SimpleTS: An Efficient and Universal Model Selection Framework for Time Series Forecasting

Yuanyuan Yao
Zhejiang University
yoyoyao@zju.edu.cn

Dimeng Li
Alibaba Group
lidimeng.ldm@alibaba-inc.com

Hailiang Jie, Lu Chen
Zhejiang University
{hljie,luchen}@zju.edu.cn

Tianyi Li
Aalborg University
tianyi@cs.aau.dk

Jie Chen
Alibaba Group
aiao.cj@alibaba-inc.com

Jiaqi Wang
Zhejiang University
jqiwang@zju.edu.cn

Feifei Li
Alibaba Group
lifeifei@alibaba-inc.com

Yunjun Gao
Zhejiang University
gaoyj@zju.edu.cn

ABSTRACT

Time series forecasting, that predicts events through a sequence of time, has received increasing attention in past decades. The diverse range of time series forecasting models presents a challenge for selecting the most suitable model for a given dataset. As such, the Alibaba Cloud database monitoring system must address the issue of selecting an optimal forecasting model for a single time series data. While several model selection frameworks, including AutoAI-TS, have been developed to predict a dataset, their effectiveness may be limited as they may not adapt well to all types of time series, resulting in reduced prediction accuracy. Alternatively, models such as AutoForecast, which train on individual data points, may offer better adaptability but are limited by longer training time required.

In this paper, we introduce SimpleTS, a versatile framework for time series forecasting that exhibits high efficiency and accuracy across all types of time series data. When performing an online prediction task, SimpleTS first classifies input time series into one type, and then efficiently selects the most suitable prediction model for this type. To optimize performance, SimpleTS (i) clusters models with similar performance to improve the efficiency of classification; (ii) uses soft labeling and weighted representation learning to achieve higher classification accuracy for different time series types. Extensive experiments on 3 private datasets and 52 public datasets show that SimpleTS outperforms the state-of-the-art toolkits in terms of both training time and prediction accuracy.

PVLDB Reference Format:
Yuanyuan Yao, Dimeng Li, Hailiang Jie, Lu Chen, Tianyi Li, Jie Chen, Jiaqi Wang, Feifei Li, and Yunjun Gao. SimpleTS: An Efficient and Universal Model Selection Framework for Time Series Forecasting. PVLDB, 16(12): 3741 - 3753, 2023.
doi:10.14778/3611540.3611561

1 INTRODUCTION

It is increasingly possible to access and store a vast majority of time series data, such as retail, supply chain, energy, finance, power, etc [11, 23, 25]. These scenarios provide opportunities for time series data mining [18, 24]. Time series prediction is a widely-used data mining task, which involves the analysis of historical data to forecast future events on the assumption that future trends will follow similar patterns and trends observed in the past [20]. Accurate time series prediction is highly demanded in real-life. For example, improving the accuracy of consumers' demand forecast by 10%-20% can save the inventory of supermarkets by 5% and enhance both the revenue and service level by 2%-3% [34].

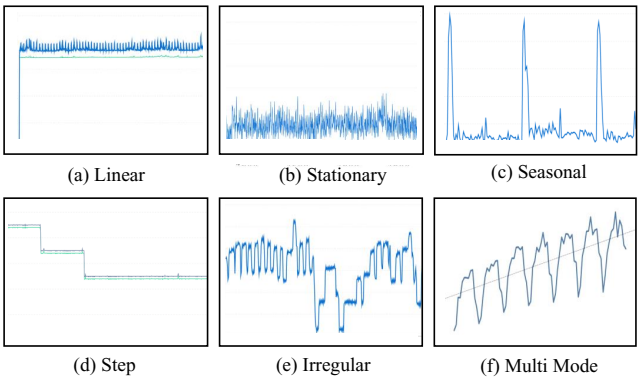

Figure 1: Different types of time series data

Time series exhibit different trends over time, as shown in Figure 1. Linear (Fig. 1(a)) and stationary (Fig. 1(b)) indicate that data fluctuates around a fixed number. Seasonal (Fig. 1(c)) expresses that data experiences regular and predictable changes. Step (Fig. 1(d)) refers to that data generally drops (or grows) but the variation is not continuous. Irregular (Fig. 1(e)) and multi-mode (Fig. 1(f)) depict dynamic variation of data. The performance of the same algorithm varies greatly when predicting different types of time series data. For example, some traditional forecast algorithms (e.g., ARIMA [45], Holt-Winters [4]) for time series perform well on linear or periodical data but perform poorly on irregular type datasets. As shown in Figure 2, the time series prediction task is the preceding task in the database autonomous service (DAS) of Alibaba Cloud (AliCloud), which plays a very important role. The ability to make accurate predictions is crucial for identifying and addressing issues in a timely manner. Motivated by this, we aim to develop effective time series prediction models that can provide accurate forecasts

Proceedings of the VLDB Endowment, Vol. 16, No. 12 ISSN 2150-8097.
doi:10.14778/3611540.3611561

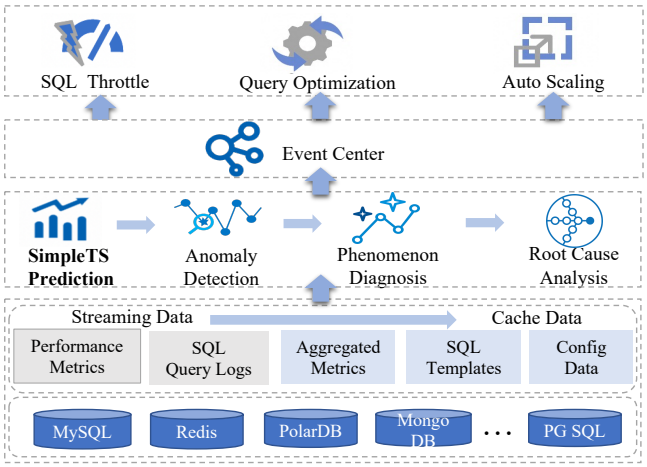

Figure 2: SimpleTS predication task in DAS

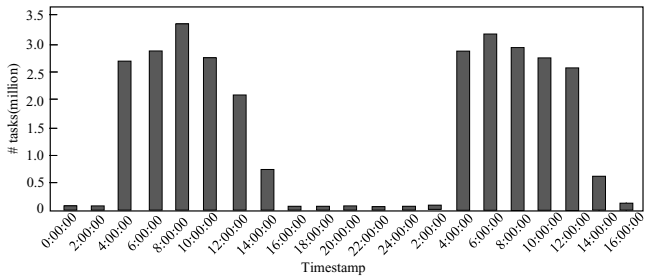

Figure 3: Number of prediction tasks run in DAS over time (accumulated every two hours)

for diverse types of time series data, irrespective of their distinct patterns and characteristics.

The database engines of AliCloud (such as time series database TSDB) store the business data of massive users. It has been applied to many fields including retail, e-commerce, finance and supply chain. As shown in Figure 3, Alibaba Cloud's DAS runs more than 2 million predictions tasks every two hours from 4am to 2pm every day, i.e., over 16 thousand tasks per minute. This huge amount of data implies that efficiency should be taken into account when the model is iteratively updated. In addition, we also aim to improve the efficiency of model training, such that better prediction results can be achieved in less training time.

AutoML [39] can be used to automatically select and create models [19, 37]. However, AutoML methods still have the following limitations. First, they need to run the prediction on the whole training dataset using each candidate model and then identify the one achieving the highest performance as the best. For example, Google AutoML [3] and IBM AutoAI-TS [36] employ this strategy, which is intuitively time-consuming and in-efficient. Second, AutoML does not take the different types of time series into consideration during the model selection. For example, AWS forecast framework [1] only considers local weather and holiday information. However, it is not sufficient to achieve high prediction accuracy, as there are multiple types of time series (cf. Figure 1). Third, all the above-mentioned AutoML methods cannot perform well on the dataset with multiple types of time series. This is because, such methods always return a universal model, which is assumed to be able to accommodate all types of time series. However, as each of time series forecasting algorithm is only designed for dealing with one or several types of time series, such an assumption is obviously unreasonable.

Other model selection studies [6, 29, 46] suffer the same problem. They compare the distribution of the input data with that of the trained data by different features. Some of them [29, 46] assume that the trained model can be applied to any input data, which may be reasonable for the area of computer vision, but is not suitable for time series data. Unlike training the entire dataset at once, AutoForecast [6] utilizes meta-learning to extract and train features for each individual data point. However, it can be time-consuming and cannot guarantee high accuracy. Motivated by these, we aim to develop an efficient and universal model selection framework for time series, where three following challenges exist.

Challenge I: How to achieve higher accuracy with less training time? With the increase in forecast demand and forecast algorithms, it is inefficient to run all the models on the whole training dataset to find the best algorithm for forecasting. Though methods are proposed (such as pruning) to accelerate model selection [46], they still cannot cope with the huge computation costs caused by performing predictions on the whole training dataset.

To tackle this challenge, we propose to train a classifier with a small amount of data and then to integrate the classifier into model selection. First, we perform predictions using all the candidate models on small amount of the training data. Next, we cluster the models by their performance and record the model that achieves the highest performance on each type of data as the corresponding optimal model. Finally, we exploit the classifier to divide the training data sets according to their types, including linear, periodic and irregular. In this way, whenever new data arrives, we only need to first identify its type by the classifier, and then apply its corresponding optimal model for prediction. Obviously, with the integrated classifier, we can find the model that performs the best for this type of data to further improve its accuracy.

Challenge II: How to improve the accuracy of the classifier? The diversity of types of time series tends to increase over time [11]. Also, the accuracy of the classification generally drops when the number of types of time series increases, as the large number of labels associated with diversified data types may make the classifier "confused". Thus, how to improve the classification accuracy on a large volume of data is challenging.

To tackle this challenge, we first cluster forecasting algorithms with similar performance to reduce the number of categories for the classification. As time series data with the same shape will behave similarly in the different models, we can cluster models with similar performance into one category to represent one time series type. This way, we can decrease the number of time series types for classification in order to improve the accuracy of the classifier. Second, for the classification task, hard labeling generally degrades accuracy because the outputs of the positive and negative samples vary greatly. On the contrary, soft labeling [17] can avoid overconfidence in correct labels, and thus prevent over-fitting and improve the generalization ability of the model.

Challenge III: How to select the classification features of source data? There are two ways to capture temporal features, one is calculating artificial features by expert experience, and the other is capturing hidden features by representation learning [26]. For time series data, the shape of the data can be represented by artificial features, such as periodicity [48], stationarity [33], turning point, maximum, minimum, average, and other statistical variables. However, these artificial features cannot represent the complex context information of the time series, especially the time series with irregular shapes, which degrades the classification accuracy.

To address this challenge, we use TS2Vec [52] that enables a robust contextual representation for each timestamp and performs contrastive learning in a hierarchical way over augmented context views. Based on this, we present a weighted representation learning. The intuition behind it is to represent the correlation between every two instances as a weight when calculating the loss function. Instances are used to describe time series data from different sources, such as data from various database machines in DAS or data from different store locations in the Walmart dataset. Experimental results on the three private datasets (cf. Section 6.4) verify that the proposed weight representation learning outperforms the methods employing artificial features.

We develop a novel framework SimpleTS by integrating the above techniques (including classification, clustering, soft labels, and representation learning). SimpleTS is equipped with fourteen time series forecasting models, including twelve state-of-the-art models and two newly developed models. Overall, the major contributions of this paper are summarized as follows:

- We develop SimpleTS, an efficient and universal framework that can identify a good prediction model based on the time series type. To the best of our knowledge, SimpleTS is the first framework that incorporates a classifier to distinguish time series types for prediction, which enhances the training efficiency and accuracy of model selection.
- We incorporate a classifier in SimpleTS to identify the time series type of the source data, while proposing to exploit clustering, soft labeling and a weighted representation learning to improve the accuracy of the classifier.
- We develop a configurable interface that offers high flexibility in parameter selection for offline model selection tasks. Furthermore, our interface enables users to easily visualize the results of their model training.
- We conduct extensive experimental evaluations on 3 private datasets and 52 public datasets, which demonstrates SimpleTS is able to outperform the state-of-the-art methods by 4-10 times faster in terms of training time, while able to achieve higher accuracy than the state-of-the-art methods.

The rest of the paper is as follows. We present preliminaries in Section 2 and give an overview framework in Section 3. Section 4 and Section 5 detail two main components in our framework respectively. Section 6 reports the experimental results. Section 7 reviews related work, while Section 8 concludes the paper.

2 PRELIMINARIES

We proceed to define the problem, and introduce existing model selection methods and a representation learning framework.

2.1 Problem Definition

Definition 2.1 (Time series Data). A time-series data is a sequence of numerical data points in time order: $X = \langle x_1, x_2, ..., x_L \rangle$, where x_i $(1 \leq i \leq L)$ is the observation value at time T_i, and L is the sequence length.

Definition 2.2 (Time series Forecasting). Given the current time t and last l observations $(x_{t-l+1}, ..., x_t)$ of a time series data X, time series forecasting aims to predict the future H observations $x_{t+1}, ..., x_{t+H}$ of X.

Problem Statement. Given a time series dataset X that consists of n time series types $\{T_1, T_2, ..., T_n\}$, the model selection task is to select n models from a set of M forecasting models such that the chosen n models $\{M_1, M_2, ..., M_n\}$ perform optimally on n time series types within this dataset.

2.2 Model Selection

AutoML [39] is a general term that refers to the process of finding the optimal model among the possible choices of models and their parameters. Earlier proposed AutoML is mainly composed of three parts: feature engineering, model selection, and algorithm selection. The model selection consists of two steps: selecting a model and setting its parameters [12, 13]. AutoML automatically selects an optimization model to achieve a balance between efficiency and accuracy. However, choosing the model and parameter values that perform the best on the given data requires trying all the possibilities, which is time-consuming. Although there are several acceleration methods, such as sampling and pruning [38], they only find a single model that performs well on all types of time series data within a dataset containing diverse types.

IBM proposes AutoAI-TS [36] for time series forecasting, which provides a zero-configuration system to effectively train, optimize and select the best model for a given time series data set. For a given data set, AutoAI-TS leverages a range of models, such as traditional statistical models, machine learning models, hybrid models that combine statistics and machine learning, and deep learning models to generate predictive pipelines for a given dataset. It evaluates and ranks the pipelines using the suggested T-Daub mechanism to select the best one.

AutoForecast [6] supports fast automatic selection of the best forecasting model for a new unseen time-series dataset, which does not need to train (or evaluate) all the models on the whole training dataset before selecting the best one. To effectively capture dataset similarity, AutoForecast performs feature extraction on each piece of time series data, and then traverses the time series forecasting algorithm to find the relationship between the features and the predicted results in order to train two meta learners. One is the general meta-learner, mapping the meta-features vector to the corresponding best-model; The other is the time-series meta-learner, capturing the relation between the models' performances in time with the meta-features.

2.3 Representation Learning

Feature extraction is the basis of the classification task. It can be done manually or automatically with the help of specific algorithms. The former is feature engineering, and the latter is representation

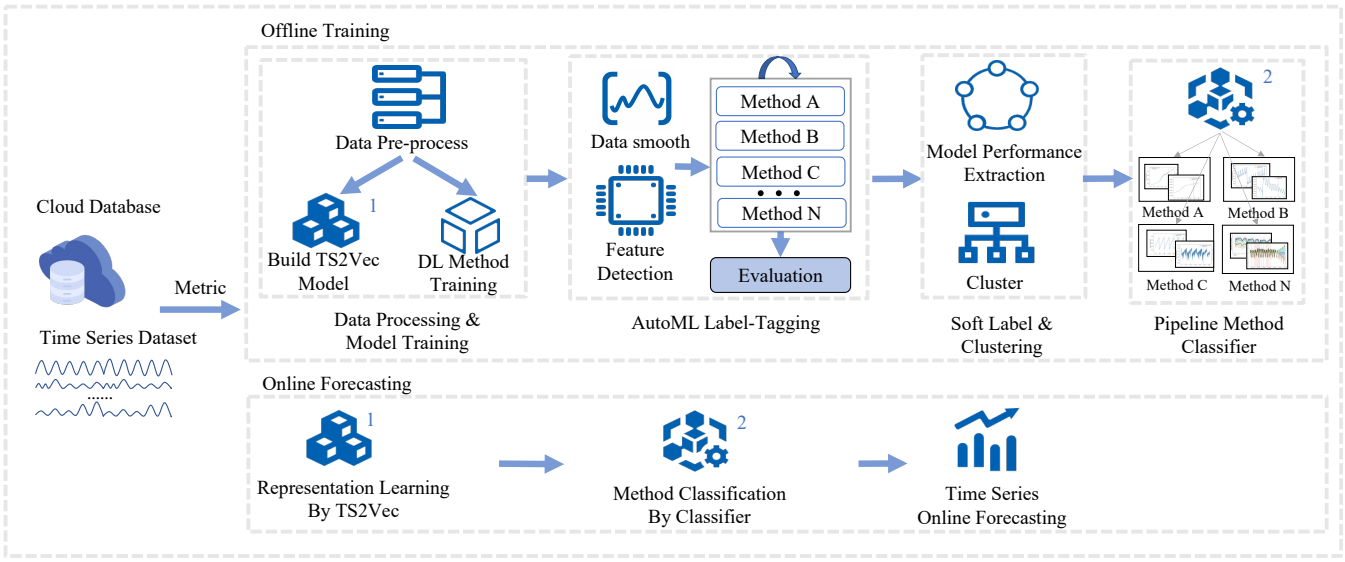

Figure 4: Overall architecture of SimpleTS

learning. If the amount of data is small, we can artificially design appropriate features based on our experience and prior knowledge for downstream tasks, such as classification. However, artificial feature engineering relies on expert experience and cannot capture the correlation between instances. Hence, the information that artificial feature engineering can express is very limited, particularly in complex data scenarios such as DAS. Hence, automated representation learning is required. Representation learning can well capture the features from the complex data. Motivated by this, we use representation learning to extract the features.

Different from feature engineering, representation learning automatically extracts features to best reflect the essence of the problem from the dataset [26]. TS2Vec [52] is a general framework for learning representations at arbitrary semantic levels for time series data. TS2Vec proposes a hierarchical contrast loss that forces the encoder to learn representations at different scales. It uses instance-wise loss and temporal contrastive loss to encode the distribution of time series. An instance represents an object that generates a sequence of time series data. For example, each store in the Walmart dataset represents an instance, which contains sales data over a period of time. Let i be the instance index of the input time series sample x and t be the timestamp; while let $r_{i,t}$ and $r'_{i,t}$ denote the representations for the same timestamp t but from two augmentations of x_i. The temporal contrastive loss for the i-th time series at timestamp t can be formulated as:

$$\ell_{temp}^{(i,t)} = -\log \frac{\exp\left(r_{i,t} \cdot r'_{i,t}\right)}{\sum_{t' \in \Omega}\left(\exp\left(r_{i,t} \cdot r'_{i,t'}\right) + \mathbb{1}_{[t \neq t']} \exp\left(r_{i,t} \cdot r_{i,t'}\right)\right)} \tag{1}$$

where Ω is the set of timestamps within the overlap of the two subseries, and $\mathbb{1}$ is the indicator function (i.e., $\mathbb{1}_{[condition]} = 1$ when $condition$ = true, $\mathbb{1}_{[condition]} = 0$ when $condition$ = false).

The instance-wise contrastive loss can be computed as:

$$\ell_{inst}^{(i,t)} = -\log \frac{\exp\left(r_{i,t} \cdot r'_{i,t}\right)}{\sum_{j=1}^{B}\left(\exp\left(r_{i,t} \cdot r'_{j,t}\right) + \mathbb{1}_{[i \neq j]} \exp\left(r_{i,t} \cdot r_{j,t}\right)\right)} \tag{2}$$

where B denotes the batch size. It uses representations of other time series at timestamp t in the same batch as negative samples.

The two loss functions are complementary to each other. For example, given a set of electricity consumption data from multiple users, instance-wise contrastive loss learns the user-specific characteristics, while temporal contrastive loss aims to mine the dynamic trends over time. Hence, the overall loss is defined as:

$$\mathcal{L}_{dual} = \frac{1}{NT} \sum_{i} \sum_{t} \left(\ell_{temp}^{(i,t)} + \ell_{inst}^{(i,t)}\right) \tag{3}$$

TS2Vec applies max pooling on the learned representations along the time axis and computes Equation 3 recursively, where NT denotes the number of iterations until $len(r) \leq 0$. In the hierarchical contrast model, the loss function is applied at all levels of granularity. TS2Vec demonstrates its universality and effectiveness on three time-series-related tasks, including time series classification, forecasting and anomaly detection.

3 SYSTEM ARCHITECTURE

The overall architecture of SimpleTS is shown in Figure 4, including offline training and online forecasting. The offline training includes four main components, i.e., data pre-processing and model training, AutoML label-tagging, soft label and clustering, and pipeline method classifier.

- The main task of data pre-processing is to transform source data into the data in the required format for training, and then use these processed data to train the weighted TS2Vec [52] model and deep learning forecasting methods. Note that, the trained models/methods will be used in the following.

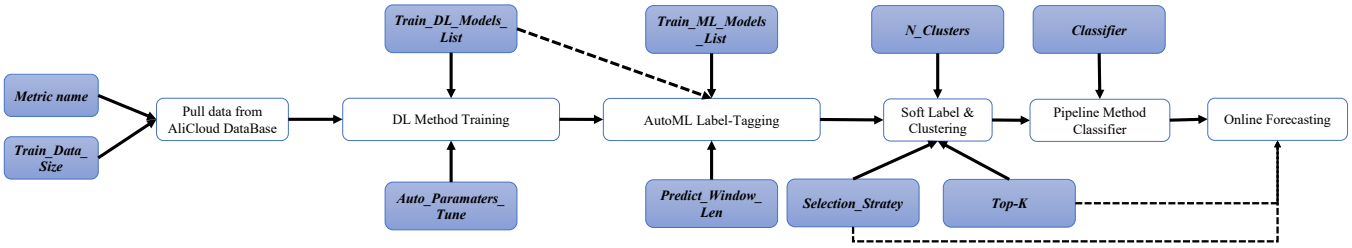

Figure 5: Offline Training System of SimpleTS

- In the AutoML label-tagging component, all the time series methods do prediction tasks on each training data, annotating the forecasting result, accuracy, precision, SMAPE, and other detailed information of each method automatically.
- After that, we use the clustering method to cluster these forecasting methods according to the annotated information, where soft and hard labels are used. Note that, we also record the best-performing model of each type in the configuration file based on the clustering result.
- Finally, we use the trained weighted TS2Vec model to encode the data, capturing the context information on the source data and the relation information between instances, and also classify the encoded data using the labels of the clustering results. To sum up, the main task of offline training is to train a classifier for online prediction.

For the online forecasting, when a piece of new data comes, we first encode the data via trained weighted TS2Vec model, and then classify it into a time series type using the trained classifier. Note that, weighted TS2Vec model and the classifier can be obtained in the offline training. According to the classification results, we search the configuration file to view the best-performing model of this type to execute the prediction. In general, once the classifier is trained, SimpleTS framework typically uses the most appropriate model for performing prediction tasks. In the following two sections, we will detail the two main components, respectively.

4 OFFLINE TRAINING

In this section, we introduce the offline training system and corresponding techniques.

4.1 Offline Training System

We have developed an offline training system for the DAS, which enables users to adjust various parameters for training. The relationship between the configurable training parameters and the business logic is depicted in Figure 5, where the blue rectangles denote the configurable training parameters. Table 1 provides the detailed parameter descriptions.

First, users can input the name of the database metric they want to analyze. The DAS system supports 63 database metrics[1], such as Mysql.cpu_usage, Mysql.disk_usage, Mysql.iops_usage, slow_queries, mem_usage, and so on. Additionally, as there are tens of millions of time series data in the AliCloud database, users can choose a certain scale of data for training, such as 10%, 20%, 30%, etc. Based on the input of the metric name and data size, the system will retrieve the data observed in the past 10 days for the specified metric from the AliCloud database.

Then, the system offers two types of models for training configuration: deep learning models and machine learning models. The deep learning models include LSTM [21], Transformer [49], DeepAR [14], DeepFactor [9], DeepState [35], GPForecaster [8], NBeats [32], while the machine learning models include ETS [2], Prophet [43], NPTS [5], ARIMA [45], Holt-Winters [4] and two self-developed methods (i.e., Period, Linear). Period method predicts by analyzing the periodicity of data, i.e., it makes the forecast based on the previous periodic data. Therefore, Period method will not be effective when the data do not have any periodicity. Linear is an optimization algorithm based on ETS. It prevents outliers or spikes from affecting the prediction results by truncating the data according to the points of change, and makes predictions based on the effective trend. Users can choose single or multiple models of each type. For deep learning models, the system provides automatic parameter tuning services, which can enhance model training accuracy to some extent. However, parameter tuning will result in longer training time. Hence, users can select this option based on their specific needs. The system also offers the option to choose the prediction window length (i.e., a percentage of window length w.r.t. the time series length) for prediction, such as 2%, 3%, 5%, 10%. Additionally, the number of clusters for each dataset varies as they contain different types of data. Users can choose the number of clustering centers for the clustering task, with options of 3, 5, 7 or 14 based on the number of models. The most common practice is to cluster the data into three classes (i.e., irregular, linear and periodic), which is set as the default.

Finally, in the model selection task, the user can select the specified classifier, such as SVM [42], XGBoost [50], RandomForest [40], LightGbm [27], KNN [41] and CNN [31]. In addition, the system provides configuration options for the model selection strategy, including efficiency priority and accuracy priority. The corresponding ranked top-K models based on the selected strategy can be written into the configuration file.

Offline training trains a classifier and records the best-performing model for each type of time series data. In the following, we will introduce each part of the offline training.

4.2 Data processing and model training

We proceed to provide the details of the data pre-processing steps and model training. Here, we optimize the TS2Vec model and set up the deep learning model in preparation for the subsequent steps.

[1]https://help.aliyun.com/document_detail/128379.html

Table 1: Parameters setting on the system page

Paramaters	Descriptions	Values
Metric_Name	The metric name of AliCloud database monitoring metrics	Mysql.cpu_usage, Mysql.disk_usage, Mysql.iops_usage, slow_queries, mem_usage, ...
Train_Data_Size	Size of training sample set (proportion in total sample size)	10%, 20%, 30%, 50%, 70%, 100%
Train_ML_Models_List	Machine learning models involved in training	ETS, Prophet, NPTS, ARIMA, Holt-Winters, Period, Linear
Train_DL_Models_List	Deep learning models involved in training	LSTM, Transformer, DeepAR, DeepFactor, DeepState, GPForecaster, NBeats
Auto_Tune_Paramaters	Whether to enable the automatic parameter setting function of the training models	True, False
Predict_Window_Len	Window size setting for time series prediction task (proportion in total time series length)	2%, 3%, 5%, 10%
N_Clusters	The number of clustering centers for a clustering task	3, 5, 7, 14
Classifier	A classifier that performs a sorting task	SVM, XGBoost, RandomForest, LightGbm, KNN and CNN
Selection_Strategy	A selection strategy of model selection task	Efficiency, Accuracy
Top-K	The Number of models selected	1, 2, 3

Data pre-processing. To prepare the source data for training, particularly for long time series data, we split it into shorter segments. This is because long time series data may contain invalid information that can negatively impact model accuracy. For example, if a piece of time series data contains one year's information, it is too long for the time series forecast task to achieve high accuracy. We can cut the piece of time series data across one year into twelve pieces (i.e., time series data across months) for training. In addition, beforing the time series forecasting task, some common pre-processing operations should be carried out, such as filling in the blank values and then smoothing the data. In SimpleTS framework, before executing self-developed linear and period algorithms, we use Kalman filtering [22] to smooth the source data in order to prevent outliers from affecting the shape of the source data and avoid leading to wrong prediction results.

TS2Vec model training. After data pre-processing, we prepare a TS2Vec [52] model for the following data classification. When training TS2Vec model, we found that different data sets have different relationships between instances. The original TS2Vec framework calculates loss function using the sum of temporal contrastive loss and instance-wise contrastive loss, which is not appropriate for all the datasets. On the contrary, for datasets with stronger relationships between instances, increasing the weight of the instance-wise contrastive loss function can improve the accuracy of prediction. Otherwise, it can be decreased. Motivated by this, we propose the weighted representation learning. The overall loss is redefined as:

$$\mathcal{L}_{dual} = \frac{1}{NT} \sum_{i} \sum_{t} \left((1-w)\ell_{temp}^{(i,t)} + w\ell_{inst}^{(i,t)} \right) \tag{4}$$

where w represents the weight of instance-wise contrastive loss, while $\ell_{temp}^{(i,t)}$ and $\ell_{inst}^{(i,t)}$ are computed via Eqs. 1 and 2 respectively. Note that, the weight w is manually set, the larger association between instances, the larger value of w will be set. We evaluate the influence of w in Section 6.3.

Deep learning (DL) model Training. Next, we pre-train deep learning models based on user-selected parameters, including 'Train_Data_Size', 'Train_DL_Models_List', 'Train_ML_Models_List', 'Predict_Window_Len' and 'Auto_Tune_Parameters'. Given the automatic parameter tuning function enabled('Auto_Tune_Parameters' is true), the parameters of each model are tuned according to the configurations shown in Table 2. Otherwise, the models are trained using the default parameters (indicated in bold in the table).

4.3 AutoML Label-Tagging

We proceed to present a comprehensive overview of the AutoML Label-Tagging.

Given a 'Train_Data_Size' (e.g., 10%), we randomly sample the training set to feed into different models. SimpleTS utilizes the selected 'Train_ML_Model_List' and 'Train_DL_Model_List' for label-tagging. Algorithm 1 presents the pseudo-code of offline training, which takes four parameters as inputs, including time series data TS_l, the target variable of prediction $target\_var$ (i.e., 'Metric_Name'), the window of prediction $window$, and the granularity of prediction $gran$. For example, for the prediction task of a database metric Mysql.cpu_usage, DAS has millions of Mysql.cpu_usage data. Thus, we randomly sample 'Train_Data_Size' pieces of data as the training set. As the indicator data of the database has strong timeliness, we select the data of the recent 10 days (i.e., 240 hours). The window of historical data for prediction is 240*0.9, and the prediction window is 240*0.1 (i.e., Predict_Window_Len is 10%). All of these windows are with the granularity of hours. The target variable for prediction is set to Mysql.cpu _usage. After that, we perform a prediction task on all models for each piece of data in the training set (lines 5–16). When a piece of data has performed the prediction task on all the models, the name, accuracy, precision, and SMAPE of the best model, prediction results, and the execution time of each model are recorded and stored in the database (lines 17–21). In order to facilitate the comparison between artificial feature extraction and representation learning, we also extract artificial features and save them in the database, including turning point, variance, periodicity, stationarity, max, min, mean, absolute variance, etc.

Table 2: Hyperparameters for various algorithms from GluonTS

Pipeline Methods	Hyperparameter 1	Hyperparameter 2
DeepAR	num_cells = [20, 30, **40**, 50, 60]	num_layers = [1, **2**, 3, 4, 5]
LSTM	num_cells = [20, 30, **40**, 50, 60]	num_layers = [1, **2**, 3, 4, 5]
Transformer	inner_ff_dim_scale = [2, **4**, 6, 8, 10]	num_heads = [4, 6, **8**, 10, 12]
Prophet	changepoint_prior_scale = [0.003, **0.03**, 0.1, 0.2, 0.5]	changepoint_range = [0.3, 0.4, **0.5**, 0.6, 0.7]
DeepFactor	cell_type = [**'lstm'**, 'gru']	num_factors = [6, 8, **10**, 12, 14]
DeepState	num_cells = [20, 30, **40**, 50, 60]	num_layers = [1, **2**, 3, 4, 5]
GPForecaster	max_iter_jitter = [5, **10**, 15, 20, 25]	N/A
NBeats	loss_function = ['SMAPE', 'MASE', **'MAPE'**]	N/A
ARIMA	information_criterion = [**'aic'**, 'bic', 'hqic','oob']	seasonal_test = [**'ocsb'**, 'ch']
Holt-Winters	seasonal_periods = [2, **4**, 6, 8, 10]	seasonal = ['add', 'mul', **'additive'**, 'multiplicative', None]

Table 3: Metrics used for model clustering features

Features	Formula	Descriptions
Std	$\frac{\sum(Y_i-\hat{Y}_i)}{N}$	Absolute Standard Deviation
Var	$\sqrt{\frac{\sum\left((Y_i-\hat{Y}_i)-Std\right)^2}{N-1}}$	Absolute Variance
Med	$MEDIAN(Y_i-\hat{Y}_i)$	Absolute Median
25-quartile	$PERCENTILE(Y_i-\hat{Y}_i, 25)$	First Quartile Deviation
75-quartile	$PERCENTILE(Y_i-\hat{Y}_i, 75)$	Third Quartile Deviation
MAE	$\frac{\sum\lvert Y_i-\hat{Y}_i\rvert}{N}$	Mean Absolute Error
MAPE	$\frac{1}{N}\cdot\sum\frac{\lvert Y_i-\hat{Y}_i\rvert}{Y}$	Mean Absolute Percentage Error
SMAPE	$\frac{100\%}{n}\sum\frac{\lvert\hat{Y}_i-Y_i\rvert}{\left(\lvert\hat{Y}_i\rvert+\lvert Y_i\rvert\right)/2}$	Symmetric Mean Absolute Percentage Error
WMAPE	$\frac{\sum(Y*MAPE)}{\sum Y}$	Weighted Mean Absolute Percentage Error

4.4 Soft Labeling and Clustering

As the number of prediction methods for time series data increases, many of them have similar performance on the same type of time series, which degrades the performance of the classifier. Hence, to avoid it, we introduce smoothed labels. Soft Labeling combines the information of the label distribution and replaces traditional one-hot encoded label vectors with weighted vectors, which further improves the classification quality.

The smoothed distribution of the labels is equivalent to adding noise to the real distribution, preventing the model from being over-confident in the correct labels, so that the difference between the output values of the predicted positive and negative samples is small. Studies [17, 30] show that soft labels encourage the representations of training examples from the same class to groups in tight clusters, thus avoiding over-fitting and making the model more robust.

We propose a new weighted soft labeling as an enhanced component of the classifier, and redesign the loss function, which takes the relevance of labels into account. The loss function is defined as:

$$y_{\text{soft}_i} = \begin{cases} 1-\omega_i\left(\frac{K-1}{K}\right), & \text{if } y_{\text{hot}_i} = \text{target} \\ \frac{\omega_i}{K}, & otherwise \end{cases} \tag{5}$$

where K is the number of labels, $\omega_i = \frac{\text{precision}_i}{\sum_{j=1}^{K}\text{precision}_j}$, i denotes the index of y_{hot} encoded by one-hot, target denotes the encoding of the target classification type.

Additionally, clustering is an effective solution to further improve the accuracy of classification. This is because the time series data of the same type behave similarly in different models. For example, when dealing with periodic time series data, Transformer [49] and DeepAR [14] models tend to yield poor results, while Prophet [43] and Holt-Winters [4] models demonstrate superior performance. Based on the prediction results, accuracy, and precision of each model stored in the database, several metrics are calculated for the clustering task as shown in Table 3, where $\hat{Y}$ is the predicted value, Y represents the actual value, and N is the size of the prediction window. In this way, we can group models that have similar performance into clusters, where the number of clusters can be configured (i.e., 'N-Clusters'). For instance, SimpleTS leverages expert knowledge to split all the models into three types: irregular, linear,

Algorithm 1: AutoML Label-Tagging

Input: Time series data TS_l, target_variable $target\_var$, granularity $gran$, prediction window $window$
Output: the status of whether saving to the database

$T_{history} \leftarrow Len(TS\_l) * window$;
$T_{predict} \leftarrow Len(TS\_l) * (1\text{-}window)$;
method_list ←[*lstm,..., hot-winters*];
results, detail_acc, detail_precision, detail_smape = ∅
foreach *method in method_list* **do**
  predict_results = invoke_method(method);
  accuracy = cal_accuracy(predict_results, $T_{predict}$);
  precision= cal_precision(predict_results, $T_{predict}$);
  smape = cal_smape(predict_results, $T_{predict}$);
  detail_acc ← detail_acc.add(method, accuracy);
  detail_precision← detail_precision.add(method, precision);
  detail_smape ← detail_smape.add(method, smape);
  if *compare manual feature* **then**
    cal_manual_feature(TS_l)
  end
end
best_*model* = max(detail_accuracy.values()).key();
best_*accuracy* = max(detail_accuracy.values());
best_*precision* = max(detail_precision.values());
best_*smape* = min(detail_smape.values());
return save_to_database(instance_id, ...,detail_smape)

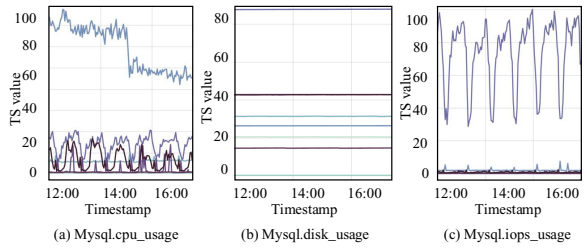

Figure 6: Time series shapes of three private datasets

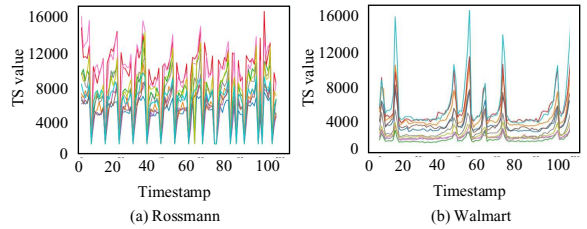

Figure 7: Time series shapes of Rossmann and Walmart

and periodic, by setting the number of clusters to three. According to the configured 'Selection_Strategy', we search for the highest accurate model or the model with the least training time for each time series type, and record this model's name in the configuration file. When performing online forecasting, we can search the configuration file for the best model in terms of accuracy or efficiency based on the time series type.

4.5 Pipeline Method Classifier

Finally, we continue to train a classifier. We use the pre-trained weighted representation learning model to encode the original data and feed the encoded data into the classifier. As TS2Vec [52] model is already trained to encode the training data into 320 dimensions, we are able to represent the target variable in the high-dimension space. We replace the source label that represents the best performance model with the clustered labels. For example, assume the model with the best performance in a piece of time series data is Transformer [49]. After clustering, it is found that Transformer belongs to the 'irregular' cluster, thus we will use 'irregular' as the label of this data for classification. In order to balance samples, we up-sample the training data according to the number of samples in each label to ensure that the number of each label is similar. After that, the encoded source data and the clustered labels are fed to train a classifier (i.e., 'Classifier'). This classifier will be further used for online forecasting.

5 ONLINE FORECASTING

For the online prediction task, we first identify the type of the new time series data, and then find the corresponding model for the prediction task. In general, we divide online forecasting into three steps as below:

- First, when a piece of new data arrives, we load the pre-trained weighted representation learning model and encode the source data to obtain its high-dimensional representation.
- Second, we classify this encoded data into a certain type by loading the offline-trained classifier model. After obtaining the result of the classification, we can choose the method with the best accuracy or the least execution time from the configuration file according to the user's configuration.
- Third, according to the method selected from the configuration file, we invoke the corresponding time series forecasting model to perform the prediction task.

In addition, in order to support large-scale data prediction tasks, we execute the online prediction tasks in a parallel manner to improve its efficiency.

6 EXPERIMENTS

In this section, we aim to evaluate the performance of SimpleTS compared with the state-of-the-art methods based on 55 datasets.

6.1 Experimental Setting

Datasets. We use 55 datasets, three private datasets from DAS, Rossmann dataset[2] and Walmart dataset[3] from Kaggle, and 50 UCR datasets[4].

- **Alibaba datasets**. The private datasets come from the DAS in AliCloud. DAS needs to monitor the usage of various types of databases (e.g., Mysql and PostSql) from millions of machines (instances) every day. For Mysql database, DAS needs to monitor over 60 indicators, including the usage of cpu, disk, and iops usage (Mysql.cpu_usage, Mysql.disk_usage, Mysql.iops_usage), which indicate the running status of the database. In the following experiments, we use the datasets of these three indicators as the private datasets for experimental verification. Due to time constraints in the experiments, we restrict the test dataset for each metric to 50,000 instances (each instance has dimension of 240). Figure 6 depicts the shapes of three private datasets.
- **Public datasets**. They are from Walmart, Rossmann, and UCR, and Figure 7 depicts the shapes of the first two.
  - **Walmart**. The dataset includes 421,570 sales records from 45 Walmart stores, covering the period from February 5th 2010 to November 1st 2012. It has eight columns, of which we only use 'Weekly_sales' column to do the prediction task.
  - **Rossmann**. This dataset provides 1,017,209 historical sales data of 1,115 Rossmann stores from January 1st 2013 to July 1st 2015. Each data contains 27 columns, but we only utilize the 'Sales' column for performing predictions.
  - **UCR**. It contains 128 univariate time series (UTS) datasets, which are collected from different domains such as finance, commerce, power, picture, hydrology, agriculture, etc. We choose 50 time series datasets with obvious time series shapes, excluding picture data. The number of data samples in each UCR dataset varies from 108 to 5,314.

Prepocessing. Some of the experimental datasets do not contain timestamps. To impute the timestamps, we re-generate the timestamps in terms of hour granularity if less than 1,000 pieces of time series data exist; otherwise, the timestamps are generated in terms of 1-day granularity. For public datasets, we set the history

[2]https://www.kaggle.com/c/rossmann-store-sales/data
[3]https://www.kaggle.com/datasets/yasserh/walmart-dataset?resource=download
[4]http://www.timeseriesclassification.com/index.php

Table 4: Comparisons of baselines and SimpleTS

(a) Mysql.cpu_usage

Method	ARIMA	DA	ETS	HW	Linear	LSTM	NPTS	Period	Prophet	TF	DS	DF	GPF	NB	GB	AF	AutoAI-TS	SimpleTS
train_time(s)	/	29.612	/	/	/	33.083	/	/	/	25.842	829.165	10.812	8.221	61.889	5872.903	184123.754	16241.737	**3407.086**
predict_time(s)	0.501	0.504	0.457	0.236	0.492	0.496	0.124	0.368	0.518	0.512	0.813	**0.102**	0.182	0.205	0.283	0.495	0.414	0.193
accuracy	0.623	0.306	0.741	0.622	0.722	0.613	0.768	0.687	0.661	0.729	0.697	0.215	0.174	0.646	0.762	0.770	0.764	**0.793**
precision	0.487	0.437	0.634	0.479	0.616	0.248	0.672	0.554	0.525	0.598	0.666	0.102	0.023	0.553	0.573	0.739	0.632	**0.773**
SMAPE	0.399	1.139	0.301	0.419	0.317	0.535	0.237	0.341	0.381	0.271	0.316	1.117	1.438	0.447	0.276	0.359	0.283	**0.217**

(b) Mysql.disk_usage

Method	ARIMA	DA	ETS	HW	Linear	LSTM	NPTS	Period	Prophet	TF	DS	DF	GPF	NB	GB	AF	AutoAI-TS	SimpleTS
train_time(s)	/	30.572	/	/	/	32.267	/	/	/	23.194	836.665	11.501	9.494	70.898	6297.091	190823.349	16400.331	**4013.729**
predict_time(s)	0.488	0.501	0.487	0.220	0.493	0.497	**0.125**	0.419	0.584	0.495	0.743	0.133	0.182	0.201	0.479	0.513	0.694	0.471
accuracy	0.956	0.914	0.995	0.978	0.995	0.902	0.988	0.987	0.992	0.959	0.962	0.346	0.569	0.963	0.995	0.992	0.979	**0.996**
precision	0.960	0.996	0.999	0.982	0.999	0.996	0.991	0.988	0.996	0.998	0.987	0.121	0.643	0.995	0.999	0.991	0.982	**0.999**
SMAPE	0.084	0.090	0.005	0.022	0.005	0.093	0.012	0.013	0.008	0.041	0.037	0.931	0.551	0.037	0.004	0.060	0.019	**0.003**

(c) Mysql.iops_usage

Method	ARIMA	DA	ETS	HW	Linear	LSTM	NPTS	Period	Prophet	TF	DS	DF	GPF	NB	GB	AF	AutoAI-TS	SimpleTS
train_time(s)	/	30.390	/	/	/	32.072	/	/	/	23.490	902.634	13.274	9.015	78.731	5735.913	180970.143	11500.701	**3988.554**
predict_time(s)	0.488	0.500	0.463	0.229	0.499	0.499	0.123	0.281	0.514	0.494	0.539	0.149	**0.093**	0.129	0.283	0.473	0.414	0.231
accuracy	0.385	0.464	0.488	0.409	0.488	0.456	0.501	0.491	0.422	0.490	0.466	0.118	0.174	0.239	0.532	0.511	0.552	**0.576**
precision	0.264	0.281	0.366	0.287	0.365	0.240	0.424	0.340	0.295	0.322	0.412	0.059	0.044	0.132	0.473	0.485	0.392	**0.518**
SMAPE	0.825	0.847	0.683	0.845	0.667	0.853	0.557	0.672	0.825	0.794	0.716	1.453	1.434	1.302	0.476	0.386	0.296	**0.224**

(d) Walmart

Method	ARIMA	DA	ETS	HW	Linear	LSTM	NPTS	Period	Prophet	TF	DS	DF	GPF	NB	GB	AF	AutoAI-TS	SimpleTS
train_time(s)	/	10.959	/	/	/	10.451	/	/	/	11.101	86.333	4.992	3.956	62.373	583.836	1145.751	991.408	**327.126**
predict_time(s)	0.301	0.074	0.366	0.075	0.733	0.076	**0.016**	0.394	0.428	0.105	1.785	0.381	0.044	0.153	0.298	0.311	0.058	0.372
accuracy	0.924	0.882	0.923	0.883	0.919	0.886	0.887	0.912	0.889	0.900	0.886	0.746	0.235	0.903	0.915	0.908	0.888	**0.922**
precision	0.949	0.823	0.958	0.854	0.938	0.853	0.864	0.909	0.858	0.873	0.956	0.804	0.006	0.981	0.939	**0.981**	0.851	0.965
SMAPE	**0.074**	0.109	0.075	0.110	0.080	0.106	0.108	0.086	0.107	0.097	0.111	0.428	1.311	0.101	0.082	0.089	0.112	0.103

(e) Rossmann

Method	ARIMA	DA	ETS	HW	Linear	LSTM	NPTS	Period	Prophet	TF	DS	DF	GPF	NB	GB	AF	AutoAI-TS	SimpleTS
train_time(s)	/	15.628	/	/	/	16.163	/	/	/	16.006	88.735	6.716	5.438	62.634	1951.223	14260.034	1085.087	**662.483**
predict_time(s)	0.511	0.080	0.398	0.090	0.355	0.080	**0.026**	0.663	0.415	0.106	21.851	0.516	0.592	1.477	0.149	0.158	0.145	0.238
accuracy	0.594	0.660	0.616	0.521	0.580	0.687	0.626	0.649	0.655	0.701	0.676	0.422	0.461	0.763	0.682	0.541	**0.741**	0.737
precision	0.435	0.621	0.480	0.440	0.426	0.667	0.499	0.580	0.584	0.681	0.414	0.380	0.264	0.643	0.581	0.624	0.423	**0.793**
SMAPE	0.618	0.522	0.575	0.811	0.639	0.487	0.565	0.538	0.524	0.466	0.396	0.813	0.766	**0.278**	0.476	0.531	0.317	0.375

window and the prediction window at 9:1 ratio of the original data. In particular, for datasets with less than 1,000 pieces of time series data, we split a time series data into multiple sub-series data to expand the number of training data.

Performance metrics. For the time series forecasting task, we use the accuracy, precision, symmetric mean absolute percentage error (SMAPE), the prediction of each data (denoted as predict_time), and the total training time of 50,000 training data (denoted as train_time) as the performance metrics. When calculating the precision, we set the tolerance as 3%, considering it is correct if the fluctuation between the predicted result and the ground truth is within the range of $[-1.5\%, 1.5\%]$. For the classification task, we use accuracy and precision as the performance metrics. All the experiments were conducted on a machine with Intel(R) Xeon(R) CPU E5-2682 with 24 cores and 96GB of memory. The AutoML-label tagging was conducted via the AliCloud Function Compute[5].

Parameter Setting. For offline training tasks, we can customize the training parameters through the interface shown in Figure 8. In the following experiments, we set 'Train_Data_Size' to 20%, 'Train_ML_Moldels_List' to {ETS [2], Prophet [43], NPTS [5], Holt-Winters [4], ARIMA [45], Period and Linear}, 'Train_DL_Moldels_List' to {LSTM [21], Transformer [49], DeepAR [14], DeepFactor [9], DeepState [35], GPForecaster [8], NBeats [32]}, 'Auto_Tune_ Paramaters' to false, 'N_Clusters' to 3, 'Classifier' to lightGBM, 'Selection_Strategy' to accuracy, 'Top-K' to 1.

Comparison Algorithms. For the time series forecasting task, we compare our framework SimpleTS with selected fourteen algorithms including LSTM, Transformer (TF), DeepAR (DA), ETS, Prophet, NPTS, ARIMA, Holt-Winters (HW), DeepState (DS), DeepFactor (DF), GPForecaster (GPF), NBeats (NB), Period and Linear, as well as three frameworks, namely Global Best (GB), AutoForecast (AF) [6] and AutoAI-TS [36]. Here, 'Global Best' trained fourteen pipeline methods to get the best model for online prediction. Except for the self-developed methods and three baseline frameworks, other methods are implemented through the gluonTS package[6]. When training the models in 'Train_DL_Models_List', we set the number of epochs to 5, learning_rate to 0.001, num_batches_per _epoch to 100, and use other default hyper-parameters in Table 2. To ensure a fair comparison benchmark for non-experts, we refrained from optimizing any parameters of deep learning algorithms when running three baseline frameworks or SimpleTS. We believe that

[5]https://help.aliyun.com/product/50980.html

[6]https://ts.gluon.ai/stable/api/gluonts/gluonts.html

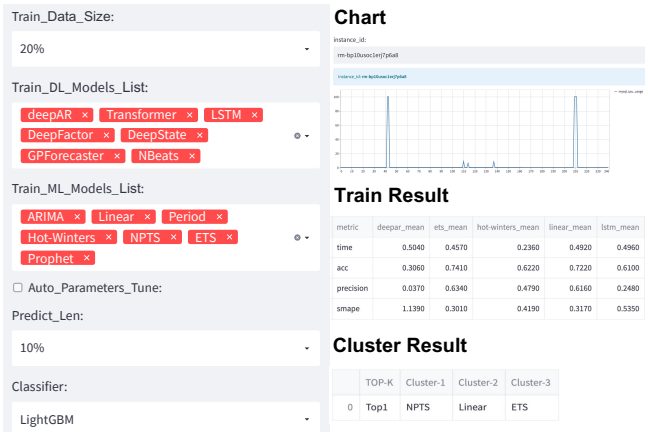

Figure 8: Experimental interface parameter configuration

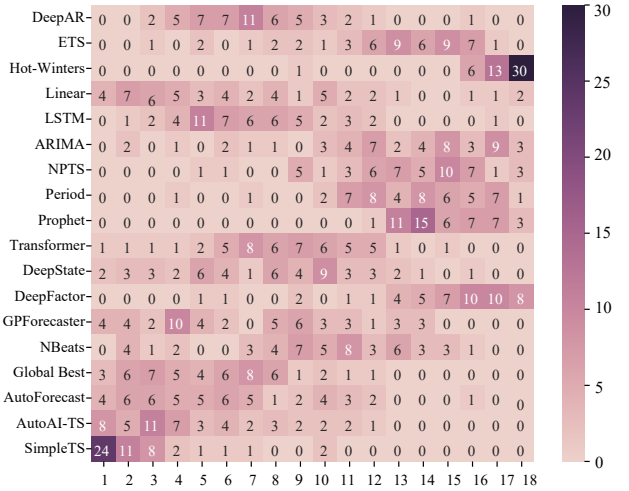

Figure 9: Forecasting accuracy based ranking of SimpleTS and SOTA toolkits for 50 UCR datasets

with the improvement of each algorithm's accuracy, the overall predictive performance will correspondingly enhance.

6.2 Performance of SimpleTS and Competitors

We compare the forecasting performance of SimpleTS and all the competitors on private and public datasets. Table 4 reports the results on three private datasets and two public datasets, while the results of 50 UCR datasets are provided in Figure 9.

Compared with fourteen pipeline methods. As shown in Figures 6 and 7, Mysql.cpu_usage, Mysql.iops_usage and Rossmann are more complex, while Mysql.disk_usage and Walmart are simple in terms of shape. Therefore, all of the models perform well in Mysql.disk_usage and Walmart, but perform worse on other datasets. SimpleTS performs better than fourteen single prediction models in most cases in terms of accuracy, precision, and SMAPE. This is because, SimpleTS can choose the best model with optimal parameters among various prediction models. However, deep learning models (i.e., DeepAR, LSTM, Transformer, DeepState, DeepFactor, GPForecaster, NBeats) have less training time compared to SimpleTS, while others require training online. This is because these deep learning methods are trained by gluonTS that is especially developed for speeding up model building. Note that, although all the single prediction models have less training time and some of them (e.g., NPTS) have less prediction time, their prediction results are much poor compared with our SimpleTS, especially for complex time series types.

Compared with three baseline frameworks. First, SimpleTS has the least training time compared with Global Best, AutoForecast, and AutoTS. 'Global Best' is a simple method to train the entire training dataset on all the single prediction models, thus, it takes more training time than SimpleTS. AutoAI-TS takes multiple sampling, training, and ranking iterations to find the best model, which leads to a long training time than 'Global Best' and SimpleTS. AutoForecast takes the longest training time. It trains the model via feature extraction of every data. Among them, more than 800 meta-features are extracted, but its effect is not as good as SimpleTS, which also explains the limitations of artificial features.

Second, although the training time for SimpleTS is the least, its performance in terms of accuracy, precision and SMAPE is better than Global Best, AutoForecast, and AutoTS in most cases. The main reason is that SimpleTS considers the effect of different time series types on selecting the best model, while the others ignore it. However, on Rossman, AutoAI-TS performs better than SimpleTS in terms of accuracy and SMAPE. This is because, when a dataset contains a large number of irregular timing data, the deep models can well capture the relationship between these irregular timing data. On Walmart dataset, AutoForecast outperforms SimpleTS in terms of precision and SMAPE. This is because AutoForecast is capable of effectively capturing information by extracting artificial features on datasets with simple rules. However, when there are a large number of periodic or linear time series data in the training set, the prediction accuracy of the deep learning model is not as good as that of the machine learning prediction models (e.g., ARIMA and Prophet specifically designed for periodic or linear time series data). Hence, SimpleTS is more suitable for the dataset with mixed time series types. In general, compared with the three baseline frameworks, SimpleTS not only greatly reduces the training time, but also achieves high prediction quality.

Due to the limited space, we only report the accuracy results for 50 public UCR dataset in Figure 9; we have similar observations for the other performance metrics. The ranking of SimpleTS across 50 datasets is shown at the bottom, with the numbers 24, 11, and 8 indicating the number of times it ranked 1, 2, and 3, respectively. As expected SimpleTS ranks 1 in most cases. Besides, SimpleTS consumes less training time than Global Best, AutoForecast and AutoAI-TS, as confirmed in Table 4.

6.3 Parameter Study

In this subsection, we evaluate the effect of parameter w on the effectiveness of weighted representation learning.

Based on Equation 4, we adjust the parameter w to calculate the loss function to train the TS2Vec [52] model. The performance

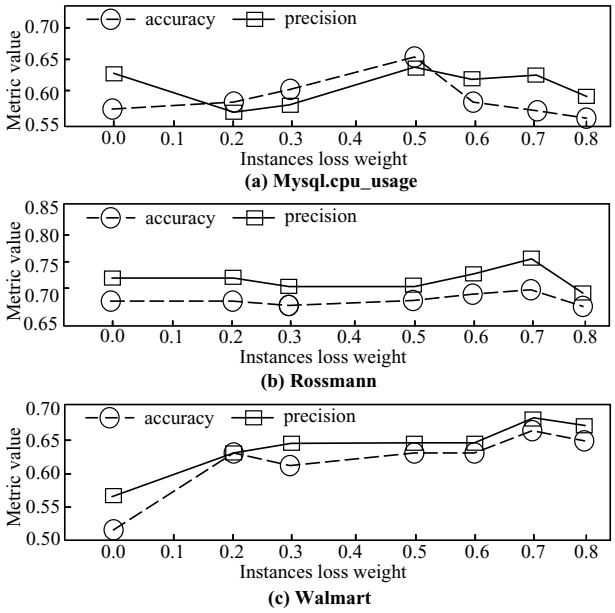

Figure 10: Performance vs. instances loss weight

of the data encoded by the trained TS2Vec model on the classifier is used to evaluate whether the weighted representation learning is effective. We do comparative experiments on Mysql.cpu_usage, Rossmann, and Walmart datasets, which have different relations between instances. We vary the instances loss weight from 0 to 0.8. Figure 10 depicts the corresponding results. On Mysql.cpu_usage dataset, the accuracy and precision reach the peak when the weight of instances loss is set to 0.5. However, on Rossmann and Walmart datasets, the accuracy and precision reach the peak when the weight is set to 0.7. As shown in Figures 6 and 7, the shapes of different instances on Mysql.cpu_usage differ, but are similar on Rossmann and Walmart datasets. That is the reason why the best loss wight of Rossmann and Walmart datasets is higher than the Mysql.cpu_usage. Besides, on Walmart dataset, we find that when the weight is 0, the accuracy and precision are much lower than others. This is because, the shape of Walmart is irregular, and it is difficult for the model to capture all the information only using the timestamp loss. However, as the weight of instances loss increases, the performance first improves and then drops. That also explains the necessity of combining the timestamp loss and instances loss, and a too-large weight will weaken the effect of timestamp loss resulting in performance degradation.

6.4 Ablation Study

In this subsection, we evaluate each technique (including representation learning, soft labeling, and clustering) used in SimpleTS in terms of the accuracy and precision of the classifier.

Representation learning evaluation. For comparing the performance of artificial feature engineering and representation learning, we calculate the artificial feature engineering based on expert experience during offline training (shown in Fig. 3). We then use these artificial features to train the classifier with the labels of clustering results.

Table 5: Comparison of artificial feature engineering and representation learning

dataset_name	N_Clusters	artificial feature		representation learning	
		accuracy	precision	accuracy	precision
Mysql.cpu_usage	3	53.12%	41.35%	**64.49%**	**65.24%**
	7	40.72%	31.24%	**47.24%**	**37.28%**
	14	36.62%	26.12%	**36.62%**	**30.06%**
Mysql.iops_usage	3	53.23%	37.28%	**62.43%**	**57.34%**
	7	42.19%	26.12%	**47.02%**	**49.14%**
	14	35.69%	30.06%	**40.20%**	**38.91%**
Mysql.disk_usage	3	72.68%	73.42%	**76.83%**	**74.67%**
	7	64.09%	63.56%	**64.60%**	**69.13%**
	14	54.39%	47.29%	**59.68%**	**56.84%**

Table 6: Comparisons of 'with cluster' and 'without cluster'

	Mysql.cpu_usage		Mysql.disk_usage		Mysql.iops_usage	
cluster (w/o)	w	o	w	o	w	o
predict_time (s)	**0.193**	0.434	**0.471**	0.661	**0.231**	0.615
accuracy	**0.793**	0.736	**0.996**	0.992	**0.576**	0.489
precision	**0.773**	0.714	**0.999**	0.936	0.518	**0.549**
SMAPE	**0.217**	0.291	**0.003**	0.007	**0.224**	0.391

We compare the accuracy and precision of architecture features and representation learning on the classifier for three private datasets. Besides, we evaluate their performance on both three-classification, seven-classification and fourteen-classification problems. The experimental results are reported in Table 5.

As observed, representation learning outperforms artificial feature engineering in all three classification problems. When the shape of the time series is complex to be represented by artificial features, the advantage of representation learning is more obvious. Specifically, on Mysql.cpu_usage (shown in Fig. 6(a)) and Mysql.iops_usage (shown in Fig. 6(c)) datasets, the accuracy and precision of representation learning is about 10%-20% better. However, the performance on Mysql.disk_usage dataset is not outstanding because the dataset is almost linear (shown in Fig. 6(b)), which is much easier for artificial feature engineering to capture the features.

Soft labeling evaluation. To evaluate soft labeling, we apply Equation 5 on the one-hot expression results output by various classifiers, including both deep neural network classifiers (such as CNN [31]) and traditional machine learning classifiers (such as LightGbm [27]). We use private and public benchmark datasets (DAS and Rossmann) with multi-class classification tasks. Figure 11 provides the corresponding results on classifiers (including precision and accuracy). As observed, our optimization technique (i.e., soft labeling) works for different classifiers and different datasets, especially for data sets where labels are associated with each other. This is because hard labels could ignore the similarity between true-positives labels and other labels, thus omitting a lot of useful information, and a multi-class classifier can often be improved significantly by using weighted soft labels.

Clustering evaluation. During offline training, we use clustering to improve the accuracy of classification. To evaluate the effective of clustering, we set the number of clusters N to three, and we classify the encoded data based on the clustered labels. If clustering is not used, we will directly classify the encoded data

into fourteen categories according to the number of models. We do comparative experiments on Mysql.cpu_usgae, Mysql.disk_usage, Mysql.iops_usage datasets. At the same time, we use four metrics (predict_time, accuracy, precision, SMAPE) to evaluate their prediction performance. Table 6 depicts the corresponding results. As observed, the classifier performs better in terms of most of performance metrics after using clustering. The prediction time of 'with cluster' is less than 'without cluster', as we use clustering to reduce the number of labels. Simultaneously, when the number of labels is reduced, the classifier quality also improves a lot.

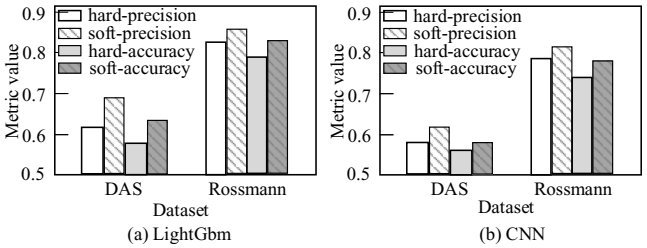

Figure 11: Performances of Soft labeling

7 RELATED WORK

7.1 Model selection

There are mainly two categories of model selection. The first category is to generate an optimal sub-model of a specified type of model, by selecting parameters or neural network structures of deep learning models [1, 3]. Google AutoML [3] develops two deep learning models: Wave-Net and Transformer, which enable users to automatically search the optimal combination of models' components as well as core hyperparameters. AWS forecast framework [1] considers local weather and holiday information to improve prediction accuracy. The second category is to identify the best model among all the candidate models. The mainstream of the category is AutoML based frameworks [37]. IBM presents AutoAI-TS [36], which encompasses three steps: sampling, training, and ranking. Mahdi et al. [7] address conditional hierarchical forecasting using machine learning based classification methods, which explore time series features to select the reconciliation method for each hierarchy. AutoForecast [6] takes statistical features, information-theoretic features, spectral features, and landmarker features of the source data into account. Hence, it can effectively capture the similarities between datasets and identify the corresponding optimal model by projecting meta-feature vectors.

The methods mentioned above, which belong to the second category, are subject to certain limitations. First, they cannot adapt to all kinds of time series data, and thus the accuracy may be low in some cases. Second, the process of using complex algorithms to calculate the similarity of the dataset is time-consuming during online prediction, which is infeasible for predicting a large amount of data. However, SimpleTS, which fits the second category, distinguishes time series by their types and identifies the optimal model according to the specific type. Hence, SimpleTS is able to select the optimal model efficiently and to achieve high prediction accuracy.

7.2 Representation learning

Unsupervised representation learning has been widely used in computer vision [47], natural language processing [16], speech recognition [51], time series analyzing [10, 15, 28, 44, 52, 53], etc. We review the studies of representation learning of time series.

SOM-VAE [15] presents a novel strategy to overcome the non-differentiability in discrete representation learning and a gradient-based version of the traditional self-organizing map algorithm. However, the information learned by SOM-VAE via representation learning is limited. A transformer-based framework [53] learns multivariate time series representation based on transformer encoder and applies the representation to time series regression and classification. TimeNet [28] trains an encoder jointly with a decoder that reconstructs the input signal according to its learned representations.

However, the above methods either cannot scale up to large time series or fail to model complex time series. To tackle these issues, TNC [44] proposes a self-supervised framework for exploring local smoothness of the signal to learn the generalized representation of time series. Eldele et al. [10] develop an unsupervised time series representation learning framework (TCC) based on time and context comparison. It facilitates the consistency of different data augmentations. However, both TNC and TCC can learn representations only at a certain semantic level and assume data has transformation-invariance, which limits them from being adapted to other scenarios. TS2Vec [52] proposes a robust contextual representation for each timestamp. However, TS2Vec ignores the weights of the instances and timestamps. Our weighted representation learning improves TS2Vec by taking the correlation between the instances into account when calculating the loss function.

8 CONCLUSION

This paper proposes an efficient and general model selection framework for time series forecasting. To the best of our knowledge, SimpleTS is the first framework that incorporates a classifier to distinguish types of time series for prediction. SimpleTS employs clustering and self-developed soft labeling. This way, the prediction accuracy is almost independent of the number of candidate prediction models. Moreover, we propose a weighted representation learning strategy for further improving the accuracy of the classifier. Finally, extensive experiments on 55 datasets show that SimpleTS outperforms seventeen baselines in terms of training time, accuracy, precision and SMAPE in most cases. Moreover, we validate the effectiveness of each component/technique enclosed in SimpleTS, including clustering, soft labeling and weighted representation learning. Specifically, compared with AutoForecast and AutoAI-TS, the training time is reduced by approximately 4-10 times. The accuracy of model prediction is improved on most of the training sets compared with the state-of-the-art methods. In future, we plan to adapt SimpleTS framework to other fields, including but not limited to anomaly detection and root cause localization.

ACKNOWLEDGMENTS

This work was supported by the NSFC under Grants No. (62102351, 62025206 and 61972338). Lu Chen is the corresponding author of the work.

REFERENCES

[1] 2020. AWS Auto-ML. https://docs.aws.amazon.com/managedservices/latest/userguide/forecast.html. 2022.11.01.

[2] 2020. Exponential Smoothing *ETS* Algorithm. https://docs.aws.amazon.com/forecast/latest/dg/aws-forecast-recipe-ets.html. 2022.11.01.

[3] 2020. Google Auto-ML. https://ai.googleblog.com/2020/12/using-automl-for-time-series-forecasting.html. 2022.11.01.

[4] 2020. Holt-Winters. https://otexts.com/fpp2/holt-winters.html. 2022.11.12.

[5] 2020. NPTS. https://docs.aws.amazon.com/forecast/latest/dg/aws-forecast-recipe-npts.html. 2022.11.01.

[6] Mustafa Abdallah, Ryan A. Rossi, Kanak Mahadik, Sungchul Kim, Handong Zhao, and Saurabh Bagchi. 2022. AutoForecast: Automatic Time-Series Forecasting Model Selection. In *CIKM*, Mohammad Al Hasan and Li Xiong (Eds.). 5–14.

[7] Mahdi Abolghasemi, Rob J. Hyndman, Evangelos Spiliotis, and Christoph Bergmeir. 2020. Model selection in reconciling hierarchical time series. *CoRR* abs/2010.10742 (2020).

[8] Giorgio Corani, Alessio Benavoli, and Marco Zaffalon. 2021. Time Series Forecasting with Gaussian Processes Needs Priors. In *ECML PKDD*. 103–117.

[9] Jan Czarnowski, Tristan Laidlow, Ronald Clark, and Andrew J. Davison. 2020. DeepFactors: Real-Time Probabilistic Dense Monocular SLAM. *IEEE* 5, 2 (2020), 721–728.

[10] Emadeldeen Eldele, Mohamed Ragab, Zhenghua Chen, Min Wu, Chee Keong Kwoh, Xiaoli Li, and Cuntai Guan. 2021. Time-Series Representation Learning via Temporal and Contextual Contrasting. In *IJCAI 2021*. 2352–2359.

[11] Christos Faloutsos, Valentin Flunkert, Jan Gasthaus, Tim Januschowski, and Yuyang Wang. 2019. Forecasting Big Time Series: Theory and Practice. In *KDD 2019*. 3209–3210.

[12] Matthias Feurer, Katharina Eggensperger, Stefan Falkner, Marius Lindauer, and Frank Hutter. 2020. Auto-Sklearn 2.0: The Next Generation. *CoRR* abs/2007.04074 (2020).

[13] Matthias Feurer, Aaron Klein, Katharina Eggensperger, Jost Tobias Springenberg, Manuel Blum, and Frank Hutter. 2019. Auto-sklearn: Efficient and Robust Automated Machine Learning. 113–134.

[14] Valentin Flunkert, David Salinas, and Jan Gasthaus. 2017. DeepAR: Probabilistic Forecasting with Autoregressive Recurrent Networks. *CoRR* abs/1704.04110 (2017).

[15] Vincent Fortuin, Matthias Hüser, Francesco Locatello, Heiko Strathmann, and Gunnar Rätsch. 2019. SOM-VAE: Interpretable Discrete Representation Learning on Time Series. In *ICLR 2019*. OpenReview.net.

[16] Tianyu Gao, Xingcheng Yao, and Danqi Chen. 2021. SimCSE: Simple Contrastive Learning of Sentence Embeddings. In *EMNLP 2021*. 6894–6910.

[17] Biyang Guo, Songqiao Han, Xiao Han, Hailiang Huang, and Ting Lu. 2021. Label Confusion Learning to Enhance Text Classification Models. In *AAAI 2021*. 12929–12936.

[18] Xiao He, Ye Li, Jian Tan, Bin Wu, and Feifei Li. 2023. OneShotSTL: One-Shot Seasonal-Trend Decomposition For Online Time Series Anomaly Detection And Forecasting. *Proceedings of the VLDB Endowment* 16, 06 (2023), 1399–1412.

[19] Xin He, Kaiyong Zhao, and Xiaowen Chu. 2021. AutoML: A survey of the state-of-the-art. *Knowl. Based Syst.* (2021), 106622.

[20] Van Long Ho, Nguyen Ho, and Torben Bach Pedersen. 2021. Efficient Temporal Pattern Mining in Big Time Series Using Mutual Information. *Proc. VLDB Endow.* 15, 3 (2021), 673–685.

[21] Sepp Hochreiter and Jürgen Schmidhuber. 1996. LSTM can Solve Hard Long Time Lag Problems. In *NIPS 1996*. MIT Press, 473–479.

[22] Ankur Jain, Edward Y. Chang, and Yuan-Fang Wang. 2004. Adaptive Stream Resource Management Using Kalman Filters. In *SIGMOD*, Gerhard Weikum, Arnd Christian König, and Stefan Deßloch (Eds.). ACM, 11–22.

[23] Tianyi Li, Lu Chen, Christian S Jensen, and Torben Bach Pedersen. 2021. TRACE: Real-time compression of streaming trajectories in road networks. *Proc. VLDB Endow.* 14, 7 (2021), 1175–1187.

[24] Tianyi Li, Lu Chen, Christian S Jensen, Torben Bach Pedersen, Yunjun Gao, and Jilin Hu. 2022. Evolutionary Clustering of Moving Objects. In *ICDE*. IEEE, 2399–2411.

[25] Tianyi Li, Ruikai Huang, Lu Chen, Christian S Jensen, and Torben Bach Pedersen. 2020. Compression of uncertain trajectories in road networks. *Proc. VLDB Endow.* 13, 7 (2020), 1050–1063.

[26] Yan Li and Tingjian Ge. 2021. Imminence Monitoring of Critical Events: A Representation Learning Approach. In *SIGMOD '21*. 1103–1115.

[27] lightgbm. 2019. lightgbm. https://lightgbm.readthedocs.io/. 2022.11.15.

[28] Pankaj Malhotra, Vishnu TV, Lovekesh Vig, Puneet Agarwal, and Gautam Shroff. 2017. TimeNet: Pre-trained deep recurrent neural network for time series classification. In *ESANN 2017, Bruges, Belgium, April 26-28, 2017*.

[29] Hui Miao, Ang Li, Larry S. Davis, and Amol Deshpande. 2017. On Model Discovery For Hosted Data Science Projects. In *SIGMOD, 2017*. 6:1–6:4.

[30] Rafael Müller, Simon Kornblith, and Geoffrey E. Hinton. 2019. When does label smoothing help?. In *NeurIPS, December 8-14, 2019, Vancouver, BC, Canada*.

[31] Lin Ning, Hui Guan, and Xipeng Shen. 2019. Adaptive Deep Reuse: Accelerating CNN Training on the Fly. In *ICDE*. IEEE, 1538–1549.

[32] Boris N. Oreshkin, Dmitri Carpov, Nicolas Chapados, and Yoshua Bengio. 2020. N-BEATS: Neural basis expansion analysis for interpretable time series forecasting. OpenReview.net.

[33] Lorenzo Perini, Connor Galvin, and Vincent Vercruyssen. 2020. A Ranking Stability Measure for Quantifying the Robustness of Anomaly Detection Methods. In *ECML PKDD*, Vol. 1323. 397–408.

[34] Alireza Pooya, Morteza Pakdaman, and Lotfi Tadj. 2019. Exact and approximate solution for optimal inventory control of two-stock with reworking and forecasting of demand. (2019), 333–346.

[35] Syama Sundar Rangapuram, Matthias W. Seeger, Jan Gasthaus, Lorenzo Stella, Yuyang Wang, and Tim Januschowski. 2018. Deep State Space Models for Time Series Forecasting. In *NeurIPS 2018*. 7796–7805.

[36] Syed Yousaf Shah, Dhaval Patel, Long Vu, Xuan-Hong Dang, Bei Chen, Peter Kirchner, Horst Samulowitz, David Wood, Gregory Bramble, Wesley M. Gifford, Giridhar Ganapavarapu, Roman Vaculín, and Petros Zerfos. 2021. AutoAI-TS: AutoAI for Time Series Forecasting. In *SIGMOD '21*. 2584–2596.

[37] Vraj Shah, Jonathan Lacanlale, Premanand Kumar, Kevin Yang, and Arun Kumar. 2021. Towards Benchmarking Feature Type Inference for AutoML Platforms. In *SIGMOD '21*. 1584–1596.

[38] Zeyuan Shang, Emanuel Zgraggen, Benedetto Buratti, Ferdinand Kossmann, Philipp Eichmann, Yeounoh Chung, Carsten Binnig, Eli Upfal, and Tim Kraska. 2019. Democratizing Data Science through Interactive Curation of ML Pipelines. In *SIGMOD 2019*. 1171–1188.

[39] Radwa El Shawi, Mohamed Maher, and Sherif Sakr. 2019. Automated Machine Learning: State-of-The-Art and Open Challenges. *CoRR* abs/1906.02287 (2019).

[40] sklearn. 2019. randomforest. https://scikit-learn.org/stable/modules/generated/sklearn.ensemble.RandomForestClassifier.html#sklearn.ensemble.RandomForestClassifier/. 2022.11.15.

[41] sklearn. 2019. sklearn.knn. https://scikit-learn.org/stable/modules/generated/sklearn.neighbors.KNeighborsClassifier.html#sklearn.neighbors.KNeighborsClassifier. 2022.11.15.

[42] sklearn. 2019. sklearn.svm. https://scikit-learn.org/stable/modules/classes.html#module-sklearn.svm. 2022.11.15.

[43] Sean J. Taylor and Benjamin Letham. 2017. Forecasting at Scale. *PeerJ Prepr.* 5 (2017), e3190. https://doi.org/10.7287/peerj.preprints.3190v1

[44] Sana Tonekaboni, Danny Eytan, and Anna Goldenberg. 2021. Unsupervised Representation Learning for Time Series with Temporal Neighborhood Coding. In *ICLR 2021*.

[45] Nancy Tran and Daniel A. Reed. 2004. Automatic ARIMA Time Series Modeling for Adaptive I/O Prefetching. *IEEE* (2004), 362–377.

[46] Chunnan Wang, Xingyu Chen, Chengyue Wu, and Hongzhi Wang. 2022. AutoTS: Automatic Time Series Forecasting Model Design Based on Two-Stage Pruning. *CoRR* abs/2203.14169 (2022).

[47] Xinlong Wang, Rufeng Zhang, Chunhua Shen, Tao Kong, and Lei Li. 2021. Dense Contrastive Learning for Self-Supervised Visual Pre-Training. In *CVPR 2021*. 3024–3033.

[48] Qingsong Wen, Kai He, Liang Sun, Yingying Zhang, Min Ke, and Huan Xu. 2021. RobustPeriod: Robust Time-Frequency Mining for Multiple Periodicity Detection. In *SIGMOD*. 2328–2337.

[49] Qingsong Wen, Tian Zhou, Chaoli Zhang, Weiqi Chen, Ziqing Ma, Junchi Yan, and Liang Sun. 2022. Transformers in Time Series: A Survey. *CoRR* abs/2202.07125 (2022).

[50] xgboost. 2019. xgboost. https://xgboost.ai/about. 2022.11.15.

[51] Qiantong Xu, Alexei Baevski, Tatiana Likhomanenko, Paden Tomasello, Alexis Conneau, Ronan Collobert, Gabriel Synnaeve, and Michael Auli. 2021. Self-Training and Pre-Training are Complementary for Speech Recognition. In *ICASSP 2021*. 3030–3034.

[52] Zhihan Yue, Yujing Wang, Juanyong Duan, Tianmeng Yang, Congrui Huang, Yunhai Tong, and Bixiong Xu. 2022. TS2Vec: Towards Universal Representation of Time Series. In *AAAI, 2022*. 8980–8987.

[53] George Zerveas, Srideepika Jayaraman, Dhaval Patel, Anuradha Bhamidipaty, and Carsten Eickhoff. 2021. A Transformer-based Framework for Multivariate Time Series Representation Learning. In *KDD '21*. 2114–2124.