AutoForecast: Automatic Time-Series Forecasting Model Selection

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

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

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

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

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

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

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

CCS CONCEPTS
• Computing methodologies → Machine learning; Feature selection;

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

ACM Reference Format:

1 INTRODUCTION
Accurate time-series forecasting at scale is critical for a wide range of industrial domains such as cloud computing [37], supply chain [1], energy [11], and finance [33]. Most of the current time-series forecasting solutions are built by experts and require significant manual effort in model construction, feature engineering, and hyper-parameter tuning [6]. Hence, they do not scale to generate high-quality forecasts for a wide variety of applications. Moreover, there is no learning scheme that is uniformly better than all other learning schemes for all problem instances. For example, from our experiments (see Figure 2), we find empirically that no single forecasting model triumphs in more than 0.7% of the datasets in our two training testbeds comprising 625 time series (details in Section 6), i.e., there is no unique single model that works well on all datasets. A naive 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 $L$ 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 $L$ is trained on the models’ performances on historical datasets and the time-series meta-features of these datasets.

We emphasize that our time-series forecasting model selection meta-learning problem has several unique characteristics and challenges compared to previous related meta-learning problems, e.g., [16, 40, 54]. First, existing time-series forecasting models have different designs and different assumptions around the characteristics of time-series (e.g., probabilistic, seasonal, traditional, etc.). Therefore, different models perform differently depending on the characteristics that each dataset exhibits. Thus, capturing the similarity among different datasets needs careful selection of representative time-series meta-features. Second, the new meta-learning approach should capture the temporal variations of the models’ performances over different time windows of the dataset. This is borne out of our observation that...
the best time-series forecasting model for time window \(w_t\) is not necessarily the best model for a subsequent time window \(w_{t+1}\) (see Figure 3 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 developing our “general meta-learner” that learns to predict the performance of a model for a time window within a dataset. Second, we learn a model’s performance evolution over successive time windows for the same dataset via our “temporal meta-learner”. We train our meta-learner using a large model space which has over 320 forecasting models (Section 5.1). We also generate more than 800 meta-features that represent five different types of meta-features (simple, statistical, information theoretic, spectral-based, and landmark, which reflect various characteristics of the time-series datasets (Section 3). We also consider diverse datasets so our meta-learning model becomes generalizable to new time series datasets (Section 5.1). To stimulate reproducible research on this topic, we publicly release the corpus of datasets, along with their meta-features and the performances across hundreds of models, plus our source codes for training and evaluation\(^1\). Given a new (unseen) dataset, AutoForecast automatically determines, using the meta-features and the meta-learners, the best forecasting model among a large space of models, without the need to train and evaluate any of the different forecasting models on this new dataset.

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

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

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

\(^1\)The URL for our database and source codes is: https://drive.google.com/drive/folders/1Ki1w1ldScSr15b5PdXax-i-3NpWZyvct. The Adobe traces are available from: https://github.com/adobe-research/AutoForecast_ResourceUsageData.

![Figure 1: An overview of AutoForecast](image)

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

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

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

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

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

2 RELATED WORK

**Meta-learning in Time-series Forecasting:** There are few works that considered meta-learning for time-series analysis [19, 24, 35, 39, 51]. Among these, a few works considered simple ranking-based [24, 29] and rule-based [4, 51] meta-learners. The works [19, 39] applied a neural network time-series forecasting model trained on a source (energy) dataset and fine-tuned it on the target (energy) dataset. However, these works did not consider using meta-learning for the general problem of forecasting model selection that we consider in our current work. There also exist few works that have explored...
model selection problem using ensemble learning [15, 46, 49]. However, their problem domain of ensemble learning is different from our problem of model selection. This is due to the fact that ensemble learning constitutes building multiple models for the same task and does not in itself involve learning from prior experience on other tasks. In contrast to those works, AutoForecast can select among any (heterogeneous) set of methods. Finally, there is a line of work that considered empirical analysis for performance estimation [5, 9, 43] and model selection [10, 47] in time-series forecasting. However, these works have several distinctions from our work: (i) the need for evaluating all forecasting models in inference and (ii) providing an analysis of the ranking ability of performance estimators without having a meta-learner. In contrast, our meta-learner learns how to automatically select the best model and can capture the dependence within the same dataset.

**Few-shot Learning & Transfer Learning:** Few-shot learning has been recently leveraged for automating machine learning pipeline [33, 38, 45, 57]. In particular, the works [38, 45, 57] investigated different problems outside the domain of time-series forecasting. The work [33] applied meta-learning for zero-shot univariate time series forecasting. However, that work has the limitations of focusing on solving the cold start problem (learning model parameter initialization that generalizes better to similar tasks) which is different from our forecasting model selection problem, considering different models from the same N-BEATS architecture [54], and tackling only univariate time-series datasets. We emphasize that our framework can use N-BEATS as one forecasting algorithm in our model space. Finally, there exist few works that applied transfer learning for time series classification (TSC) [2, 13, 31, 53]. These works however have two distinctions from our problem: (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 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.

**Performance Tensor:** Now we introduce the performance tensor:

**Definition 1.** A model \( M_i \in M \) is given by the tuple \( M_i = (a_i, h_i, g_i(\cdot)) \), where \( a_i \) is the forecasting algorithm, \( h_i \) is the hyperparameter vector for the forecasting algorithm \( a_i \), and \( g_i(\cdot) : \mathbb{R}^{n\times m} \rightarrow \mathbb{R}^{n\times m} \) is the time-series data representation.

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

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

\[
\mathbf{P} = \{ p_{ij}^{1}, p_{ij}^{2}, \ldots, p_{ij}^{T} \}
\]

where \( P_{ik} = (p_{ik}^{ij}) \in \mathbb{R}^{n \times m} \) and the element \( p_{ik}^{ij} = M_j(w_k(D_i)) \) denotes the \( i \)-th model \( M_j \)’s performance on the time window \( w_k \) of the \( i \)-th training dataset \( D_i \). We denote \( P_k = \left[ p_{k}^{i1}, \ldots, p_{k}^{im} \right] \) as the performance vector of all models in \( M \) on time window \( w_k \) of \( D_i \).

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

**Definition 3.** Time-series forecasting model selection problem. Given a new input task (dataset) \( D_{\text{test}} \) (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 \( D_{\text{test}} \), select the best model \( \hat{M}_t \in M \) to employ on that window. Formally, such selection problem is given by

\[
\hat{M}_t = \arg \max_{M_j \in M} M_j(w_t(D_{\text{test}})), \quad t \in \{1, 2, \ldots, T\}.
\]

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

**Time-series Meta-Features:** A key component of AutoForecast is the extraction of meta-features that aims to capture the important characteristics of a time-series dataset. To achieve such a goal, we extract meta-features (defined below) for each time-series dataset.
We show the overview of the major components of work that capture the main characteristics of a dataset can be where \(\psi\) consists of two-phases: offline training of the \(\Theta\) meta-learner. Moreover, the cell update \(\sigma\) is given by

\[
\sigma_i = \sigma(W_o \cdot [h_{i-1}, X_i] + b_o).
\]

where \(W_f, W_l, W_o, b_f, b_l, b_o \in \mathbb{R}^m\) 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 \(u_i\) is constructed with a tanh activation function as follows.

\[
u_i = \text{tanh}(W_u \cdot [h_{i-1}, X_i] + b_u).
\]

\[
u_i = \sigma(W_o \cdot [h_{i-1}, X_i] + b_o).
\]

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 \(\Theta\) while others datasets would have weak temporal dependence among different windows performances in which the general meta-learner \(\Phi\) 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 (in Section 1).

General Meta-learner \(\Phi\): We propose multi-output regression model for training our general meta-learner \(\Phi\). From running all the models in \(M\) on different time windows \(\omega_t\) for all datasets in the training database \(D_{train}\), we collect a set of \(N = T \times n\) distinct training samples of the meta-features matrix and the performance vector \((F_i, \hat{p}_i^t)\), with \(t \in [1, T]\) and \(i \in [1, n]\).

Thus, the multi-output regression model is given by

\[
\hat{p}_i^t = \Phi(F_i, \beta) : t \in [1, T], i \in [1, n],
\]

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

\[
L_\Phi = \sum_{i=1}^{n} \sum_{t=1}^{T} L(p_i^t, \hat{p}_i^t),
\]

where \(L\) is the loss metric (e.g., MSE, MAPE, etc). Therefore, \(\Phi\) 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 \(\Theta\): The goal of the time-series meta-learner \(\Theta\) is to learn how the models’ performances evolve with the time-series meta-feature matrices over time. For this purpose, we propose time-series multi-regression model to learn such performance evolution. For any dataset \(D_i\), given the time-series meta-feature matrices \(F_i, F_i', \ldots, F_i^{T-1}\) and the history of the performance vectors \(p_i^1, \ldots, p_i^{T-1}\), we aim to predict performance vector \(p_i^T\) of current time window \(\omega_t\).

The time-series regression equation would be

\[
p_i^T = \Theta(F_i, p_i^{1}, \ldots, p_i^{T-1}, F_i', p_i'^1, \ldots, p_i'^{T-1}), t \in [1, n], t \in [1, T],
\]

where \(\Theta\) denotes the time-series regression function.

We adapt long-short term memory (LSTM) inputs for our time-series meta-learner \(\Theta\). We denote \(X_t\) as the input at the time window \(\omega_t\) which is given by \(X_t = [F_i, F_i', F_i'', \ldots, F_i^{T-1}, F_i^T]\). The predicted LSTM’s output denoted by \(\hat{p}_i^T\) is a function of \(X_t\). We now provide the detailed equations of such relation between \(\hat{p}_i^T\) and \(X_t\). The LSTM cell at time \(t\) has two recurrent features, denoted by \(h_t^i\) and \(c_t^i\), called the hidden state and the cell state, respectively. The LSTM cell consists of three layers (forget gate layer, input gate layer, and output gate layer). The activation of those layers is given by

\[
f_i = \sigma(W_f \cdot [h_{i-1}, X_i] + b_f),
\]

\[
l_i = \sigma(W_l \cdot [h_{i-1}, X_i] + b_l),
\]

\[
u_i = \text{tanh}(W_u \cdot [h_{i-1}, X_i] + b_u),
\]
where \( W_u \) and \( b_u \in \mathbb{R}^m \) are further weight and bias parameters to be learned. Thus, the new cell and hidden states at time \( t \) are
\[
\begin{align*}
    c_t^i &= f_t^i \cdot c_{t-1}^i + l_t^i \cdot u_t^i \\
    h_t^i &= o_t^i \cdot \tanh(c_t^i)
\end{align*}
\]
Finally, the output equations of the LSTM cell are given by
\[
V_t^i = W_o h_t^i + b_v
\]
\[
\hat{p}_t^i = \sigma(V_t^i),
\]
with \( W_o \) and \( b_v \in \mathbb{R}^m \) are learned weight and bias parameters. This gives the relationship between the input \( X_t \) and the predicted performance output vector \( \hat{p}_t^i \).

During training, \( \Theta \) learns the parameters \( W_f, b_f, W_i, b_i, W_o, b_o, W_c, b_c, W_r, b_r \), 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 \( \Theta \), is to minimize the loss denoted by \( L_\Theta \), given by
\[
L_\Theta = \sum_{t=1}^{T} \sum_{i=1}^{n} L(\hat{p}_t^i, p_t^i). \tag{6}
\]
We emphasize that \( \Theta \) 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 \( \Phi \) and the time-series meta-learner \( \Theta \), we now define the objective of our meta-learner \( \mathcal{L} \) given by a linear combination of the two components as follows:
\[
\min_{\Phi, \Theta} \alpha L_\Phi(F, P) + (1 - \alpha)L_\Theta(F, P), \tag{7}
\]
The parameter \( \alpha \) defines the relative weight of the two meta-learners. In our experiments, we chose \( \alpha = 0.5 \) for training our meta-learner. The meta-learner \( \mathcal{L} \) learns jointly general meta-learner \( \Phi \) (Equation 4) and time-series meta-learner \( \Theta \) (Equation 6) given meta-learner inputs, the performance tensor \( P \) and the meta-features tensor \( F \). 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-learner \( \mathcal{L} \) to quickly infer the best model for the current task. Given a new time-series dataset \( D_{test} \), AutoForecast first computes the corresponding meta-features tensor \( \hat{F}_{test} = \psi(D_{test}) \). Those time-series meta-features are then embedded (using PCA) to obtain the final meta-features tensor \( F_{test} \). Then, in the model inference, as shown in Figure 1, the model set performances are predicted for each available model in \( \mathcal{M} \). The model \( \hat{M}_t \) with the lowest predicted error score by \( \mathcal{L} \) on the time window \( w_0 \) of \( D_{test} \) is chosen as the selected model for that window \( w_0 \). Such a process is repeated for all time windows \( w_0, w_1, \ldots, w_T \) of \( D_{test} \).

Now, we explain such model selection process for each time window across the time windows \( w_0, w_1, \ldots, w_T \) of \( D_{test} \) as follows. For the first window \( w_0 \), the inference is given by \( \hat{M}_0 \in \arg\min_{M \in \mathcal{M}} \mathcal{L}(F_{0, test}) \). For any other window \( w_t \) \( (t > 0) \), the time-series meta-learner \( \Theta \) inference depends on the history of the meta-features and the history of the models’ performances as follows
\[
\hat{M}_t^\prime \in \arg\min_{M \in \mathcal{M}} \Theta(F_{t, test}, \hat{M}_{t-1}^\prime, \hat{p}_{t-1}^\prime). \tag{8}
\]
On the other hand, the general meta-learner \( \Phi \) inference depends on the predicted (regression) output on the meta-features of current time window where \( \hat{M}_t^\prime \in \arg\min_{M \in \mathcal{M}} \Phi(F_{t, test}) \). Thus, the final selected model is given by
\[
\hat{M}_t \in \arg\min_{M \in \{\hat{M}_t^\prime, M_0, \ldots \}} \hat{p}_{t, test}(\hat{M}). \tag{8}
\]
Table 1: Time-Series Forecasting Model Space. See hyperparameter definitions for various algorithms from GluonTS [3] and statsmodels [42]. The number of models (last column) is all possible combinations of hyperparameters and data representations.

<table>
<thead>
<tr>
<th>Forecasting Algorithm</th>
<th>HyperParameter 1</th>
<th>HyperParameter 2</th>
<th>Data Representation</th>
<th>Total</th>
</tr>
</thead>
<tbody>
<tr>
<td>DeepAR</td>
<td>num_cells = [10,20,30,40,50]</td>
<td>num_rnn_layers = [1,2,3,4,5]</td>
<td>[Exp_smoothing, Raw]</td>
<td>50</td>
</tr>
<tr>
<td>Prophet</td>
<td>changepoint_prior_scale = [0.001, 0.01, 0.1, 0.2, 0.5]</td>
<td>seasonality_prior_scale = [0.01, 0.1, 0.1, 0.5, 1.0]</td>
<td>[Exp_smoothing, Raw]</td>
<td>50</td>
</tr>
<tr>
<td>Seasonal Naive</td>
<td>season_length = [1.5,7,10,30]</td>
<td>N/A</td>
<td>[Exp_smoothing, Raw]</td>
<td>10</td>
</tr>
<tr>
<td>Gaussian Process</td>
<td>cardinality = [2,4,6,8,10]</td>
<td>max_iter_sitter = [5,10,15,20,25]</td>
<td>[Exp_smoothing, Raw]</td>
<td>50</td>
</tr>
<tr>
<td>Vector Auto Regression</td>
<td>core_type = [HC0, HC1, HC2, HC3, nonrobust]</td>
<td>trend = [n, c, t, ct]</td>
<td>[Exp_smoothing, Raw]</td>
<td>40</td>
</tr>
<tr>
<td>Random Forest Regressor</td>
<td>n_estimators = [10,100,250,500,1000]</td>
<td>max_depth = [2,5,10,25,50, None]</td>
<td>[Exp_smoothing, Raw]</td>
<td>72</td>
</tr>
</tbody>
</table>

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.

<table>
<thead>
<tr>
<th>Dataset Testbed</th>
<th>k</th>
<th>Global Best</th>
<th>AS</th>
<th>ISAC</th>
<th>MLP</th>
<th>AutoForecast-TSL</th>
<th>AutoForecast</th>
</tr>
</thead>
<tbody>
<tr>
<td>Univariate</td>
<td>1</td>
<td>2.46</td>
<td>2.15</td>
<td>0.82</td>
<td>0.62</td>
<td>2.67</td>
<td>3.95</td>
</tr>
<tr>
<td></td>
<td>5</td>
<td>7.18</td>
<td>4.92</td>
<td>2.67</td>
<td>1.13</td>
<td>9.04</td>
<td>14.57</td>
</tr>
<tr>
<td></td>
<td>10</td>
<td>11.97</td>
<td>7.89</td>
<td>4.10</td>
<td>4.51</td>
<td>14.15</td>
<td>21.45</td>
</tr>
<tr>
<td></td>
<td>50</td>
<td>37.40</td>
<td>28.00</td>
<td>11.45</td>
<td>22.25</td>
<td>35.28</td>
<td>52.65</td>
</tr>
<tr>
<td>Multivariate</td>
<td>1</td>
<td>6.78</td>
<td>2.26</td>
<td>4.19</td>
<td>0.43</td>
<td>5.16</td>
<td>5.87</td>
</tr>
<tr>
<td></td>
<td>5</td>
<td>12.18</td>
<td>4.73</td>
<td>5.69</td>
<td>1.51</td>
<td>9.03</td>
<td>13.86</td>
</tr>
<tr>
<td></td>
<td>10</td>
<td>16.21</td>
<td>9.03</td>
<td>7.31</td>
<td>4.06</td>
<td>11.39</td>
<td>20.91</td>
</tr>
<tr>
<td></td>
<td>50</td>
<td>41.72</td>
<td>24.73</td>
<td>14.64</td>
<td>20.86</td>
<td>35.06</td>
<td>51.67</td>
</tr>
</tbody>
</table>

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

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

Dataset Types in AutoForecast: We consider two general types of time-series datasets depending on the number of the variables \( v_i \) in the time-series dataset. (1) **Univariate Datasets** with single time-series (\( v_1 = 1 \)) and (2) **Multivariate Datasets** with several time-series (variables) (i.e., \( v_i > 1 \)) that need to be predicted. In particular, we collect 308 univariate time-series datasets for the first testbed and 40 multivariate datasets with 317 time-series for the second testbed. In total, we have 625 time-series in our testbeds. For each dataset in the testbeds, we use different time windows selected randomly from the dataset, where each time window has a length of 16 (i.e., \( |w_i| = 16 \forall t \in \{1, \ldots, T\} \)). Note that our approach has no restriction on the length of the time window nor any assumption of the homogeneity of windows duration.

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

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

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

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

- **Random Forest (RF) [27]:** is a SOTA tree ensemble that combines the predictions made by many decision trees into a single model. In prediction, the RF regression model takes the average of all the individual decision tree estimates.

- **SOTA Forecasting Algorithms:** We selected seven popular time series forecasting models, including the recent works DeepAR [41], DeepFactors [52], and Prophet [48]. For each model, we generated multiple variants by varying the values of hyperparameters and data representations (10-72 variants, details in Table 1) and we chose the model variant with the best average performance across all training datasets.

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

- **Global Best (GB):** It selects the forecasting model with the largest average performance across all train datasets (across all time windows), without using any meta-features.

- **ISAC [20]:** clusters the training datasets based on meta-features. Given a new test time-series dataset, it identifies its closest cluster and selects the best model with the best average performance on the cluster’s datasets.

- **ARGOSMART (AS) [32]:** finds the closest training time-series dataset to a given test time-series dataset, based on meta-feature similarity, and selects the model with the best performance on that nearest neighbor training dataset.

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

- **Multi-layer Perceptron (MLP):** Given the training datasets and selected time window, the MLP regressor directly maps the meta-features onto model performances by regression. However, this does not learn temporal dependence within datasets.

- **AUTOFORECAST-TSL:** is a variant of our solution in which the meta-learner $\mathcal{L}$ consists only of the time-series learner $\Theta$.

5.3 Results

5.3.1 Variation of Best Model across Time. Figure 2 shows that no single forecasting model triumphs in more than 0.7% of the datasets. Figure 3 shows the aggregate statistics on all datasets of the univariate testbed. This contradicts the claims that one forecasting algorithm can work best for different datasets and motivates the need for an effective approach for learning such both dimensions, which we propose in our current work.
Table 4: Results for one-step ahead forecasting (MSE; the lower the better) for both testbeds. The selected model by AutoForecast yields better performance compared to baseline meta-learners and SOTA methods. For each SOTA, we choose the model with the best average performance from all its model variants.

<table>
<thead>
<tr>
<th>Dataset Testbed</th>
<th>Seasonal Naive</th>
<th>DeepAR</th>
<th>Deep Factors</th>
<th>Random Forest</th>
<th>Prophet</th>
<th>Gaussian Process</th>
<th>VAR</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Univariate</strong></td>
<td>0.0345 ± 0.0082</td>
<td>0.0164 ± 0.0056</td>
<td>0.0217 ± 0.0415</td>
<td>0.0199 ± 0.0398</td>
<td>0.0155 ± 0.0295</td>
<td>0.1661 ± 0.2104</td>
<td>0.0062 ± 0.1260</td>
</tr>
<tr>
<td><strong>Multivariate</strong></td>
<td>0.0055 ± 0.0199</td>
<td>0.0153 ± 0.0566</td>
<td>0.0071 ± 0.0145</td>
<td>0.0531 ± 0.1184</td>
<td>0.00463 ± 0.0118</td>
<td>0.00256 ± 0.0090</td>
<td></td>
</tr>
</tbody>
</table>

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.

<table>
<thead>
<tr>
<th>Method 1</th>
<th>Method 2</th>
<th>p-value</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>AutoForecast</strong></td>
<td>GB</td>
<td>0.0712 ± 10^{-3}</td>
</tr>
<tr>
<td>AutoForecast AS</td>
<td>1.0726 ± 10^{-7}</td>
<td></td>
</tr>
<tr>
<td>AutoForecast ISAC</td>
<td>0.1404</td>
<td></td>
</tr>
<tr>
<td><strong>AutoForecast MLP</strong></td>
<td>0.6357</td>
<td></td>
</tr>
<tr>
<td>AutoForecast TSL</td>
<td>8.1683 × 10^{-3}</td>
<td></td>
</tr>
<tr>
<td>AutoForecast-TSL GB</td>
<td>2.3611 × 10^{-4}</td>
<td></td>
</tr>
<tr>
<td>AutoForecast-TSL AS</td>
<td>1.5706 × 10^{-3}</td>
<td></td>
</tr>
<tr>
<td>AutoForecast-TSL ISAC</td>
<td>2.3943 × 10^{-3}</td>
<td></td>
</tr>
<tr>
<td><strong>AutoForecast-TSL MLP</strong></td>
<td>1.3658 × 10^{-6}</td>
<td></td>
</tr>
<tr>
<td>GB AS</td>
<td>9.6952 × 10^{-3}</td>
<td></td>
</tr>
<tr>
<td>GB ISAC</td>
<td>0.0222</td>
<td></td>
</tr>
<tr>
<td>GB MLP</td>
<td>4.5899 × 10^{-3}</td>
<td></td>
</tr>
<tr>
<td>AS ISAC</td>
<td>1.7842 × 10^{-7}</td>
<td></td>
</tr>
<tr>
<td>AS MLP</td>
<td>4.4658 × 10^{-16}</td>
<td></td>
</tr>
<tr>
<td>ISAC MLP</td>
<td>2.2062 × 10^{-16}</td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Method 1</th>
<th>Method 2</th>
<th>p-value</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>AutoForecast</strong></td>
<td>GB</td>
<td>1.56</td>
</tr>
<tr>
<td>AutoForecast AS</td>
<td>3.9999 × 10^{-1}</td>
<td></td>
</tr>
<tr>
<td>AutoForecast ISAC</td>
<td>0.8240</td>
<td></td>
</tr>
<tr>
<td>AutoForecast MLP</td>
<td>0.0004</td>
<td></td>
</tr>
<tr>
<td>AutoForecast TSL</td>
<td>0.0025</td>
<td></td>
</tr>
<tr>
<td>AutoForecast-TSL GB</td>
<td>0.00254</td>
<td></td>
</tr>
<tr>
<td>AutoForecast-TSL AS</td>
<td>5.4033 × 10^{-1}</td>
<td></td>
</tr>
<tr>
<td>AutoForecast-TSL ISAC</td>
<td>1.5598 × 10^{-1}</td>
<td></td>
</tr>
<tr>
<td><strong>AutoForecast-TSL MLP</strong></td>
<td>3.4572 × 10^{-4}</td>
<td></td>
</tr>
<tr>
<td>GB AS</td>
<td>3.9999 × 10^{-1}</td>
<td></td>
</tr>
<tr>
<td>GB ISAC</td>
<td>0.8240</td>
<td></td>
</tr>
<tr>
<td>GB MLP</td>
<td>0.0004</td>
<td></td>
</tr>
<tr>
<td>AS ISAC</td>
<td>1.4217 × 10^{-1}</td>
<td></td>
</tr>
<tr>
<td>AS MLP</td>
<td>6.6412 × 10^{-4}</td>
<td></td>
</tr>
<tr>
<td>ISAC MLP</td>
<td>3.7789 × 10^{-1}</td>
<td></td>
</tr>
</tbody>
</table>

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

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

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

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

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

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

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

(1) Reproducibility of AutoForecast: To further research into the important problem introduced in our work, we have publicly released our source codes and benchmark data to enable others to reproduce our work. In particular, we are publicly releasing, with this submission, our meta-learning database corpus of 348 datasets, containing 625 time series in all, performances of the 322 forecasting models, and meta-features for the datasets. This resource will hopefully encourage the community to standardize efforts at benchmarking time series forecasting model selection. We also encourage the community to expand this resource by contributing their new datasets and models. The anonymized website with our database and source codes is: https://drive.google.com/drive/folders/1Kiwl1d5a5Cr15b5Hidax-i-NpWZyvct. The details of each dataset in the two testbeds and the different categories of meta-features are presented in Section 5.1. This serves as the training data and ground truth evaluation data for AutoForecast.

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

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

7 CONCLUSION

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