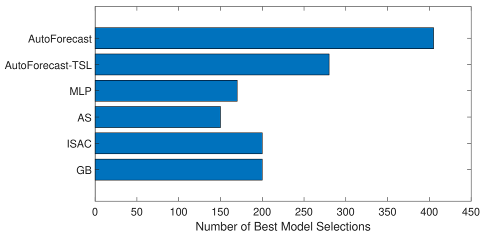

Fig. 5. The number of best model selections by each meta-learning approach. AutoForecast has 2× gain in the selection of best model compared to the closest baselines (ISAC and GB).

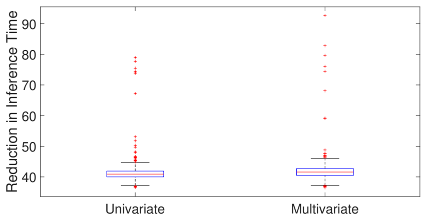

Fig. 6. The inference time reduction of AutoForecast over the naïve approach. AutoForecast gives a median reduction of 42X over naïve approach for both testbeds.

Table 6. Average and standard deviation inference and training runtime performance (in seconds) for both dataset testbeds. AutoForecast select the best model within comparable time to fastest baseline meta-learner.

Phase	Dataset Testbed	Naïve	Global Best	AS	ISAC	MLP	FFORMA	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	16.8927 ± 5.8421	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	11.9714 ± 1.1423	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	428.5880 ± 22.5101	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	252.1106 ± 15.2327	178.1978 ± 18.0098	376.3956 ± 40.0195

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, ISAC, and FFORMA.

Prophet and DeepAR (local) have the best performance across the baseline forecasting algorithms: Table 4 shows that Prophet model has the best average MSE across the baseline forecasting algorithms. Notably, DeepAR (local) has the second best average MSE.

5.3.4 Dataset-wise Performance. We now present the detailed performances for each dataset for both testbeds. It is noted these results are averaged across the different time windows for each dataset. Again, the results show that AutoForecast achieves the best MSE and average rank among all meta-learners.

(a) Univariate Testbed: We now present the full evaluation of univariate testbed. Such evaluation is shown in Table 7 (we show one fold in that table in the interest of space). We note similar performances for the rest of the folds (as reflected in the average MSE and average rank for all datasets shown in Table 3-4). AutoForecast outperforms the meta-learner baselines for most datasets. Moreover, AutoForecast has the lowest average MSE, and the lowest average rank.

(b) Multivariate Testbed: We present the full evaluation of multivariate testbed. Such evaluation is shown in Table 8. It is clear that AutoForecast outperforms the meta-learner baselines for most datasets (best for 28 out of the 40 datasets). Again, we note that AutoForecast has the lowest average MSE, and lowest average rank. In particular, AutoForecast (MSE = 0.12) gives a gain of 50% over the best baseline (MSE = 0.26) on the Adobe_CPU_Mem_15d dataset.

5.3.5 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 6 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 experiment emphasizes that our better model selection performance does *not* entail a prohibitive training cost.

5.3.6 Overall Feature Importance. We next show the feature ranking for different categories of our features. In particular, we show the most important features that affect the performance of our meta-learner for both datasets. Table 9-10 show the top-20 meta-features for Univariate and Multivariate testbeds, respectively. For the Multivariate datasets, the landmarker features, spectral features, and information-theoretic meta-features constitute the top-10 features. For the Univariate datasets, the spectral features, landmarker, statistical, and information-theoretic meta-features constitute the top-10 features. We also observe several common top meta-features across the both testbeds, including Autoregression coefficients (landmarker), Fast Fourier Transform (spectral), linear trend (information-theoretic), lag autocorrelation (statistical). We also observe that the couple simple features do not appear in the top-20 meta-features. Such a result of feature importance can help in identifying the main characteristics that affect meta-learning for our task of model selection by showing most common features that can describe different types of time-series datasets.

5.3.7 Consistency and Stability of AutoForecast*:* We perform an experiment to evaluate the consistency and stability of AutoForecast against the brute force approach. Specifically, in the brute force approach, one can train all models on a given time series, and sort these models' performance as 'rank_BF'. This 'rank_BF' can be considered as the benchmark to compare with the

Table 7. Method evaluation in Univariate testbed (one fold is shown in the interest of space). The most performing (lowest MSE) method is highlighted in **bold**. The rank is provided in parenthesis (lower ranks denote better performance). AutoForecast achieves the best average MSE and average rank among all meta-learners.

Dataset	Global Best	AS	ISAC	MLP	AutoForecast-TSL	AutoForecast
FEATHER.1	0.003429 (4)	0.000916 (3)	0.036634 (6)	0.013679 (5)	**0.000716 (1)**	**0.000716 (1)**
FURNAS.DAT	0.017221 (4)	0.081947 (5)	0.085216 (6)	0.004338 (3)	**0.000177 (1)**	**0.000177 (1)**
EGGS.1	0.0005 (3)	0.000732 (4)	0.006797 (5)	0.02572 (6)	0.000392 (2)	**0.000096 (1)**
NYSE.1	0.041847 (4)	**0.011433 (1)**	0.035967 (2)	0.109728 (6)	0.083661 (5)	0.036996 (3)
apple	0.000071 (3)	0.000727 (5)	0.000256 (4)	0.000842 (6)	**0.000038 (1)**	**0.000038 (1)**
STJOHNS.1	0.015178 (4)	0.152463 (5)	**0.000281 (1)**	0.51043 (6)	0.010065 (3)	0.009741 (2)
SERIESM.2	0.00011 (3)	**0.000031 (1)**	0.000081 (2)	0.000254 (6)	0.000197 (5)	0.000184 (4)
ALIGN.1	0.116045 (5)	**0.020837 (1)**	0.158484 (6)	0.10843 (4)	0.07156 (3)	0.031977 (2)
LYNX.1	0.029034 (3)	0.036677 (4)	0.204602 (5)	0.076905 (6)	0.021614 (2)	**0.012232 (1)**
ARCTIC.1	0.002078 (4)	0.013534 (5)	0.001891 (3)	0.029793 (6)	0.001175 (2)	**0.0006 (1)**
RHINE.1	**0.000992 (1)**	0.030022 (6)	0.026165 (5)	0.011251 (3)	0.013101 (4)	0.008896 (2)
ASKEW15.1	0.002235 (5)	0.00036 (2)	0.001733 (4)	0.084906 (6)	0.000881 (3)	**0.000182 (1)**
MEASLBAL.1	0.000007 (2)	0.012992 (5)	0.000068 (3)	0.073299 (6)	0.00038 (4)	**0.000002 (1)**
REDDEER.1	0.091479 (3)	0.031401 (2)	0.134347 (4)	0.910554 (5)	**0.044033 (1)**	**0.044033 (1)**
rail_lines	**0.000043 (1)**	0.105375 (5)	0.001864 (4)	0.1923 (6)	0.000289 (2)	0.000289 (2)
data_temp_dev	0.00011 (2)	0.000145 (3)	**0.000096 (1)**	0.03142 (6)	0.000335 (4)	0.000335 (4)
DELL.1	0.005872 (3)	0.008785 (4)	0.013155 (5)	0.196153 (6)	**0.00209 (1)**	**0.00209 (1)**
TEMPER.1	0.015079 (4)	0.011044 (3)	0.015079 (5)	0.489135 (6)	**0.006356 (1)**	**0.006356 (1)**
SERIESB.1	0.005424 (3)	0.0321 (6)	0.005424 (3)	0.006143 (5)	**0.000104 (1)**	**0.000104 (1)**
LACSTJRA.1	0.022971 (5)	0.012493 (3)	0.016164 (4)	0.261073 (6)	**0.010394 (1)**	**0.010394 (1)**
Ozone	0.004042 (4)	0.03911 (5)	0.00079 (2)	0.131998 (6)	0.005935 (3)	**0.000706 (1)**
RAPPAHAN.1	0.006714 (3)	0.014144 (5)	0.006714 (3)	0.048011 (6)	**0.001641 (1)**	**0.001641 (1)**
HBCO.1	0.077075 (5)	0.014459 (3)	0.077075 (5)	0.021279 (4)	**0.000486 (1)**	**0.000486 (1)**
CURRENT.1	**0.005603 (1)**	0.007585 (2)	0.031418 (5)	0.050923 (6)	0.010159 (4)	0.00879 (3)
ASKEW7.1	0.118959 (5)	**0.003655 (1)**	0.118959 (5)	0.080627 (4)	0.03322 (3)	0.012513 (2)
SIMAR4.1	0.05864 (4)	0.010436 (3)	0.05864 (4)	0.097847 (6)	**0.000093 (1)**	**0.000093 (1)**
NYWATER.1	**0.000024 (1)**	0.008165 (3)	**0.000024 (1)**	0.108533 (6)	0.021238 (4)	0.021238 (4)
CONSUM.1	0.006701 (3)	0.192148 (5)	0.006701 (3)	0.410414 (6)	0.003062 (2)	**0.001941 (1)**
children_per_woman	**0.000027 (1)**	0.408328 (5)	**0.000027 (1)**	1.65767 (6)	0.013115 (3)	0.013115 (3)
AMERICAN.1	0.099565 (4)	0.006172 (3)	0.099565 (4)	0.27694 (6)	**0.000867 (1)**	**0.000867 (1)**
SERIESJ.2	**0.000311 (1)**	0.021111 (3)	**0.000311 (1)**	0.867712 (6)	0.0531 (4)	0.0531 (4)
FRNCHB.1	0.031783 (4)	0.103844 (2)	0.032671 (4)	0.235725 (6)	0.013224 (3)	**0.008845 (1)**
OTTER_L.1	**0.000697 (1)**	0.154446 (5)	0.003607 (4)	0.228931 (6)	0.001445 (3)	0.001383 (2)
IBM.1	0.099506 (3)	0.007062 (1)	0.029243 (2)	1.192122 (6)	0.181934 (5)	0.150301 (4)
IV.1	0.053671 (4)	0.01878 (3)	0.053671 (4)	0.479534 (6)	**0.000081 (1)**	**0.000081 (1)**
HANKOU.1	0.005295 (4)	**0.000199 (1)**	0.085691 (5)	0.427704 (6)	0.001 (2)	0.001 (2)
ESPANOLA.1	0.107704 (5)	0.063331 (3)	0.107704 (5)	0.069497 (4)	0.038676 (2)	**0.03053 (1)**
NEUMUNAS.1	0.019752 (2)	0.05087 (5)	0.019752 (2)	0.039879 (4)	0.074946 (6)	**0.018099 (1)**
NINEMILE.1	0.034372 (2)	0.150897 (6)	0.034372 (2)	0.038648 (4)	0.084953 (5)	**0.021218 (1)**
NAVAJO.1	0.025011 (3)	0.045677 (4)	0.053738 (5)	0.124633 (6)	**0.005094 (1)**	**0.005094 (1)**
BLUME.1	0.002218 (2)	0.107086 (6)	**0.001929 (1)**	0.049793 (5)	0.003823 (3)	0.003519 (3)
NILE2.1	**0.00137 (1)**	0.001152 (3)	**0.00137 (1)**	0.193211 (6)	0.054431 (5)	0.014202 (4)
LOGISTIC.1	0.080491 (3)	0.200174 (6)	0.080491 (3)	0.16654 (5)	**0.000003 (1)**	**0.000003 (1)**
Y.1	0.010648 (3)	0.00421 (2)	0.002022 (1)	0.722437 (6)	0.065461 (5)	0.061986 (4)
WBDELAWA.1	0.050875 (4)	0.017499 (3)	0.050875 (4)	0.094832 (6)	**0.003033 (1)**	**0.003033 (1)**
AMAZON.2	0.000339 (2)	0.001309 (2)	**0.000156 (1)**	0.219018 (6)	0.051773 (4)	0.051773 (4)
CN.1	0.015912	0.079448	0.012808	0.007689	0.00342	**0.00067 (1)**
ASKEW14.1	0.021909 (4)	**0.000364 (1)**	0.021909 (4)	0.046782 (6)	0.004908 (2)	0.004908 (2)
LACSTJIN.1	**0.000059 (1)**	0.043367 (4)	0.029783 (5)	0.094873 (6)	0.001409 (2)	0.001409 (2)
CD.1	**0.004928 (1)**	0.139645 (5)	0.041752 (4)	0.476946 (6)	0.010701 (2)	0.010701 (2)
FISHERT.1	0.023112 (3)	0.117186 (6)	0.033638 (4)	0.08318 (5)	**0.008069 (1)**	**0.008069 (1)**
RGNP.1	0.000071 (2)	0.034112 (5)	**0.000009 (1)**	0.079975 (6)	0.01095 (4)	0.00114 (3)
AROSA.1	**0.007575042 (1)**	0.010614463 (2)	0.242362866 (5)	0.285212298 (6)	0.01804633 (3)	0.01804633 (3)
U.1	0.005295366 (4)	**0.000199081 (1)**	0.085691031 (5)	0.427703736 (6)	0.000999952 (2)	0.000999952 (2)
RACOON.1	0.001346976 (3)	0.060859373 (5)	0.001346976 (3)	0.293946208 (6)	**0.000630665 (1)**	**0.000630665 (1)**
BND.1	0.00047181	0.111166889	**0.0000889 (1)**	0.043800188	0.002446093	0.000256424 (2)
Average	0.0065 (2.4693)	0.0158 (3.3663)	0.0071 (2.5742)	0.0351 (4.5742)	0.00463 (2.8019)	**0.00256** (2.0571)
STD	0.028074	0.063828	0.043792	0.274294	0.028135	0.020976

Table 8. Method evaluation in multivariate testbed (average MSE). The most performing (lowest MSE) method is highlighted in **bold**. The rank is provided in parenthesis (lower ranks denote better performance). AutoForecast achieves the best average MSE and average rank among all meta-learners. In particular, it has the best performance (lowest average MSE) on 28 datasets out of the 40 multivariate datasets and has comparable performance for remaining datasets.

Dataset	Global Best	AS	ISAC	MLP	AutoForecast-TSL	AutoForecast
Processed_S&P	0.137347 (2)	0.455675 (5)	0.137347 (2)	4.936163 (6)	0.158537 (4)	**0.071178 (1)**
ozone_onehr	0.780519 (5)	0.364615 (3)	0.780519 (5)	0.765257 (4)	0.189151 (2)	**0.042093 (1)**
Occ_Train	**0.000106 (1)**	0.66462 (5)	**0.000106 (1)**	0.717224 6)	0.303732 (4)	0.072843 (3)
Scanline	0.00811 (2)	0.627556 (5)	0.00811 (2)	1.292029 (6)	0.064343 (4)	**0.005702 (1)**
knoy_mpu_3_300	**0.009803 (1)**	0.176418 (6)	**0.009803 (1)**	0.04145 (3)	0.10648 (5)	0.043316 (4)
energydata_complete	0.273006 (3)	0.298254 (5)	0.273006 (3)	3.02911 (6)	0.145331 (2)	**0.065071 (1)**
AdobeAveCPU_96x3270	**0.0005 (1)**	0.097903 (4)	**0.0005 (1)**	5.486816 (6)	0.151596 (5)	0.003049 (3)
Adobe_Service_CPU_Mem_15d	0.264722 (2)	0.813373 (5)	0.264722 (2)	10.782728 (6)	0.581017 (4)	**0.120063 (1)**
knoy_mpu_1_340	0.009342 (4)	0.005052 (3)	0.009342 (4)	0.239649 (6)	0.001921 (2)	**0.000374 (1)**
iowa-electricity	0.003171 (3)	0.025284 (5)	0.003171 (3)	0.966798 (6)	**0.000001 (1)**	**0.000001 (1)**
knoy_mpu_1_400	0.077215 (4)	0.020974 (3)	0.077215 (4)	0.101848 (6)	**0.000009 (1)**	**0.000009 (1)**
knoy_mpu_2_400	0.132552 (4)	**0.005615 (1)**	0.132552 (4)	2.452516 (6)	0.011195 (3)	0.00656 (2)
Occupancy	**0.000097 (1)**	0.056036 (4)	**0.000097 (1)**	2.821039 (6)	0.253453 (5)	0.022591 (3)
ozone_eighthr	0.131161 (2)	0.286562 (5)	0.131161 (2)	4.974266 (6)	0.151916 (4)	**0.090387 (1)**
knoy_mpu_3_400	0.024336 (4)	0.005892 (3)	0.024336 (4)	0.083823 (6)	0.001408 (2)	**0.000618 (1)**
quality_control	0.028198 (4)	0.027566 (3)	0.028198 (4)	4.303693 (6)	0.025724 (2)	**0.006722 (1)**
knoy_mpu_1_500	0.022832 (4)	0.015874 (3)	0.022832 (4)	0.063941 (6)	0.000024 (2)	**0.000005 (1)**
knoy_mpu_3_100	**0.008206 (1)**	0.014191 (3)	**0.008206 (1)**	0.021298 (4)	0.025675 (5)	0.025675 (5)
knoy_mpu_1_360	0.023268 (5)	0.018886 (4)	0.023268 (5)	0.001753 (3)	0.001025 (2)	**0.000034 (1)**
knoy_mpu_2_100	0.005647 (3)	0.009183 (5)	0.005647 (3)	1.82924 (6)	**0.000588 (1)**	**0.000588 (1)**
knoy_mpu_2_500	0.042863 (3)	0.450295 (5)	0.042863 (3)	0.703812 (6)	0.042598 (2)	**0.001608 (1)**
Sales_Transactions	0.068063 (2)	0.122557 (5)	0.068063 (2)	0.57930 (6)	0.074798 (4)	**0.005779 (1)**
co2-concentration	**0.001174 (1)**	0.044724 (5)	**0.001174 (1)**	0.613426 (6)	0.006403 (4)	0.001976 (3)
knoy_mpu_1_100	0.00652 (2)	0.123459 (6)	0.00652 (2)	0.064915 (5)	0.006806 (4)	**0.002683 (1)**
Processed_NASD	0.037896 (2)	0.255123 (5)	0.037896 (2)	0.6712377 (6)	0.117047 (4)	**0.020919 (1)**
knoy_mpu_3_380	0.081679 (3)	0.103497 (5)	0.081679 (3)	0.627697 (6)	0.023772 (2)	**0.005902 (1)**
knoy_mpu_2_320	0.036278 (3)	0.014435 (2)	0.036278 (3)	0.193747 (6)	0.069011 (5)	**0.01057 (1)**
knoy_mpu_2_380	**0.000959 (1)**	0.009084 (4)	**0.000959 (1)**	1.172323 (6)	0.025441 (5)	0.001284 (3)
us-employment	**0.005489 (1)**	0.578479 (5)	**0.005489 (1)**	4.724258 (6)	0.046471 (4)	0.009717 (3)
knoy_mpu_1_380	0.052463 (4)	0.131345 (6)	0.052463 (4)	0.000476 (2)	0.008725 (3)	**0.000013 (1)**
knoy_mpu_2_200	0.021158 (3)	0.021852 (5)	0.021158 (3)	0.692687 (6)	0.002585 (2)	**0.000488** (1)
fast-storage-1	0.113182 (2)	0.69159 (5)	0.113182 (2)	2.765069 (6)	0.130727 (4)	**0.032013 (1)**
fast-storage-20	**0.00008 (1)**	0.347404 (5)	**0.0008 (1)**	2.515765 (6)	0.156555 (4)	0.009153 (3)
knoy_mpu_1_600	0.103141 (5)	0.014413 (3)	0.103141 (5)	0.034072 (4)	0.002585 (2)	**0.000359 (1)**
knoy_mpu_3_200	0.011019 (4)	0.006598 (3)	0.011019 (4)	0.463735 (6)	0.002928 (2)	**0.002038 (1)**
Processed_DJI	**0.002731 (1)**	0.407639 (5)	**0.002731 (1)**	6.420626 (6)	0.110294 (4)	0.027849 (3)
Processed_RUSS	0.109793 (3)	0.159702 (5)	0.109793 (3)	2.493557 (6)	0.072741 (2)	**0.026686 (1)**
knoy_mpu_3_600	0.006719 (3)	0.024021 (6)	0.006719 (3)	**0.000626 (1)**	0.011989 (5)	0.003086 (2)
knoy_mpu_2_600	0.072972 (4)	0.031156 (3)	0.072972 (4)	1.704111 (6)	**0.005216 (1)**	**0.005216 (1)**
Processed_NYSE	0.099705 (3)	0.50898 (5)	0.099705 (3)	2.794788 (6)	0.073811 (2)	**0.029246 (1)**
Average	0.0584 (2.3191)	0.1683 (3.0851)	0.0584 (2.3191)	1.6938 (3.8723)	0.065 (2.3404)	**0.0163 (1.3191)**
STD	0.1252	0.2295	0.1252	2.3421	0.1068	0.0271

one outputted from equation 8. We then use Spearman correlation [52] to measure their consistency. In particular, we computed the Spearman's rank correlation coefficient (ρ) to measure the strength and direction of association between the two ranked arrays that sort the models' performance of Brute Force (using suggested method) and AutoForecast for each time window for every dataset. It basically gives the measure of monotonicity of the relation between two methods i.e. how well the relationship between two methods (here Brute Force and AutoForecast) could be represented using a monotonic function.

For the Univariate testbed, the mean of the Spearman's rank correlation coefficient across all time windows is 0.517059, the median of the Spearman's rank correlation coefficient across all time windows is 0.532343. The maximum of the Spearman's rank correlation coefficient across all time windows is 0.738170. We also emphasize that in 99.687% of the Spearman's rank correlation vector's entries are positive. For the Multivariate testbed, the mean of the Spearman's rank correlation coefficient across all time windows is 0.506604, the median of the Spearman's rank correlation

Table 9. The most important features and their categories for the Univariate testbed.

Name	Category	Rank
Fast Fourier Transform (real, imag)	Spectral	1
Count Above Mean	Statistical	2
Wavelet Transform	Spectral	3
Change Quantiles (mean, var)	Information-theoretic	4-5
Auto Regression Coefficients (coeff no. 2 and coeff no. 7)	Landrmarker	6-7
Lag Autocorrelation (partial, agg)	Statistical	8-9
Fast Fourier Transform (abs, angle)	Spectral	10-11
Entropy (approx entropy)	Information-theoretic	12
Linear Trend (intercept, var)	Information-theoretic	13-14
Range Count Max	Statistical	15
Peaks (number)	Statistical	16
Range Standard Deviation	Statistical	17
Absolute Energy	Information-theoretic	18
Recurrence Statistics (sum)	Statistical	19
Reversal_Asymmetry	Statistical	20

Table 10. The most important features and their categories for the Multivariate testbed.

Name	Category	Rank
Auto Regression Coefficients (coeff no. 1 and coeff no. 7)	Landrmarker	1-2
Fast Fourier Transform (real, imag)	Spectral	3-4
Fourier Entropy (binned)	Spectral	5
Linear Trend (min, var)	Information-theoretic	6-7
Change Quantiles (mean, var)	Information-theoretic	8-9
Wavelet Transform	Spectral	10
Fast Fourier Transform (abs, angle)	Spectral	11-12
Range Standard Deviation	Statistical	13
Absolute Sum of Change (mean)	Information-theoretic	14
Absolute Energy	Information-theoretic	15
Median	Statistical	16
Entropy (approx entropy)	Information-theoretic	17
Lag Autocorrelation	Statistical	18
Recurrence Statistics (sum)	Statistical	19
Kurtosis	Statistical	20

coefficient across all time windows is 0.551903. The maximum of the Spearman's rank correlation coefficient across all time windows is 0.743404. We also emphasize that 98.667% of the Spearman's rank correlation vector's entries are positive. Such results show the consistency and stability of AutoForecast and show how effective AutoForecast is in ranking and selecting best forecasting models in exchange for saving time through the brute force searching. We also emphasize that our aforementioned Hit-at-k accuracy results show the clear superiority of AutoForecast in selecting better forecasting models compared to baseline meta-learners.

5.3.8 Interpretable Features of Datasets. Now, we show some of the interpretable input feature values for all datasets in our testbed. In particular, we show mean (Figure 7), median (Figure 8), variance (Figure 9), skeweness (Figure 10), kutosis (Figure 11), absolute energy (Figure 12), benford correlation (Figure 13), and existence of duplicates (Figure 14) for each of the individual time series in that testbed. Such features are shown in Figures 7-14. Note that time-series datasets have standard normalization before feature extraction. Such features values show the diversity in the datasets, where each dataset has different characteristics. We emphasize that in the interest of space, we show sample of these interpretable features.

5.3.9 Micro Evaluations. We finally perform the following micro evaluations.

(a) **Tuning of Time-series Meta-learner:** We show the effect of different hyper-parameters used in the training of the time-series meta-learner Θ on the performance of the selected model by Θ (see Appendix E.1).

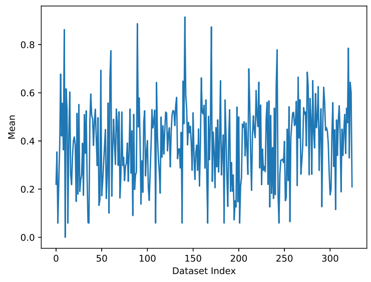

Fig. 7. Mean

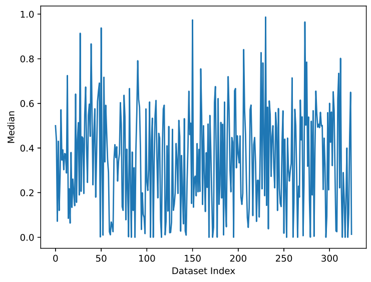

Fig. 8. Median

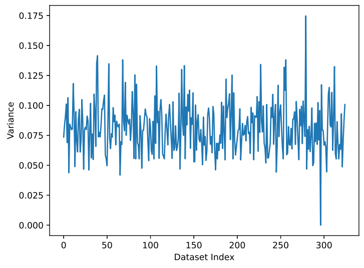

Fig. 9. Variance

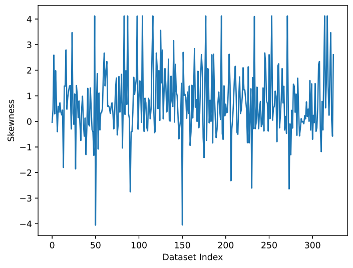

Fig. 10. Skeweness

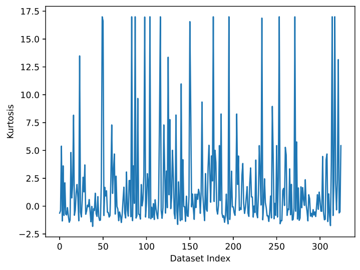

Fig. 11. Kurtosis

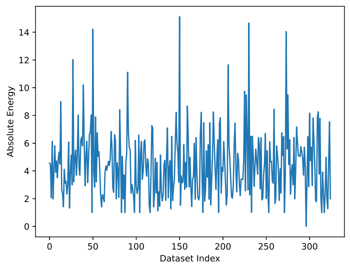

Fig. 12. Absolute Energy

(b) **Time overhead of AutoForecast relative to training of selected model:** we show via aggregate statistics that AutoForecast incurs only negligible overhead relative to actual training of the selected model (see Appendix E.2).

(c) **Dataset-wise inference time comparison:** To further show the superiority of AutoForecast in terms of inference, we pick several random groups of datasets and show the time (in seconds) that AutoForecast takes versus the naïve approach for forecasting model selection (see Appendix E.3).

6 DISCUSSION

6.1 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

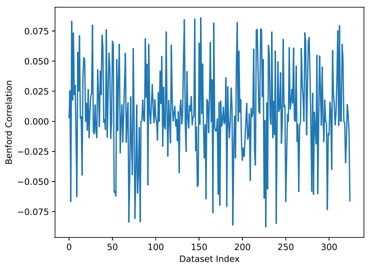

Fig. 13. Benford Correlation

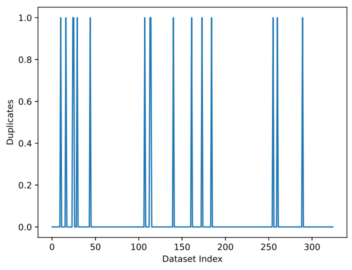

Fig. 14. Existence of duplicates (binary)

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.

6.2 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, [31, 56], e.g., on our univariate testbed, N-Beats [56] 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.

6.3 Diversity of Datasets and Addition of More 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.). In this context, we emphasize that the 625 time series used in the current paper show the potential of AutoForecast in selecting good time forecasting methods efficiently. However, the addition of more datasets would be a crucial step for the future related works to have a benchmark representative sample. For instance, larger collections of time series have been made available in [29]. This collection contains 30 large-scale time-series, including M4, London Smart Meters, Dominick sales, and Web Traffic datasets. We have released our benchmark datasets (link provided earlier in this Section) to help the community build on our work with more datasets and models.

6.4 Performance Collection

For our performance collection, each MSE value represents the difference between the predicted reading (using corresponding forecasting model in our model space) and actual reading (which already exists in the dataset). Moreover, to ensure quality of such values for training our meta-learners, we considered different popular time series forecasting models, and trustworthy toolboxes and tools that are recommended in the related time series forecasting literature. We also share the details of our experimental setups in Evaluation Section and Appendix C and D along with our source codes for replicating our performance generation process. We emphasize that extending the performance collection with more models and hyperparameters (e.g., "multiplicative" seasonality mode in Prophet model) would enhance the performance of our framework. Furthermore, we emphasize that incorporating confidence measures for performance collection can be a fruitful avenue for future research.

6.5 Selection of Meta-Features

We reemphasize that we used tsfresh [16] to generate most of our meta-features. For each time-series dataset, we generated a meta-feature vector that consists of 800 meta-features (810 meta-features). We summarize the meta-features used by AutoForecast in Table 13 in Appendix B. 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. We also emphasize that our meta-feature set has common features with several state-of-the-art feature selection methods. In Table 11, we compare the meta-features introduced in our current study with those from established state-of-the-art time-series feature extraction methods, including tsfresh [16], tsfeatures [33], Compengine [48], and Monash time series forecasting repository [29]. It is worth noting that our proposed meta-features share several common features with each of these leading methods (that have been commonly used for generating meta-features for time-series datasets), with 30 features overlapping with tsfeatures, 15 features with Compengine, and 29 features with Monash. Additionally, we highlight that we have introduced 40 landmark features that have not been previously considered in any of these existing time-series feature selection methods. We acknowledge that further enhancing feature selection and expanding representative meta-features would enhance the performance of our proposed model selection framework in AutoForecast.

6.6 Model Selection vs. Model Averaging

We emphasize that our approach seeks to find the most suitable forecasting method using meta-learning. In particular, the main focus of our work is to automatically identify (select) the suitable forecasting model for an unseen time-series dataset, eliminating the need to initially train (or assess) all models on the new data to determine the most suitable one. This quick inference is the main goal of this work. However, we posit that with our model selection, some accuracy in selecting best forecasting models can be lost to achieve much faster processing at inference time (e.g., see the Hit-at-10 accuracy compared to Hit-at-50 accuracy in Table 2). The main focus of our work is model selection which is different from model averaging in most prior works in forecasting literature. In this context, we emphasize that model averaging usually guarantee better accuracy (i.e., mean square error in our setup). There exist several works that have shown that model averaging is better than model selection approach [51]. The main insight in this line of work is that model combination is better than only selecting a single forecasting method (e.g., the prominent FFORMA model [51] that learns weights for the models and then combines (or ensembles) the predictions according to these weights). Another example in this context is the work [41]. This work has shown that weighted forecast combinations perform better than forecasts selected using information criteria,

Table 11. A comparison between the proposed meta-features in our current work and the meta-features in the established state-of-the-art time-series feature extraction methods, including tsfresh [16], tsfeatures [33], Compengine [48], and Monash time series forecasting repository [29]. We emphasize that our proposed meta-features has several common features with each one of these different state-of-the-art methods (i.e., 30 features common with tsfeatures, 15 features common with Compengine, and 29 features common with Monash). Furthermore, we stress that we have 40 landmarker features (see Appendix B.3 for its details) that have not been considered in any of these prior feature selection methods.

Feature Name (Category)	AUTOFORECAST (Ours)	tsfresh [16]	tsfeatures [33]	Monash [29]	Compengine [27]
Box-Cox lambda values	✗	✗	✗	✓	✗
acf_features (Autocorrelation-based)	✓	✓	✓	✓	✗
Periodicity measure	✓	✓	✗	✓	✓
Longest period of consecutive values above mean	✓	✓	✗	✓	✓
Time intervals between successive outliers	✗	✗	✗	✓	✓
ac_9 (Lag Autocorrelation)	✓	✓	✓	✓	✓
autocorr_features (htsa autocorrelation)	✗	✗	✓	✓	✓
arch_stat (Statistical)	✓	✓	✓	✓	✓
binarize_mean (Count below/above Mean)	✓	✓	✓	✓	✓
pred_feat (prediction)	✓	✓	✓	✓	✓
station_features (stationarity)	✓	✓	✓	✓	✓
dist_feature (distribution)	✗	✗	✓	✓	✓
scal_features (scaling)	✓	✓	✓	✓	✓
crossing_points (no. times the series crosses median)	✓	✓	✓	✓	✗
embed2_incircle (lag of first zero crossing of autocorr.)	✗	✗	✓	✓	✓
entropy (Spectral entropy of a time series)	✓	✓	✓	✓	✓
firstmin_ac (Time of first minimum in the autocorr.)	✓	✓	✓	✓	✓
firstzero_ac (Time of first zero in the autocorrelation)	✓	✓	✓	✓	✓
flat_spot (longest flat spots)	✗	✗	✓	✓	✗
fluctanal_prop_r1 (range and fluctuations)	✓	✓	✓	✓	✓
heterogeneity (Heterogeneity coefficients)	✓	✓	✓	✓	✗
histogram_mode (mode of data vector using histograms)	✗	✗	✓	✓	✓
holt_parameters (holt linear trend)	✓	✓	✓	✓	✓
hurst (difference-based)	✓	✓	✓	✓	✗
Successive differences features	✓	✓	✗	✓	✓
localsimple_taures (zero crossing of residuals)	✓	✓	✓	✓	✗
spike (sudden and significant changes in time series)	✓	✓	✓	✓	✗
lumpiness (mean and variance of tiled windows)	✓	✓	✓	✓	✗
max_level_shift (features of sliding windows)	✓	✓	✓	✓	✗
motiftwo_entro3 (Local motifs)	✗	✗	✓	✓	✗
nonlinearity	✓	✓	✓	✓	✗
outlierinclude_mdrmd (median of outliers)	✗	✗	✓	✓	✓
pacf_features (Partial autocorrelation-based)	✓	✓	✓	✓	✗
sampenc (Second Sample Entropy)	✓	✓	✓	✓	✓
sampen_first (First Sample Entropy)	✓	✓	✓	✓	✓
spreadrandomlocal_meantaul (Bootstrap stationarity)	✗	✗	✓	✓	✗
std1st_der (STD of first derivative)	✓	✓	✓	✓	✗
stl_features (Strength of trend and seasonality)	✓	✓	✓	✓	✗
trev_num (trev autocorrelation)	✓	✓	✓	✓	✗
freq (frequency)	✓	✓	✓	✓	✗
unitroot_kpss (Unit Root Test Statistic)	✓	✓	✓	✓	✗
walker_propcross (walker particle trend)	✗	✗	✓	✓	✗
zero_proportion (Proportion of zero)	✗	✗	✓	✓	✗
Fast Fourier Transform Coeeficients	✓	✓	✗	✓	✓
Symm_looking (Symmetry looking of time-series)	✓	✓	✗	✗	✗
Reversal_Asymmetry (Reversal asymmetry of time-series)	✓	✓	✗	✗	✗
Absolute Summation of Change	✓	✓	✗	✗	✗
Change Quantiles	✓	✓	✗	✗	✗
Friedrich Coefficients (Dynamics)	✓	✓	✗	✗	✗
Wavelet Transform Coefficients	✓	✓	✗	✗	✗
Absolute Energy	✓	✓	✗	✗	✗
Fourier Entropy	✓	✓	✗	✗	✗
(1) Auto Regeression Coefficients	✓	✗	✗	✗	✗
(2) Random Forest Landmarker Features	✓	✗	✗	✗	✗
Number of Leaves	✓	✗	✗	✗	✗
Tree depth	✓	✗	✗	✗	✗
Mean of Base Tree Feature Importance	✓	✗	✗	✗	✗
Max of Base Tree Feature Importance	✓	✗	✗	✗	✗
Out-of-bag estimate score	✓	✗	✗	✗	✗
(3) Bayesian Ridge Regression	✓	✗	✗	✗	✗
Mean of Distribution	✓	✗	✗	✗	✗
Log Marginal Likelihood	✓	✗	✗	✗	✗
Estimated Weights' Precision	✓	✗	✗	✗	✗
Estimated Noise Precision	✓	✗	✗	✗	✗
Stopping Number of Iterations	✓	✗	✗	✗	✗

in terms of both point forecast accuracy and prediction interval coverage. Model averaging would usually be the better choice to achieve the best accuracy possible if the runtime is not an issue. Nonetheless, our choice of model selection in this work instead of model averaging is based on the above mentioned goal of having much faster inference compared to model averaging methods. Our evaluation has provided clear evidence for such gain in inference time (as shown in Table 6). Also, single model selection can potentially provide more interpretability. For example, explaining forecasts of model averaging along with its top features can be a challenging task [67]. In particular, interpreting the predictions from different (possibly correlated) predictors needs caution [18].

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 also included few AutoML solutions (including AutoArima, AutoETS, and Auto-AI-TS) in our evaluation. We performed an experiment to evaluate the consistency and stability of AutoForecast against the brute force approach that rank all forecasting algorithms. 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. Furthermore, we explained in details the generation of our novel landmarker meta-features and provided a full list of the meta-features we used in our work. This can help further the replication of our results. We believe that our work provides an essential stepping-stone in enhancing quick model selection of different time-series forecasting models.

ACKNOWLEDGMENTS

This work is funded in part from Indiana University-Purdue University Indianapolis. The authors thank Adobe Research for providing computational resources for performing part of the experiments (when Mustafa Abdallah was a research intern) and for approving sharing the Adobe traces publicly.

REFERENCES

[1] Hossein Abbasimehr, Mostafa Shabani, and Mohsen Yousefi. 2020. An optimized model using LSTM network for demand forecasting. *Computers & industrial engineering* 143 (2020), 106435.

[2] Mustafa Abdallah, Wo Jae Lee, Nithin Raghunathan, Charilaos Mousoulis, John W Sutherland, and Saurabh Bagchi. 2021. Anomaly Detection through Transfer Learning in Agriculture and Manufacturing IoT Systems. *arXiv preprint arXiv:2102.05814* (2021).

[3] 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 & Knowledge Management (Atlanta, GA, USA) *(CIKM '22)*. Association for Computing Machinery, New York, NY, USA, 5–14. https://doi.org/10.1145/3511808.3557241

[4] Salisu Mamman Abdulrahman, Pavel Brazdil, Jan N. van Rijn, and Joaquin Vanschoren. 2018. Speeding up algorithm selection using average ranking and active testing by introducing runtime. *Mach. Learn.* 107, 1 (2018), 79–108. https://doi.org/10.1007/s10994-017-5687-8

[5] Alexander Alexandrov, Konstantinos Benidis, Michael Bohlke-Schneider, Valentin Flunkert, Jan Gasthaus, Tim Januschowski, Danielle C Maddix, Syama Sundar Rangapuram, David Salinas, Jasper Schulz, et al. 2020. GluonTS: Probabilistic and Neural Time Series Modeling in Python. *J. Mach. Learn. Res.* 21, 116 (2020), 1–6.

[6] Bay Arinze, Seung-Lae Kim, and Murugan Anandarajan. 1997. Combining and selecting forecasting models using rule based induction. *Computers & Operations Research* 24, 5 (1997), 423–433. https://doi.org/10.1016/S0305-0548(96)00062-7

[7] Christoph Bergmeir and José M Benítez. 2012. On the use of cross-validation for time series predictor evaluation. *Information Sciences* 191 (2012), 192–213.

[8] James Bergstra, Rémi Bardenet, Yoshua Bengio, and Balázs Kégl. 2011. Algorithms for hyper-parameter optimization. In *25th annual conference on neural information processing systems (NIPS 2011)*, Vol. 24. Neural Information Processing Systems Foundation.

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

[10] Casper Solheim Bojer and Jens Peder Meldgaard. 2021. Kaggle forecasting competitions: An overlooked learning opportunity. *International Journal of Forecasting* 37, 2 (2021), 587–603.

[11] Pavel Brazdil, Christophe Giraud Carrier, Carlos Soares, and Ricardo Vilalta. 2008. *Metalearning: Applications to data mining*. Springer Science & Business Media.

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

[13] Vitor Cerqueira, Luis Torgo, and Carlos Soares. 2021. Model Selection for Time Series Forecasting: Empirical Analysis of Different Estimators. *arXiv preprint arXiv:2104.00584* (2021).

[14] Chris Chatfield. 1978. The Holt-winters forecasting procedure. *Journal of the Royal Statistical Society: Series C (Applied Statistics)* 27, 3 (1978), 264–279.

[15] Baibhab Chatterjee, Dong-Hyun Seo, Shramana Chakraborty, Shitij Avlani, Xiaofan Jiang, Heng Zhang, Mustafa Abdallah, Nithin Raghunathan, Charilaos Mousoulis, Ali Shakouri, Saurabh Bagchi, Dimitrios Peroulis, and Shreyas Sen. 2021. 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 (2021), 6800–6814. https://doi.org/10.1109/JIOT.2020.3036087

[16] Maximilian Christ, Nils Braun, Julius Neuffer, and Andreas W Kempa-Liehr. 2018. Time series feature extraction on basis of scalable hypothesis tests (tsfresh–a python package). *Neurocomputing* 307 (2018), 72–77.

[17] Christophmark. 2020. Python Implementation of FFORMA. https://github.com/christophmark/fforma

[18] Merlise Clyde. 2003. Model averaging. *Subjective and objective Bayesian statistics* (2003), 636–642.

[19] Fred Collopy and J Scott Armstrong. 1992. Rule-based forecasting: Development and validation of an expert systems approach to combining time series extrapolations. *Management science* 38, 10 (1992), 1394–1414.

[20] Alysha M De Livera, Rob J Hyndman, and Ralph D Snyder. 2011. Forecasting time series with complex seasonal patterns using exponential smoothing. *Journal of the American statistical association* 106, 496 (2011), 1513–1527.

[21] Hassan Ismail Fawaz, Germain Forestier, Jonathan Weber, Lhassane Idoumghar, and Pierre-Alain Muller. 2018. Transfer learning for time series classification. In *2018 IEEE international conference on big data (Big Data)*. IEEE, 1367–1376.

[22] Matthias Feurer and Frank Hutter. 2019. Hyperparameter optimization. In *Automated Machine Learning*. Springer, Cham, 3–33.

[23] Matthias Feurer, Aaron Klein, Katharina Eggensperger, Jost Springenberg, Manuel Blum, and Frank Hutter. 2015. Efficient and Robust Automated Machine Learning. In *Advances in Neural Information Processing Systems*, C. Cortes, N. Lawrence, D. Lee, M. Sugiyama, and R. Garnett (Eds.), Vol. 28. Curran Associates, Inc. https://proceedings.neurips.cc/paper/2015/file/11d0e6287202fced83f79975ec59a3a6-Paper.pdf

[24] Matthias Feurer, Jost Springenberg, and Frank Hutter. 2015. Initializing bayesian hyperparameter optimization via meta-learning. In *Proceedings of the AAAI Conference on Artificial Intelligence*, Vol. 29.

[25] Chelsea Finn, Pieter Abbeel, and Sergey Levine. 2017. Model-agnostic meta-learning for fast adaptation of deep networks. In *International Conference on Machine Learning*. PMLR, 1126–1135.

[26] Jean-Yves Franceschi, Aymeric Dieuleveut, and Martin Jaggi. 2019. Unsupervised scalable representation learning for multivariate time series. *arXiv preprint arXiv:1901.10738* (2019).

[27] Ben D Fulcher, Carl H Lubba, Sarab S Sethi, and Nick S Jones. 2019. CompEngine: a self-organizing, living library of time-series data. *arXiv preprint arXiv:1905.01042* (2019).

[28] Everette S Gardner Jr. 1985. Exponential smoothing: The state of the art. *Journal of forecasting* 4, 1 (1985), 1–28.

[29] Rakshitha Godahewa, Christoph Bergmeir, Geoffrey I. Webb, Rob J. Hyndman, and Pablo Montero-Manso. 2021. Monash Time Series Forecasting Archive. In *Neural Information Processing Systems Track on Datasets and Benchmarks*.

[30] Rakshitha Godahewa, Christoph Bergmeir, Geoffrey I. Webb, and Pablo Montero-Manso. 2023. An accurate and fully-automated ensemble model for weekly time series forecasting. *International Journal of Forecasting* 39, 2 (2023), 641–658. https://doi.org/10.1016/j.ijforecast.2022.01.008

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

[32] Ali Hooshmand and Ratnesh Sharma. 2019. Energy predictive models with limited data using transfer learning. In *Proceedings of the Tenth ACM International Conference on Future Energy Systems*. 12–16.

[33] Rob Hyndman, Yanfei Kang, Pablo Montero-Manso, Thiyanga Talagala, Earo Wang, Yangzhuoran Yang, Mitchell O'Hara-Wild, et al. 2019. tsfeatures: Time series feature extraction. *R package version* 1, 0 (2019).

[34] Rob J Hyndman. 2014. Measuring forecast accuracy. *Business forecasting: Practical problems and solutions* (2014), 177–183.

[35] Rob J Hyndman and Yeasmin Khandakar. 2008. Automatic time series forecasting: the forecast package for R. *Journal of statistical software* 27 (2008), 1–22.

[36] Rob J Hyndman, Anne B Koehler, Ralph D Snyder, and Simone Grose. 2002. A state space framework for automatic forecasting using exponential smoothing methods. *International Journal of forecasting* 18, 3 (2002), 439–454.

[37] Serdar Kadioglu, Yuri Malitsky, Meinolf Sellmann, and Kevin Tierney. 2010. ISAC-Instance-Specific Algorithm Configuration.. In *ECAI*, Vol. 215. Citeseer, 751–756.

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

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

[40] Alexandros Kalousis. 2002. *Algorithm selection via meta-learning*. Ph. D. Dissertation. University of Geneva.

[41] Stephan Kolassa. 2011. Combining exponential smoothing forecasts using Akaike weights. *International Journal of Forecasting* 27, 2 (2011), 238–251.

[42] Mirko Kück, Sven F Crone, and Michael Freitag. 2016. Meta-learning with neural networks and landmarking for forecasting model selection an empirical evaluation of different feature sets applied to industry data. In *2016 international joint conference on neural networks (IJCNN)*. IEEE, 1499–1506.

[43] Christiane Lemke and Bogdan Gabrys. 2010. Meta-learning for time series forecasting and forecast combination. *Neurocomputing* 73, 10-12 (2010), 2006–2016.

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

[45] Lisha Li, Kevin Jamieson, Giulia DeSalvo, Afshin Rostamizadeh, and Ameet Talwalkar. 2017. Hyperband: A novel bandit-based approach to hyperparameter optimization. *The Journal of Machine Learning Research* 18, 1 (2017), 6765–6816.

[46] Andy Liaw, Matthew Wiener, et al. 2002. Classification and regression by randomForest. *R news* 2, 3 (2002), 18–22.

[47] Pablo Ribalta Lorenzo, Jakub Nalepa, Michal Kawulok, Luciano Sanchez Ramos, and José Ranilla Pastor. 2017. Particle swarm optimization for hyper-parameter selection in deep neural networks. In *Proc. of the Genetic & Evolutionary Computation Conference*. 481–488.

[48] Carl H Lubba, Sarab S Sethi, Philip Knaute, Simon R Schultz, Ben D Fulcher, and Nick S Jones. 2019. catch22: CAnonical Time-series CHaracteristics: Selected through highly comparative time-series analysis. *Data Mining and Knowledge Discovery* 33, 6 (2019), 1821–1852.

[49] Marin Matijaš, Johan AK Suykens, and Slavko Krajcar. 2013. Load forecasting using a multivariate meta-learning system. *Expert systems with applications* 40, 11 (2013), 4427–4437.

[50] Gaurav Mittal, Chang Liu, Nikolaos Karianakis, Victor Fragoso, Mei Chen, and Yun Fu. 2020. HyperSTAR: Task-Aware Hyperparameters for Deep Networks. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*. 8736–8745.

[51] Pablo Montero-Manso, George Athanasopoulos, Rob J Hyndman, and Thiyanga S Talagala. 2020. FFORMA: Feature-based forecast model averaging. *International Journal of Forecasting* 36, 1 (2020), 86–92.

[52] Leann Myers and Maria J Sirois. 2004. Spearman correlation coefficients, differences between. *Encyclopedia of statistical sciences* 12 (2004).

[53] Jyoti Narwariya, Pankaj Malhotra, Lovekesh Vig, Gautam Shroff, and TV Vishnu. 2020. Meta-learning for few-shot time series classification. In *Proceedings of the 7th ACM IKDD CoDS and 25th COMAD*. ACM, 28–36.

[54] Mladen Nikolić, Filip Marić, and Predrag Janičić. 2013. Simple algorithm portfolio for SAT. *Artificial Intelligence Review* 40, 4 (2013), 457–465.

[55] Boris N Oreshkin, Dmitri Carpov, Nicolas Chapados, and Yoshua Bengio. 2020. Meta-learning framework with applications to zero-shot time-series forecasting. *arXiv:2002.02887* (2020).

[56] Boris N. Oreshkin, Dmitri Carpov, Nicolas Chapados, and Yoshua Bengio. 2020. N-BEATS: Neural basis expansion analysis for interpretable time series forecasting. In *International Conference on Learning Representations*. https://openreview.net/forum?id=r1ecqn4YwB

[57] Zheyi Pan, Wentao Zhang, Yuxuan Liang, Weinan Zhang, Yong Yu, Junbo Zhang, and Yu Zheng. 2020. Spatio-Temporal Meta Learning for Urban Traffic Prediction. *IEEE Transactions on Knowledge and Data Engineering* (2020), 1–1. https://doi.org/10.1109/TKDE.2020.2995855

[58] Fabian Pedregosa, Gaël Varoquaux, Alexandre Gramfort, Vincent Michel, Bertrand Thirion, Olivier Grisel, Mathieu Blondel, Peter Prettenhofer, Ron Weiss, Vincent Dubourg, et al. 2011. Scikit-learn: Machine learning in Python. *the Journal of machine Learning research* 12 (2011), 2825–2830.

[59] Arnak Poghosyan, Ashot Harutyunyan, Naira Grigoryan, Clement Pang, George Oganesyan, Sirak Ghazaryan, and Narek Hovhannisyan. 2021. An Enterprise Time Series Forecasting System for Cloud Applications Using Transfer Learning. *Sensors* 21, 5 (2021), 1590.

[60] Ricardo BC Prudêncio and Teresa B Ludermir. 2004. Meta-learning approaches to selecting time series models. *Neurocomputing* 61 (2004), 121–137.

[61] Min Qi and Guoqiang Peter Zhang. 2001. An investigation of model selection criteria for neural network time series forecasting. *European Journal of Operational Research* 132, 3 (2001), 666–680.

[62] Aniruddh Raghu, Maithra Raghu, Samy Bengio, and Oriol Vinyals. 2020. Rapid Learning or Feature Reuse? Towards Understanding the Effectiveness of MAML. In *International Conference on Learning Representations*. https://openreview.net/forum?id=rkgMkCEtPB

[63] Sachin Ravi and Hugo Larochelle. 2017. 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. https://openreview.net/forum?id=rJY0-Kcll

[64] Mauro Ribeiro, Katarina Grolinger, Hany F ElYamany, Wilson A Higashino, and Miriam AM Capretz. 2018. Transfer learning with seasonal and trend adjustment for cross-building energy forecasting. *Energy and Buildings* 165 (2018), 352–363.

[65] Andrei A. Rusu, Dushyant Rao, Jakub Sygnowski, Oriol Vinyals, Razvan Pascanu, Simon Osindero, and Raia Hadsell. 2019. Meta-Learning with Latent Embedding Optimization. In *International Conference on Learning Representations*. https://openreview.net/forum?id=BJgklhAcK7

[66] David Salinas, Valentin Flunkert, Jan Gasthaus, and Tim Januschowski. 2020. DeepAR: Probabilistic forecasting with autoregressive recurrent networks. *International Journal of Forecasting* 36, 3 (2020), 1181–1191.

[67] Michael Schomaker and Christian Heumann. 2014. Model selection and model averaging after multiple imputation. *Computational Statistics & Data Analysis* 71 (2014), 758–770.

[68] Skipper Seabold and Josef Perktold. 2010. Statsmodels: Econometric and statistical modeling with python. In *Proceedings of the 9th Python in Science Conference*, Vol. 57. Austin, TX, 61.

[69] Syed Yousaf Shah, Dhaval Patel, Long Vu, Xuan-Hong Dang, Bei Chen, Peter Kirchner, Horst Samulowitz, David Wood, Gregory Bramble, Wesley M Gifford, et al. 2021. AutoAI-TS: AutoAI for Time Series Forecasting. In *Proceedings of the 2021 International Conference on Management of Data*. 2584–2596.

[70] Bobak Shahriari, Alexandre Bouchard-Côté, and Nando Freitas. 2016. Unbounded Bayesian optimization via regularization. In *Artificial intelligence and statistics*. PMLR, 1168–1176.

[71] Qi Shi, Mohamed Abdel-Aty, and Jaeyoung Lee. 2016. A Bayesian ridge regression analysis of congestion's impact on urban expressway safety. *Accident Analysis & Prevention* 88 (2016), 124–137.

[72] Slawek Smyl. 2020. A hybrid method of exponential smoothing and recurrent neural networks for time series forecasting. *International Journal of Forecasting* 36, 1 (2020), 75–85.

[73] Jake Snell, Kevin Swersky, and Richard Zemel. 2017. Prototypical Networks for Few-shot Learning. In *Advances in Neural Information Processing Systems*, I. Guyon, U. V. Luxburg, S. Bengio, H. Wallach, R. Fergus, S. Vishwanathan, and R. Garnett (Eds.), Vol. 30. Curran Associates, Inc. https://proceedings.neurips.cc/paper/2017/file/cb8da6767461f2812ae4290eac7cbc42-Paper.pdf

[74] Thiyanga S Talagala, Rob J Hyndman, and George Athanasopoulos. 2023. Meta-learning how to forecast time series. *Journal of Forecasting* 42, 6 (2023), 1476–1501.

[75] Thiyanga S Talagala, Rob J Hyndman, George Athanasopoulos, et al. 2018. Meta-learning how to forecast time series. *Monash Econometrics Working Papers* 6 (2018), 18.

[76] Thiyanga S. Talagala, Feng Li, and Yanfei Kang. 2021. FFORMPP: Feature-based forecast model performance prediction. *International Journal of Forecasting* (2021). https://doi.org/10.1016/j.ijforecast.2021.07.002

[77] Sean J Taylor and Benjamin Letham. 2018. Forecasting at scale. *The American Statistician* 72, 1 (2018), 37–45.

[78] Evaldas Vaiciukynas, Paulius Danenas, Vilius Kontrimas, and Rimantas Butleris. 2020. Meta-Learning for Time Series Forecasting Ensemble. *arXiv preprint arXiv:2011.10545* (2020).

[79] Joaquin Vanschoren. 2018. Meta-learning: A survey. *arXiv preprint arXiv:1810.03548* (2018).

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

[81] Yuyang Wang, Alex Smola, Danielle Maddix, Jan Gasthaus, Dean Foster, and Tim Januschowski. 2019. Deep factors for forecasting. In *International Conference on Machine Learning*. PMLR, 6607–6617.

[82] Tailai Wen and Roy Keyes. 2019. Time series anomaly detection using convolutional neural networks and transfer learning. *arXiv preprint arXiv:1905.13628* (2019).

[83] Agus Widodo and Indra Budi. 2013. Model selection using dimensionality reduction of time series characteristics. In *International Symposium on Forecasting, Seoul, South Korea*. 57–118.

[84] Martin Wistuba, Nicolas Schilling, and Lars Schmidt-Thieme. 2018. Scalable gaussian process-based transfer surrogates for hyperparameter optimization. *Machine Learning* 107, 1 (2018), 43–78.

[85] David H Wolpert. 1996. The lack of a priori distinctions between learning algorithms. *Neural computation* 8, 7 (1996), 1341–1390.

[86] Weizhong Yan, Hai Qiu, and Ya Xue. 2009. Gaussian process for long-term time-series forecasting. In *2009 International Joint Conference on Neural Networks*. IEEE, 3420–3427.

[87] Rui Ye and Qun Dai. 2018. A novel transfer learning framework for time series forecasting. *Knowledge-Based Systems* 156 (2018), 74–99.

[88] Yue Zhao, Ryan A Rossi, and Leman Akoglu. 2020. Automating outlier detection via meta-learning. *arXiv preprint arXiv:2009.10606* (2020).

[89] Fan Zhou, Chengtai Cao, Kunpeng Zhang, Goce Trajcevski, Ting Zhong, and Ji Geng. 2019. Meta-GNN: On Few-Shot Node Classification in Graph Meta-Learning. In *Proceedings of the 28th ACM International Conference on Information and Knowledge Management* (Beijing, China) *(CIKM '19)*. Association for Computing Machinery, New York, NY, USA, 2357–2360. https://doi.org/10.1145/3357384.3358106

A SUMMARY OF NOTATION

We summarize the notation used in this paper in Table 12.

Table 12. A summary of our notation.

Symbol	Description
$\mathcal{D}_{train}$	Meta-train database
n	Meta-train database size
n_i	No. of observations in $\boldsymbol{D}_i$
v_i	No. of variables in $\boldsymbol{D}_i$
$\mathcal{M}$	Model space
M_j	A forecasting model
m	Size of the model space
$\boldsymbol{h}_i$	Hyperparameter vector of a_i
T	Number of time windows
$\lvert w_i \rvert$	Window length
$\mathbf{P}$	Performance tensor
$\boldsymbol{P}_k$	Performance matrix of window w_k
$p_k^{i,j}$	Model M_j's performance on time window w_k of $\boldsymbol{D}_i$
$\boldsymbol{p}_k^i$	Models performances vector on time window w_k for $\boldsymbol{D}_i$
$g_i(\cdot)$	Data representation
$\mathbf{F}_i$	Meta-features tensor of $\boldsymbol{D}_i$
$\boldsymbol{F}_k^i$	Meta-feature matrix of window w_k in $\boldsymbol{D}_i$
$\psi(\cdot)$	Meta-features extraction module
d	Number of meta-features
$\mathcal{L}$	Meta-learner
$\boldsymbol{\Phi}$	General Meta-learner
$\boldsymbol{\Theta}$	Time-series Meta-learner
$\hat{M}_t^{\Theta}$, $\hat{M}_t^{\Phi}$	Selected model for time window w_t by Θ and Φ, respectively
$\hat{M}_t$	Selected model for window w_t in inference by AutoForecast

B TIME-SERIES META FEATURES

There are prior works that generated standard time-series features [26], tsfresh [16] (that we used for generating part of our meta-features). We now provide details of our meta-features (shared with our database in the link provided in Section 1). For each dataset, we generate a meta-feature vector that consists of more than 800 meta-features.

B.1 Meta-features Categories

For each dataset, we generate a meta-feature vector that consists of more than 800 meta-features. This meta-features vector includes different components that capture various properties of the time-series dataset, *e.g.*,, the statistical features (number of crossings, count of observations above/below the mean value, quantiles, etc.), the data trend (linear, non-linear, dynamics, etc.), data interdependence (lag autocorrelation, difference features, etc.), information-theoretic features that are typically based on entropy measures, and the frequency (FFT, wavelet, etc.). 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.

B.2 Complete List of Features

We summarize the meta-features used by AutoForecast in Table 13. When applicable, we provide the formula for computing the meta-feature(s) and corresponding variants we used for generating the meta-features, and the corresponding number of features. Some are based on [79]. 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 [79]. To the best of our knowledge, we emphasize that our landmarker meta-features (detailed below) are novel and that some components of the spectral meta-features have not been used in any related work.

B.3 Landmarker Meta-Feature Generation

In addition to simple, statistical, and information-theoretic meta-features, we use three time-series forecasting landmarker algorithms for computing forecasting-specific landmarker meta-features, which are Auto Regression [44], Random Forest [46], and Bayesian Ridge Regression [71] to capture landmarker characteristics of a time-series dataset.

We now provide a quick overview of each algorithm and then discuss how we are using them for building meta-features. The algorithms are executed with the default parameter.

Auto Regression [44]: In this landamarker feature, we fits the unconditional maximum likelihood of an autoregressive process. The k parameter represents the maximum lag of the process

$$X_t = \varphi_0 + \sum_{i=1}^{k} \varphi_i X_{t-i} + \varepsilon_t$$

Then, we extract the coefficients φ_i whose index $i \in \{0, \cdots, k\}$.

Random Forest [46]: It is a tree-based ensemble method that builds a collection of base trees using the sub-sampled unlabeled data, splitting on (randomly selected) features as nodes. Random

Forest grows internal nodes until the terminal leaves contain only one sample or the predefined max depth is reached.

For Random Forest, we use the balance of base trees (i.e., depth of trees and number of leaves per tree) and additional information (e.g., feature importance of each base tree). It is noted that feature importance information is available for each base tree—we therefore analyze the statistic of mean and max of base tree feature importance. The following information of base trees are used:

- Tree depth: min, max, mean, std, skewness, and kurtosis
- Number of leaves: min, max, mean, std, skewness, and kurtosis
- Mean of base tree feature importance: min, max, mean, std, skewness, and kurtosis
- Max of base tree feature importance: min, max, mean, std, skewness, and kurtosis
- Out-of-bag estimate score.

Bayesian Ridge Regression [71]: It estimates a Gaussian probabilistic model of the regression problem using Bayesian regression. The prior for the coefficient is given by a spherical Gaussian distribution.

For Bayesian Ridge Regression, we use the following information of the probabilistic model (fitted using the dataset) as landmarker features:

- Mean of the distribution
- Log marginal Likelihood score
- The precision of the estimated weights
- The noise precision
- The actual number of iterations to achieve the stopping criterion

C MODEL SPACE

We now provide details on the model space used to study the meta-learning problem formulated in our work. Recall that a model in the context of our problem is a time-series forecasting algorithm and the hyperparameters used.

DeepAR [66]: DeepAR experiments are using the model implementation provided by GluonTS version 1.7. We did grid search on different values of number of cells and the number of RNN layers hyperparameters of DeepAR since the defaults provided in GluonTS would often lead to apparently suboptimal performance on many of the datasets. The training parameters for each dataset are described in Table 1. All other parameters are defaults of gluonts.model.deepar.DeepAREstimator.

Deep Factors [81]: Deep Factors experiments are using the model implementation provided by GluonTS version 1.7. We did grid search over the number of units per hidden layer for the global RNN model and the number of global factors hyperparameters of Deep Factors. The training parameters for each dataset are described in Table 1. All other parameters are defaults of gluonts.model.deep_factor.DeepFactorEstimator.

Prophet [77]: Prophet experiments are using the model python implementation provided by Facebook (fbprophet) version 0.7.1. We did grid search over the change point prior scale and the seasonality prior scale hyperparameters of Prophet. The training parameters for each dataset are described in Table 1. All other parameters are defaults of fbprophet.Prophet.

Seasonal Naive [34]: Seasonal Naive experiments are using the model implementation provided by GluonTS version 1.7. We did grid search over the length of seasonality pattern, since it is different unknown for each, dataset hyperparameter of Seasonal Naive. The training parameters for each dataset are described in Table 1. All other parameters are defaults of gluonts.model.seasonal_naive.SeasonalNaivePredictor.

Gaussian Process [86]: Gaussian Process experiments are using the model implementation provided by GluonTS version 1.7. We did grid search over the cardinality of the time-series

Table 13. Time-series meta-features for characterizing an arbitrary time-series dataset. We extracts a comprehensive number of meta-features. The extracted meta-features are five categories: simple, statistical, information-theoretic meta-features, spectral, and landmarker meta-features. To the best of our knowledge, we emphasize that our landmarker meta-features are novel and that some components of spectral meta-features have not been used in any related work.

Name	Formula	Property	Variants	No. of Features
Window Length	$\lvert w_i \rvert$	Speed/Scalability	N/A	1
Number of Time-series variables	v_i	Type	N/A	1
Lag Autocorrelation	ρ_n	Feature Interdependence	$\rho_0 - \rho_9$	10
Lag Partial Autocorrelation	α_k	Feature Interdependence	acf_agg	15
Standard Deviation Range	$\sigma > r(max - min)$	Dispersion	$r \in [0.05, 0.95]$	20
Maximum	max_X	Data Range	max_duplicates	2
Minimum	min_X	Data Range	min_duplicates	2
Peaks	$peak_X$	Data Range	peaks_supports, no_peaks	6
Reccurence Statistics	$\overline{X}$	Data Range	recc_sum,recc_count,recc_ratio	5
Median	μ	Concentration	rms	2
Mean	$\bar{X}$	Concentration	rms	2
Variance	σ^2	Dispersion	std_dev	2
Covariance	Cov	Dispersion	benford_corr	8
Quantiles	$q_{0.1} - q_{0.9}$	Dispersion	diff_quantiles	10
Mass Quantiles	$q\%_{0.1} - q\%_{0.9}$	Dispersion	diff_quantiles	10
Count below Mean	$\sum \nVdash (X < \mu)$	Statistics	min,max,σ,μ	5
Count Above Mean	$\sum \nVdash (X > \mu)$	Statistics	min,max,σ,μ	5
Number of Crossings	$\sum_{i=1}^{n} 1_{[X==val]}$	Statistics	zero_cross, minus_cross, root_hyp_test	5
Kurtosis	$\frac{\mu_4}{\sigma^4}$	Feature normality	sample_kurt	2
Skewness	Skw	Feature normality	sample_skew	2
Symm_looking	$\lvert \mu_X - median_X \rvert < r * (max_X - min_X)$	Symmetry	$r \in [0.05, 0.95]$	12
Reversal_Asymmetry	$\mathbb{E}[L^2(X)^2 \cdot L(X) - L(X) \cdot X^2]$	Reversal Asymmetry	lag_L	12
Absolute Sum of Change	$\sum_{i=1}^{n} \lvert x_{i+1} - x_i \rvert$	Difference	mean_chg, mean_abs_chg, cid	5
Change Quantiles	Corridor Quantiles	Difference	$q_l \in [0, 1], q_h \in [q_l, 1]$	60
Entropy	D	Regularity	approx_entropy, sample_entropy	21
Linear Trend	Linear_reg_chunks	Trend	{“pvalue”, “rvalue”, “intercept”, “slope”, “stderr”}	50
Non-Linearity	c_3	Non-linearity	$\ell \in \{1, 2, 3\}$, matrix_profile_feat	9
Friedrich Coefficients	$x' = h(x)$	Dynamics	Langevin_Coeffs	5
Wavelet Transform Coefficients	Mexican hat wavelet	Time-Freq	“widths”, “coeffs”	60
Fast Fourier Transform Coeeficients	FFT(X)	Frequency	{“real”, “imag”, ’agg_metrics’}	201
Polar Fast Fourier Transform Coeeficients	FFT(X)	Frequency	{“abs”, “angle”, ’agg_metrics’}	201
Absolute Energy	E	Spectral	Energy_ratio_chunks, cross_pw_spect_dens	15
Fourier Entropy	D_F	Spectral Regularity	binned_entropy	1
(1) Auto Regeression				
Regression Coefficients	See Appendix B	Landmarker	$\ell \in \{1, \cdots, 10\}$	10
(2) Random Forest				
Number of Leaves	See Appendix B	Landmarker	max, min, mean, std, skew, kurtosis	6
Tree depth	See Appendix B	Landmarker	max, min, mean, std, skew, and kurtosis	6
Mean of Base Tree Feature Importance	See Appendix B	Landmarker	max, min, mean, std, skew, and kurtosis	6
Max of Base Tree Feature Importance	See Appendix B	Landmarker	max, min, mean, std, skew, and kurtosis	6
Out-of-bag estimate score	See Appendix B	Landmarker	N/A	1
(3) Bayesian Ridge Regression				
Mean of Distribution	See Appendix B	Landmarker	N/A	1
Log Marginal Likelihood	See Appendix B	Landmarker	N/A	1
Estimated Weights’ Precision	See Appendix B	Landmarker	N/A	1
Estimated Noise Precision	See Appendix B	Landmarker	N/A	1
Stopping Number of Iterations	See Appendix B	Landmarker	N/A	1
				810

and the maximum number of iterations for jitter to iteratively make the matrix positive definite hyperparameter of Gaussian Proces. The training parameters for each dataset are described in Table 1. All other parameters are defaults of gluonts.model.gp_forecaster.GaussianProcessEstimator.
Vector Auto Regression [44]: Vector Auto Regression experiments are using the model implementation provided by statsmodels python library version 0.12.2. We did grid search over the loss covariance type and the trend hyperparameter of Vector Auto Regression. The training parameters for each dataset are described in Table 1. All other parameters are defaults of statsmodels.tsa.var_model.
Random Forest [46]: Random Forest models’ experiments are using the model implementation provided by sklearn python library version 0.24.2. We did grid search over the the number of estimators (trees) and the max_depth (i.e., the longest path between the root node and the leaf node in a tree) hyperparameter of Random Forest. The training parameters for each dataset are described in Table 1. Other parameters are defaults of sklearn.ensemble.RandomForestRegressor.
Exponential Smoothing Data Representation [39]: We use the exponential smoothing from the statsmodels python library version 0.12.2. We optimized the smoothing level for the exponential

smoothing (which controls the weights given for the history samples). All other parameters are defaults of statsmodels.tsa.api.ExponentialSmoothing.

D DATASET TESTBEDS DESCRIPTION

D.1 Dataset Sources

Our testbeds are built to simulate the case when meta-train comes from many different distributions. This diversity makes the meta-learning model learn from such diversity. Model selection on test data can thus benefit from the prior experience on the train set. For this purpose, we use benchmark datasets from Kaggle [38], Adobe real traces, and other open source repositories. In particular, 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.

D.2 Testbeds Summary

In short, we have collected 308 univariate datasets from such different sources. The details of the datasets (i.e., the dataset name, and the number of points in the dataset) are shown in Table 17. We also collected 40 multivariate datasets, where each dataset can have different number of variables and different type. For instance, Adobe service CPU and memory 15 days utilization is a Homogeneous multivariate dataset that has $r = 100$. Most of the datasets are from different application domains (e.g., finance, IoT, energy, storage, etc.). The details of the datasets (i.e., the dataset name, the variate name, and the number of points for each variate the dataset) are detailed in Table 16. For robustness, for each testbed, we split its datasets 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. We released the datastes for the community for future usage (link is in Section 1). Table 14 provides a short summary of our dataset testebds. Table 15 provides their main applications.

Table 14. Time-series dataset corpus statistics and properties.

# time-series datasets $\|\mathcal{D}\|$	# overall time-series (across all datasets)	# of multivariate time-series datasets	# of univariate time-series datasets
348	625	40	308

Table 15. Time-series dataset corpus applications.

Application	**# Univariate time-series datasets**	**# Multiivariate time-series datasets**
IoT	10	16
Finance	28	6
Sales	16	2
Cloud Computing	10	2
Energy	20	5
Quality Control	10	3
Fast Storage	22	2
Environment	36	2
Employment	6	1
Others	150	1

E EXTENDED EVALUATION

We next turn our attention to the tuning of the time-series meta-learner and the timing of AutoForecast, respectively. In particular, we show both the computational cost and the inference time for AutoForecast and compare it with our different baselines.

E.1 Tuning of Time-series Meta-learner Θ

We show the effect of different hyper-parameters used in the training of the time-series meta-learner Θ on the performance (in terms of MSE under the selected model by Θ). For searching on each parameter, we fix the other parameters on their best values. In the interest of space, we show full details for the univariate testbed. Figure 15 shows the effect of the number of the LSTM layers of the time-series meta-learner. We observe that LSTM with 4 layers gives the best performance (lowest MSE). Second, Figure 16 shows the effect of the number of the units in the LSTM layer of the time-series meta-learner. We observe that LSTM with 50 units per layer gives the best performance (lowest MSE). Third, Figure 17 shows the effect of the training batch size of the time-series meta-learner. A batch with size = 25 instances gives the best performance (lowest MSE). Finally, Figure 18 illustrates the effect of the number of training epochs of the time-series meta-learner. We note that training with 50 epochs gives the best performance (lowest MSE). We have also used dropout rate of 0.2 to prevent over-fitting.

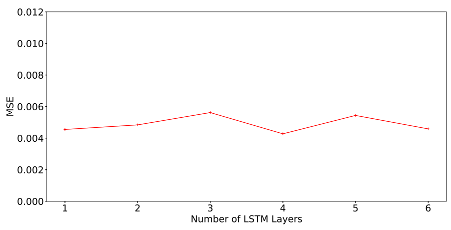

Fig. 15. The performance of the time-series meta-learner Θ vs. number of the LSTM layers of Θ for the univariate testbed. LSTM with 4 layers gives the best performance (lowest MSE).

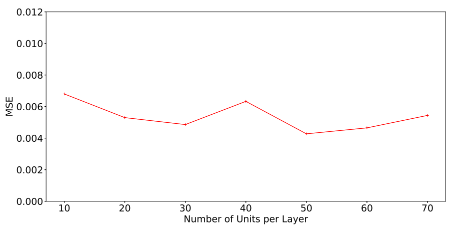

Fig. 16. The performance of the time-series meta-learner Θ vs. number of units per layer of Θ for the univariate testbed. LSTM with 50 units per layer gives the best performance.

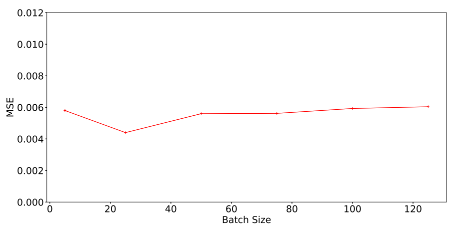

Fig. 17. The performance of the time-series meta-learner Θ vs. the training batch size of Θ for the univariate testbed. A batch with size = 25 instances gives the best performance.

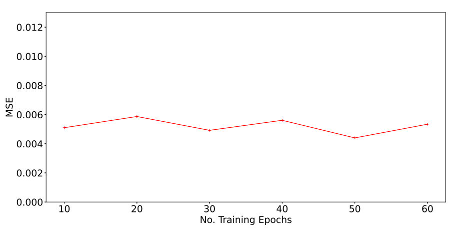

Fig. 18. The performance of the time-series meta-learner Θ vs. number of training epochs of Θ for the univariate testbed. Training with 50 epochs gives the best performance.

E.2 Time Overhead of AutoForecast Relative to Training of Selected Model

Figure 19 shows that it incurs only negligible overhead relative to actual training of the selected model (with median = 0.1%). Similarly, AutoForecast incurs only negligible overhead relative to actual training of the selected model for the multivariate testbed (with median = 0.3%) (Figure is omitted for multivariate testbed). This shows that AutoForecast is lightweight, incurring small selection time overhead.

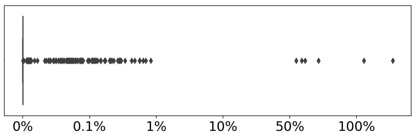

Fig. 19. Boxplot of time AutoForecast takes relative to training of selected model in univariate testbed. AutoForecast incurs negligible overhead, (median = 0.1%).

E.3 Dataset-wise Inference Time Comparison

Next, we pick several random groups of 10 datasets each and show the time (in seconds) that AutoForecast takes versus the time the naïve approach takes for forecasting model selection. Figures 20-23 show such comparison for the univariate testbed. It is noted the higher reduction of inference time for larger datasets (i.e., with more data points).

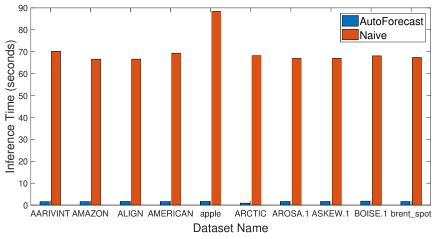

Fig. 20. Inference time comparison between AutoForecast and naïve approach (Group 1).

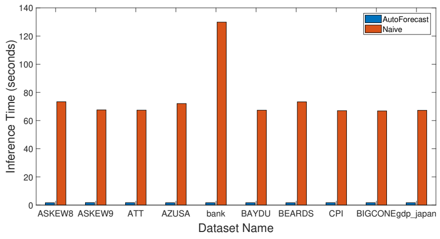

Fig. 21. Inference time comparison between AutoForecast and naïve approach (Group 2).

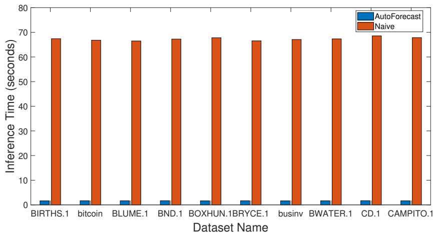

Fig. 22. Inference time comparison between AutoForecast and naïve approach (Group 3).

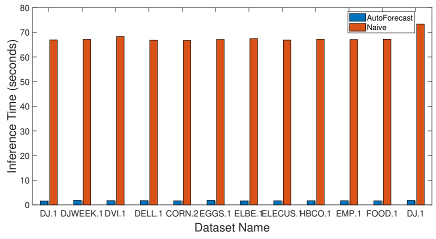

Fig. 23. Inference time comparison between AutoForecast and naïve approach (Group 4).

Table 16. Multivariate Time-series dataset corpus description and details (i.e., the dataset name, the variate name, and the number of points for each variate of the dataset).

Dataset Name	Variable Name	# pts	Dataset Name	Variable Name	# pts	Dataset Name	Variable Name	# pts
Processed_S&P	Oil	1984	Processed_S&P	Nikkei-F	1984	Processed_S&P	ROC_5	1984
Processed_S&P	NASDAQ-F	1984	Processed_S&P	mom1	1984	Processed_S&P	mom3	1984
Processed_S&P	mom2	1984	Processed_S&P	MSFT	1984	Processed_S&P	NZD	1984
Processed_S&P	NYSE	1984	Processed_S&P	ROC_10	1984			
ozone_onehr	T9	2536	ozone_onehr	T8	2536	ozone_onehr	T1	2536
ozone_onehr	T3	2536	ozone_onehr	T2	2536	ozone_onehr	T6	2536
ozone_onehr	T7	2536	ozone_onehr	T5	2536	ozone_onehr	T4	2536
ozone_onehr	T12	2536	ozone_onehr	T11	2536	ozone_onehr	T10	2536
energydata_complete	RH_1	19735	energydata_complete	Press_mm_hg	19735	energydata_complete	lights	19735
energydata_complete	Appliances	19735	energydata_complete	RH_2	19735	energydata_complete	RH_3	19735
energydata_complete	RH_1	19735	energydata_complete	Visibility	19735	energydata_complete	Windspeed	19735
Sales_Transactions	p19	52	Sales_Transactions	p18	52	Sales_Transactions	p20	52
Sales_Transactions	p1	52	Sales_Transactions	p2	52	Sales_Transactions	p3	52
Sales_Transactions	p7	52	Sales_Transactions	p6	52	Sales_Transactions	p4	52
Sales_Transactions	p5	52	Sales_Transactions	p8	52	Sales_Transactions	p9	52
Sales_Transactions	p10	52	Sales_Transactions	p11	52	Sales_Transactions	p13	52
Sales_Transactions	p12	52	Sales_Transactions	p16	52	Sales_Transactions	p17	52
Sales_Transactions	p15	52	Sales_Transactions	p14	52			
AdobeAveCPU_96x3270	S40	96	AdobeAveCPU_96x3270	S36	96	AdobeAveCPU_96x3270	S37	96
AdobeAveCPU_96x3270	S31	96	AdobeAveCPU_96x3270	S33	96	AdobeAveCPU_96x3270	S32	96
AdobeAveCPU_96x3270	S4	96	AdobeAveCPU_96x3270	S6	96	AdobeAveCPU_96x3270	S38	96
AdobeAveCPU_96x3270	S1	96	AdobeAveCPU_96x3270	S20	96			
fast-storage-20	Memory capacity provisioned	8615	fast-storage-20	Network received throughput	8615	fast-storage-20	Network transmitted throughput	8615
fast-storage-20	Disk write throughput	8615	fast-storage-20	CPU capacity provisioned	8615	fast-storage-20	CPU cores	8615
fast-storage-20	Timestamp	8615	fast-storage-20	CPU usage	8615	fast-storage-20	Disk read throughput	8615
fast-storage-20	Memory usage	8615						
knoy_mpu_3_300	X, Y	599	knoy_mpu_1_400	X, Y	1230	knoy_mpu_1_340	Y	1709
Scanline	scanline_42049	481	Scanline	scanline_126007	481			
iowa-electricity	net_generation	51	iowa-electricity	local_generation	51			
Adobe_CPU_Mem_15d	stageva6−STGusedcpu	2016	Adobe_CPU_Mem_15d	stageirl1−QA2usedcpu	2016	Adobe_CPU_Mem_15d	stageirl1−QA2usedmem	2016
Adobe_CPU_Mem_15d	stageva6−STG1usedmem	2016	Adobe_CPU_Mem_15d	stageirl1−QAusedcpu	2016	Adobe_CPU_Mem_15d	stageirl1−QA2usedcpu	2016
Adobe_CPU_Mem_15d	stageirl1−QA2usedmem	2016	Adobe_CPU_Mem_15d	stageirl1−QAusedmem	2016	Adobe_CPU_Mem_15d	stageva6−STG1usedcpu	2016
Adobe_CPU_Mem_15d	prodjpn3−PRODusedcpu	2014	Adobe_CPU_Mem_15d	stageirl1−Stageusedmem	2016	Adobe_CPU_Mem_15d	stageva6−QAusedmem	2016
Adobe_CPU_Mem_15d	stageirl1−QAusedcpu	2016	Adobe_CPU_Mem_15d	stageirl1−STG10usedcpu	2016	Adobe_CPU_Mem_15d	stageirl1−STG10usedmem	2016
Adobe_CPU_Mem_15d	stageirl1−QAusedmem	2016	Adobe_CPU_Mem_15d	stageva6−QAusedcpu	2016	Adobe_CPU_Mem_15d	prodjpn3−PRODusedmem	2014
Adobe_CPU_Mem_15d	stageirl1−Stageusedcpu	2016	Adobe_CPU_Mem_15d	prodjpn3−PROD1usedmem	2014	Adobe_CPU_Mem_15d	prodjpn3−Productionusedmem	2014
Adobe_CPU_Mem_15d	stageva6−STG1usedmem	2016	Adobe_CPU_Mem_15d	prodirl1−PRODusedcpu	2016	Adobe_CPU_Mem_15d	prodva6−PROD10usedmem	2016
Adobe_CPU_Mem_15d	prodva6−PROD10usedcpu	2016	Adobe_CPU_Mem_15d	prodjpn3−Productionusedcpu	2014	Adobe_CPU_Mem_15d	prodjpn3−PROD1usedcpu	2014
Adobe_CPU_Mem_15d	prodirl1−PRODusedmem	2016	Adobe_CPU_Mem_15d	stageva6−STG1usedcpu	2016	Adobe_CPU_Mem_15d	stageva6−QAusedmem	2016
Adobe_CPU_Mem_15d	prodirl1−PROD1usedcpu	2016	Adobe_CPU_Mem_15d	prodirl1−PRODusedcpu	2016	Adobe_CPU_Mem_15d	stageirl1−QAusedmem	2016
Adobe_CPU_Mem_15d	stageirl1−QAusedcpu	2016	Adobe_CPU_Mem_15d	stageva6−QAusedcpu	2016	Adobe_CPU_Mem_15d	prodirl1−PROD1usedmem	2016
Adobe_CPU_Mem_15d	prodirl1−PRODusedmem	2016	Adobe_CPU_Mem_15d	prodjpn3−PRODusedcpu	2014	Adobe_CPU_Mem_15d	stageirl1−QA2usedcpu	2016
Adobe_CPU_Mem_15d	stageva6−QAusedmem	2016	Adobe_CPU_Mem_15d	prodjpn3−PROD10usedcpu	2014	Adobe_CPU_Mem_15d	prodjpn3−PROD10usedmem	2014
Adobe_CPU_Mem_15d	stageva6−QAusedcpu	2016	Adobe_CPU_Mem_15d	stageirl1−QA2usedmem	2016	Adobe_CPU_Mem_15d	prodjpn3−PRODusedmem	2014
Adobe_CPU_Mem_15d	stageirl1−QA2usedcpu	2016	Adobe_CPU_Mem_15d	stageirl1−QA2usedmem	2016	Adobe_CPU_Mem_15d	stageva6−QAusedcpu	2016
Adobe_CPU_Mem_15d	stageirl1−QA2usedmem	2016	Adobe_CPU_Mem_15d	stageirl1−QAusedcpu	2016	Adobe_CPU_Mem_15d	stageva6−QA1usedmem	2016
Adobe_CPU_Mem_15d	stageva6−QAusedmem	2016	Adobe_CPU_Mem_15d	stageirl1−QAusedmem	2016	Adobe_CPU_Mem_15d	stageirl1−QA2usedcpu	2016
Adobe_CPU_Mem_15d	stageva6−QA1usedcpu	2016	Adobe_CPU_Mem_15d	prodva6−PROD1usedcpu	2016	Adobe_CPU_Mem_15d	prodirl1−PROD10usedcpu	2016
Adobe_CPU_Mem_15d	stageva6−QAusedmem	2016	Adobe_CPU_Mem_15d	stageva6−QAusedcpu	2016	Adobe_CPU_Mem_15d	stageva6−QAusedcpu	2016
Adobe_CPU_Mem_15d	stageva6−QAusedmem	2016	Adobe_CPU_Mem_15d	prodva6−PROD1usedmem	2016	Adobe_CPU_Mem_15d	prodirl1−PROD10usedmem	2016
Adobe_CPU_Mem_15d	stageirl1−QAusedcpu	2016	Adobe_CPU_Mem_15d	stageirl1−QAusedcpu	2016	Adobe_CPU_Mem_15d	stageirl1−QAusedmem	2016
Adobe_CPU_Mem_15d	stageirl1−QAusedmem	2016	Adobe_CPU_Mem_15d	stageirl1−QA2usedcpu	2016	Adobe_CPU_Mem_15d	stageirl1−QA2usedcpu	2016
Adobe_CPU_Mem_15d	stageirl1−QA2usedmem	2016	Adobe_CPU_Mem_15d	stageirl1−QA2usedmem	2016	Adobe_CPU_Mem_15d	stageirl1−STGusedmem	2016
Adobe_CPU_Mem_15d	stageva6−QAusedcpu	2016	Adobe_CPU_Mem_15d	stageirl1−STGusedcpu	2016	Adobe_CPU_Mem_15d	stageva6−QAusedmem	2016
Adobe_CPU_Mem_15d	prodva6−PRODusedmem	2016	Adobe_CPU_Mem_15d	stageirl1−STGusedmem	2016	Adobe_CPU_Mem_15d	stageirl1−QA10usedmem	2016
Adobe_CPU_Mem_15d	stageirl1−STGusedcpu	2016	Adobe_CPU_Mem_15d	stageirl1−QA10usedcpu	2016	Adobe_CPU_Mem_15d	prodva6−PRODusedcpu	2016
Adobe_CPU_Mem_15d	prodirl1−Productionusedmem	2016	Adobe_CPU_Mem_15d	stageirl1−QA10usedmem	2016	Adobe_CPU_Mem_15d	stageva6−STGusedmem	2016
Adobe_CPU_Mem_15d	stageirl1−STG1usedmem	2016	Adobe_CPU_Mem_15d	prodva6−Productionusedmem	2016	Adobe_CPU_Mem_15d	stageva6−STGusedcpu	2016
Adobe_CPU_Mem_15d	prodva6−Productionusedcpu	2016	Adobe_CPU_Mem_15d	stageirl1−STG1usedcpu	2016	Adobe_CPU_Mem_15d	prodirl1−Productionusedcpu	2016
Adobe_CPU_Mem_15d	stageirl1−QA10usedcpu	2016	Adobe_CPU_Mem_15d	stageirl1−STG1usedcpu	2016	Adobe_CPU_Mem_15d	stageirl1−QAusedcpu	2016
Adobe_CPU_Mem_15d	prodva6−PRODusedmem	2016	Adobe_CPU_Mem_15d	stageirl1−STGusedmem	2016	Adobe_CPU_Mem_15d	stageirl1−STGusedcpu	2016
Adobe_CPU_Mem_15d	prodva6−PRODusedcpu	2016	Adobe_CPU_Mem_15d	stageirl1−STG1usedmem	2016	Adobe_CPU_Mem_15d	stageirl1−QAusedmem	2016
Adobe_CPU_Mem_15d	stageva6−STGusedmem	2016						
ozone_eighthr	HT70	2534	ozone_eighthr	SLP	2534	ozone_eighthr	Precp	2534
ozone_eighthr	RH70	2534	ozone_eighthr	RH85	2534	ozone_eighthr	RH50	2534
ozone_eighthr	KI	2534	ozone_eighthr	SLP	2534	ozone_eighthr	HT85	2534
ozone_eighthr	HT50	2534						
quality_control	4	500	quality_control	5	325	quality_control	2	283
quality_control	3	366	quality_control	1	313			
knoy_mpu_3_400	X, Y	720	knoy_mpu_2_400	X, Y	1546	knoy_mpu_1_500	X, Y	2871
knoy_mpu_3_100	X, Y	824	knoy_mpu_1_360	X, Y	1252	knoy_mpu_2_100	X, Y	757
knoy_mpu_2_500	X, Y	1605	knoy_mpu_1_100	X, Y	1215	knoy_mpu_3_380	X, Y	574
knoy_mpu_2_320	X, Y	887	knoy_mpu_2_380	X, Y	848	knoy_mpu_1_380	X, Y	1499
Processed_NASD	DTB4WK	1984	Processed_NASD	EMA_50	1984	Processed_NASD	DTB3	1984
Processed_NASD	DTB6	1984	Processed_NASD	EMA_20	1984	Processed_NASD	FCHI	1984
Processed_NASD	FTSE-F	1984	Processed_NASD	EMA_10	1984	Processed_NASD	EMA_200	1984
Processed_NASD	EUR	1984						
Occupancy_CO2	Occupancy	8143	Occupancy_CO2	Temperature	8143	Occupancy_CO2	CO2	8143
Occupancy_CO2	Humidity	8143	Occupancy_CO2	Light	8143			
us-employment	financial_activities	120	us-employment	nonfarm_change	120	us-employment	construction	120
us-employment	mining_and_logging	120	us-employment	information	120	us-employment	professional_and_business_services	120
us-employment	durable_goods	120						
Occ_train	txt_Light	8143	Occ_train	txt_CO2	8143	Occ_train	txt_Humidity	8143
Occ_train	txt_HumidityRatio	8143	Occ_train	txt_Temperature	8143	Occ_train	txt_Occupancy	8143
Processed_DJI	DAX-F	1984	Processed_DJI	DE6	1984	Processed_DJI	DGS10	1984
Processed_DJI	DE5	1984	Processed_DJI	DE4	1984	Processed_DJI	DE1	1984
Processed_DJI	DE2	1984	Processed_DJI	DAAA	1984	Processed_DJI	DGS5	1984
Processed_DJI	DBAA	1984						
Processed_RUSS	Brent	1984	Processed_RUSS	AUD	1984	Processed_RUSS	AAPL	1984
Processed_RUSS	CNY	1984	Processed_RUSS	Close	1984	Processed_RUSS	CAD	1984
Processed_RUSS	copper-F	1984	Processed_RUSS	CHF	1984	Processed_RUSS	AMZN	1984
Processed_RUSS	CAC-F	1984						
Processed_NYSE	IXIC	1984	Processed_NYSE	JNJ	1984	Processed_NYSE	gold-F	1984
Processed_NYSE	Gold	1984	Processed_NYSE	JPM	1984	Processed_NYSE	GBP	1984
Processed_NYSE	GE	1984	Processed_NYSE	GDAXI	1984	Processed_NYSE	HSI-F	1984
Processed_NYSE	HSI	1984						
knoy_mpu_3_600	X, Y	876	knoy_mpu_2_600	X, Y	342	knoy_mpu_2_200	X, Y	765
knoy_mpu_1_600	X, Y	2321	knoy_mpu_3_200	X, Y	845			
co2-concentration	CO2	741	co2-concentration	adjusted CO2	741			
fast-storage-1	Disk write throughput	8634	fast-storage-1	Network received throughput	8634	fast-storage-1	Memory capacity provisioned	8634
fast-storage-1	Memory usage	8634	fast-storage-1	CPU capacity provisioned	8634	fast-storage-1	CPU cores	8634
fast-storage-1	Disk read throughput	8634	fast-storage-1	Network transmitted throughput	8634	fast-storage-1	CPU usage	8634

Table 17. Univariate Time-series dataset corpus description and details. The details of the datasets (i.e., the dataset name, and number of points in the dataset) are shown.

Time-series Name	# pts	Time-series Name	# pts	Time-series Name	# pts	Time-series Name	# pts	Time-series Name	# pts
OLDMANT.1	1461	ATT.1	97	NONEMERG.1	319	IV.1	44	HOPEDALE.1	101
WOLF.1	71	FRASER.1	946	EGDEMAN.2	100	IBM.1	87	LACSTJIN.1	1440
HARBOR.1	157	SERIESK.3	192	PORKH.1	99	construction	319	ESPANOLA.1	668
RIOTIETE.1	372	gdp_croatia	24	MINIMUM.1	848	TURTLE.1	672	OLDMAN.1	1470
THAMES.1	71	SERIESE.1	100	sp500_price	123	SP500.1	99	US.1	100
Y.1	44	BIGCONE.1	509	LACSTJRA.1	1440	FEEDH.1	95	SERIESG.1	153
FISHER.1	1470	GLOBWARM.1	129	TBILLS.1	102	CCPI.1	102	CAMPITO.1	5405
JOE.3	1294	CN.1	44	SERIESJY.1	307	bank	581	PRECIP.1	1096
GNPR.1	85	businv	330	MISINAB.1	672	JUDITH.1	492	TEMPER.1	1096
RING.1	66	LAKEVIEW.1	544	DAILYSAP.1	3333	GEODUCK.1	97	HALSEY.1	108
DAL.1	70	SALESX.1	93	AMAZON.2	55	IP.1	111	well_log	675
PCRGNP.1	62	DS_Store	137	FEEDL.1	95	DAILYIBM.1	3333	SERIESC.1	228
FRNCHB.1	45	MAD.1	552	UN.1	81	JAMES.1	600	CPI.1	288
LYNDPIN.2	136	OZONE.1	228	GRANT.1	151	SUNSPOTS.1	289	KIEWA.1	72
NEUMUNAS.1	132	YD.1	44	CHICKNYC.1	498	NILEJJ.1	75	robocalls	52
AMERICAN.1	660	RHINE.1	150	USM1.1	398	PORKL.1	99	ENGINES.1	188
NIAGARA.1	1861	YULE1.1	106	SERIESA.1	200	ASKEW4.1	660	homeruns	118
SKUNK.1	71	stocks_price	560	PRGNP.1	82	OLDMANP.1	1507	OOSTANAU.1	816
DANUBE.1	120	LACSTJSN.1	1440	EMERGING.1	319	CAFFEINE.1	178	us_population	816
PLSUPER.1	104	CIG.3	138	SIMAR4.1	818	SUNSPTMO.1	2820	ENGLISH.1	660
SERIESF.1	70	BOISE.1	588	FREEDMAN.1	58	ASKEW3.1	708	global_co2	104
WHEAT.1	370	EGGS.1	319	VATNSD.1	1098	SERIESJX.1	312	QBIRTHS.1	5117
MARTEN.1	71	FOOD.1	178	CONSUM.1	147	FEED.1	95	EXSHAW.1	506
WHITEMTN.1	1164	centralia	15	TSEOIL.1	361	BRYCE.1	625	MADISON.1	456
apple	622	GOLDH.1	97	MUSKRAT.1	71	BAYDU.1	358	jfk_passengers	468
SFSKYKOM.1	456	SERIESD.1	312	LOGISTIC.1	200	VELMON.1	86	PEAS.1	768
OTTER_L.1	71	METALS.1	178	BIRTHS.1	59	CURRENT.1	468	ASKEW5.1	108
ASKEW13.1	372	ELBE.1	300	FRNCHA.1	70	PLHURON.1	104	TOTAL.1	319
unemployment_nl	214	BEARDS.1	67	RIOGRAND.1	576	occupancy	509	COLUM.1	444
EMP.1	81	RGNP.1	85	rail_lines	37	PIPER.1	348	GNPN.1	85
SPIRITS.3	254	NIGERIA.1	123	MBOULDER.1	588	debt_ireland	21	IPI.1	85
CIGB.2	128	GOLDL.1	97	CMINEF.1	96	NILE2.1	100	ozone	54
ASKEW7.1	600	GLOBTP.1	136	gdp_iran	58	TIOGA.1	661	SERIESB.1	385
G.1	46	MCKEN.1	55	children_per_woman	301	YULE2.1	107	ISH66.1	163
GOTA.1	150	iot_temp	8402	PPHIL.1	1572	WOODS.1	629	DJWEEK.1	186
SAUGEEN.1	1403	USH.1	100	nile	100	co2_canada	215	NAVAJO.1	700
TRADE.1	178	GRUEN.1	53	PACK.2	344	bee_waggle_6	609	ARCTIC.1	66
whin_temp	6074	shanghai_license	205	WG.1	71	NYSE.1	87	AZUSA.1	180
SERIESL.2	549	SOY.1	99	FORTALEZ.1	150	NMAGNET.1	732	AROSA.1	480
FURNAS.DAT	576	LAKEMICH.1	115	MCKENZIE.1	600	ROCKY.1	122	SNOW.1	54
GUELPH.1	72	gdp_argentina	59	SOYL.1	99	SAUGEENP.1	1412	PORK.1	99
uk_coal_employ	105	ASKEW9.1	588	ASKEW14.1	588	TRINITY.1	588	OGDEN.1	97
TPMON.1	2976	I.1	44	DEATHS.1	319	SSASK.1	780	SOYH.1	99
SAUGEENT.1	1412	SKIRTS.1	69	M.1	85	OKAK.1	109	EAGLECOL.1	858
ASKEW10.1	600	WOLVEREN.1	71	SCHOLES.1	114	KINGS.1	49	STJOHNS.1	600
SERIESJ.2	616	NILEMON.1	910	BOXHU1.1	48	YEAR.1	208	BOXHUN.1	217
CORN.2	76	GNP.1	62	SUMMER.1	208	BLUME.1	64	NARAMATA.1	515
HURON.1	157	USM2.1	398	CRYER.1	43	usd_isk	247	MSTOUIS.1	96
MEASLNYC.1	534	seatbelts	192	ASKEW12.1	456	NILE.1	75	ALIGN.1	55
USL.1	100	ELECUS.1	51	SERIESB2.1	271	brent_spot	500	measles	991
RWG.1	71	SNAKE.1	669	REDDEER.1	396	MUMPS.1	534	CANFIRE.1	71
CMINER.1	528	CLEARWAT.1	600	bitcoin	774	NYWATER.1	71	ASKEW15.1	432
ASKEW.1	264	NECHES.1	564	RACOON.1	71	ASKEW8.1	456	CD.1	44
RAPPAHAN.1	600	AARIVINT.1	213	gdp_japan	58	WBDELAWA.1	540	TPYR.1	248
VEL.1	102	TRANEQ.1	178	NINEMILE.1	771	HEBRON.1	109	NAMAKAN.1	648
DELL.1	655	PLMICH.1	104	SERIESM.2	300	MINK.1	71	run_log	376
SUNSPT.1	261	CO2.1	192	PAPER.2	320	FEATHER.1	708	FLOW.1	468
FISHERT.1	1471	PREC.1	136	JOKULSA.1	1096	HBCO.1	66	WINTER.1	208
NAIN.1	109	CMINET.1	528	WATERQ.1	147	MEASLBAL.1	402	BND.1	71
LYNX.1	154	GOLD.1	97	lga_passengers	468	PIGEON.1	636	USM3.1	398
FISHERP.1	1471	IRONSU.3	143	BWATER.1	79	HANKOU.1	1368	DVI.1	470
RICHELU.1	468	DJ.1	157	U.1	85				