Journal Européen des Systèmes Automatisés
Vol. 56, No. 6, December, 2023, pp. 981-991
Journal homepage: http://iieta.org/journals/jesa

Lightweight RNN-Based Model for Adaptive Time Series Forecasting with Concept Drift Detection in Smart Homes

Nitin B. Ghatage[1*], Pramod D. Patil[1], Sagar Shinde[2]

[1] Computer Engineering, Dr. D Y Patil Institute of Technology, Pune 411018, India
[2] Computer Engineering, NMVPM's Nutan College of Engineering and Research, Pune 410507, India

Corresponding Author Email: nitinbaghatage@gmail.com

https://doi.org/10.18280/jesa.560609

Received: 25 August 2023
Revised: 1 December 2023
Accepted: 15 December 2023
Available online: 28 December 2023

Keywords:
time series, lightweight, recurrent neural networks (RNN), concept drift detection, adaption to concept drift

ABSTRACT

Time-series forecasting is challenging in the real world. Both short-term and long-term forecasting are important in various fields of research and industry. Most forecasting algorithms perform great in providing one-step predictions, i.e., predicting only the next value in the time series data, but do not perform well while predicting multiple steps into the future. On top of that, concept drift makes it more challenging. The aim of this paper is to develop a lightweight recurrent neural networks (RNN)-based model that can do forecasting in the short or long term with the ability to detect concept drift and adapt to it automatically using a recent window of the data stream. The suggested model performs better than current techniques, with the lowest Root Mean Square Error (RMSE) of 0.0701, demonstrating increased accuracy in adaptive time series forecasting for temperature control in smart homes.

1. INTRODUCTION

Time series forecasting finds wide applications across the world, such as in the financial domain, weather forecasting, microfluidic chip production [1], high- frequency stock price forecasting [2], transport energy demand [3], load forecasting [4, 5], etc. For a long time, data scientists across the world have developed many algorithms that can predict the future values of a time series. However, most time series forecasting models suffer from problems like high volume and concept drift as shown in Figure 1. A high volume of time series data streams is a significant challenge as it requires scalable and efficient methods to handle the large amounts of data. Analysing and processing large volumes of data can be time-consuming and resource-intensive, which can affect the accuracy and efficiency of forecasting models. Concept drift, on the other hand, is one of the most critical challenges in time series forecasting as it can lead to inaccuracies in forecasting models that are trained on historical data that no longer accurately represent the current data distribution. Accurately detecting and adapting to concept drift is crucial for reliable forecasting.

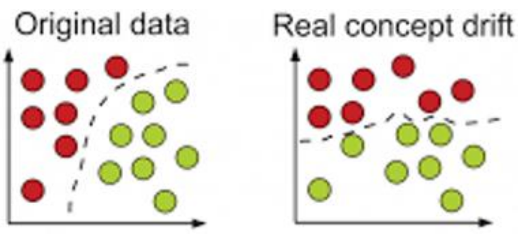

Figure 1. Concept drift

Retraining is a common technique used to tackle concept drift in time series forecasting. Retraining involves continuously updating the forecasting model based on the most recent data to adapt to changes in the underlying data distribution. By retraining the model on the most recent data, the model can adapt to changes in the underlying data distribution, including changes in the relationships between the variables in the time series. Retraining can help improve the accuracy and reliability of time-series forecasting models, particularly when concept drift is a significant factor in the data. However, retraining with all the historical data is infeasible due to the potential high volume of time series data. Time series forecasting is impacted by high volume and concept drift, which provide scalability and efficiency issues. Model accuracy is impacted by the resource-intensive nature of large-scale data analysis. Concept drift causes errors because models educated on old data do not match the distribution of new data, requiring adaptive methods like retraining.

In this paper, we discuss a novel end-to-end model that is lightweight in nature, can detect concept drift in the data stream, and can adapt to it automatically. The model is based on RNN. Unlike existing RNN-based models such as AIS-RNN [6], LSTM ensembles [7], or Ada-RNN [8], the proposed model focuses on dealing with concept drifts. In the proposed model, we used feature selection to feed only the important features of the data to the model to make it lightweight and faster. The proposed RNN model can retrain itself using only a defined window of recent data if concept drift is detected by itself. It also utilizes model versioning to reuse old models if suitable for the latest data stream. So, the proposed model can provide end to end forecasting with adaptability and lightweight nature. We also compare the proposed model against a recent research work, DSTP-RNN [9], and the results shown that the proposed model performed significantly better and has the potential to outperform state-of-the art algorithms.

2. RELATED WORK

In recent years, many models have been developed to address time-series forecasting. Some of the slightly older approaches include distributed ML techniques [10], which utilize a distributed mode of learning from the data stream. Recurrent broad learning systems [11] were developed to extract patterns from time series data. Some of the more advanced approaches are adaptive XGBoost [12], where new trees replace old trees with new data arrivals. AdaRNN [8] tries to learn the common pattern in the data stream across different time periods. Adaptive SVR [2] elevates traditional SVR to be able to process high-speed data streams, and CNN-LSTM [13] uses a grey wolf optimizer to control and update the hyperparameters of the model.

DSTP-RNN [9] is a dual-stage, two-phase attention-based model that finds the influence of features on target to perform well for long- and short-term multivariate forecasting. Noise augmentation to reduce drift is also proposed in literature [14]. However, the existing models require a significant amount of data processing and lack a proper, easy-to- interpret drift detection and adaptation approach. To tackle the high volume of time series data, some techniques have been applied, such as feature selection, where only important features get fed into the model; segmentation [15], where the data is segmented into multiple segments to be processed parallelly; and stratification [16], where a heterogeneous time series is split into multiple homogeneous time series to speed up the learning process. Techniques like OASW [17] and ADWIN [12] are popular approaches to detecting concept drift in the data stream based on the sliding window approach. In OASW, they measure the performance of the model over a recent sliding window of size t (that holds t data points) and compare it with previous model performance over the data points in the sliding window at t time in the past. If the difference is greater than a threshold, concept drift is detected. ADWIN divides the data into two windows, dynamically adjusts the window size to adapt to changes in data distribution and detects drift by looking for significant differences between the two windows. Some other techniques to handle concept drift include matching new data with old, segmented data [15] and continuous adaptation [18] without explicit detection of drift. Despite all these developments addressing certain issues with time series forecasting, there is a lack of end-to-end models that can handle most of these problems together and provide a complete solution. That's our motivation to develop such an end-to-end model that can handle any time series data with its complexities and provide stable, robust forecasting for a prolonged period without much human intervention. The suggested model performs well because it provides a comprehensive answer to problems with time series forecasting. In contrast to previous models, it integrates concept drift detection, feature selection, and adaptive learning to guarantee interpretability and scalability. It is an end-to-end solution for consistent and robust forecasting in a variety of time series settings, as demonstrated by comparative studies that demonstrate superior performance.

3. RESEARCH GAP

After reviewing existing literature in the field of multivariate time series forecasting, we found some areas where improvements are much needed:

•Generalized pre-processing of data before feeding it to models is heavily underutilized. Time series data can be huge in size as it is collected frequently over a long period of time. Building a model on this large amount of data takes a lot of time, and the biggest problem arises when we need to retrain the model later. It becomes simply infeasible to use all the past collected data with all the collected features as well. Thus, a well-defined pre-processing model to transform the data will be beneficial for building the forecasting model.

•Addressing concept drift is still a big challenge for time series forecasting, though several studies have been done on this. There is no universally proven best method, and most methods have limitations.

•A well-defined generalized methodology does not exist for retraining a forecasting model to adapt to changes in the data stream. The frequency and choice of different hyperparameters for retraining are two of the most important criteria when setting up a model retraining phase.

The proposed model thus tries to address the current research gaps by incorporating the following techniques:

•A pre-processing technique using feature selection to discard non-important features from the data to save storage and time consumption by the forecasting model, improve data quality, and make the model lightweight.

•An Online adaptive version of an RNN-based model to detect any kind of concept drift in the data stream and adapt by retraining itself on only a defined window of recent data such that the model performance does not degrade and gets steady. By using only the recent window of data for retraining the proposed model, the retraining is faster, and there is also no need to store all past data. The proposed model is also capable of identifying concepts that have occurred before and utilize model versioning to use past trained models without the need of retraining every time, thus saving a significant amount of time and efficiency.

The proposed time-series forecasting model is designed to provide businesses with the most accurate and up-to-date predictions possible. Unlike traditional models that require frequent total retraining to handle changes in the underlying data distribution, proposed lightweight model can detect concept drift and adapt to it in real-time automatically. The suggested model closes research gaps by presenting a feature selection pre-processing method for effective data use. It uses an online adaptive RNN to address concept drift, allowing for quick retraining on new data and real-time detection. The model provides an effective and flexible solution that addresses the shortcomings of generalised retraining approaches. It may be easily customised to meet changing business requirements. There will be a revolutionary effect if the gaps in multivariate time series forecasting are filled. The suggested adaptive RNN approach addresses the elusive idea of drift and simplifies retraining, while an improved pre-processing model improves effectiveness and data quality. The end product is a forecasting tool that is portable, precise, and flexible, giving organisations access to trustworthy real-time data for well-informed decision-making.

4. THE PROPOSED METHODOLOGY

As shown in Figure 2, the proposed methodology is an end-to-end process consisting of several stages. It starts with the collection of time series data and then proceeds to a lightweight pre-processing phase that includes the essential step of feature selection. The subsequent phase involves building the RNN model, with careful attention given to training and hyperparameter tuning using HyperOpt. After generating forecasts using unseen data, a concept drift detection mechanism is introduced, which involves monitoring performance through a sliding error window of recent data. Once concept drift is detected by the model, the model adaptation phase is initiated, and the model is retrained on a retraining window of recent data to adapt to new data patterns if the pattern was not already seen by past tracked models. In the following sections, each phase is discussed in greater detail.

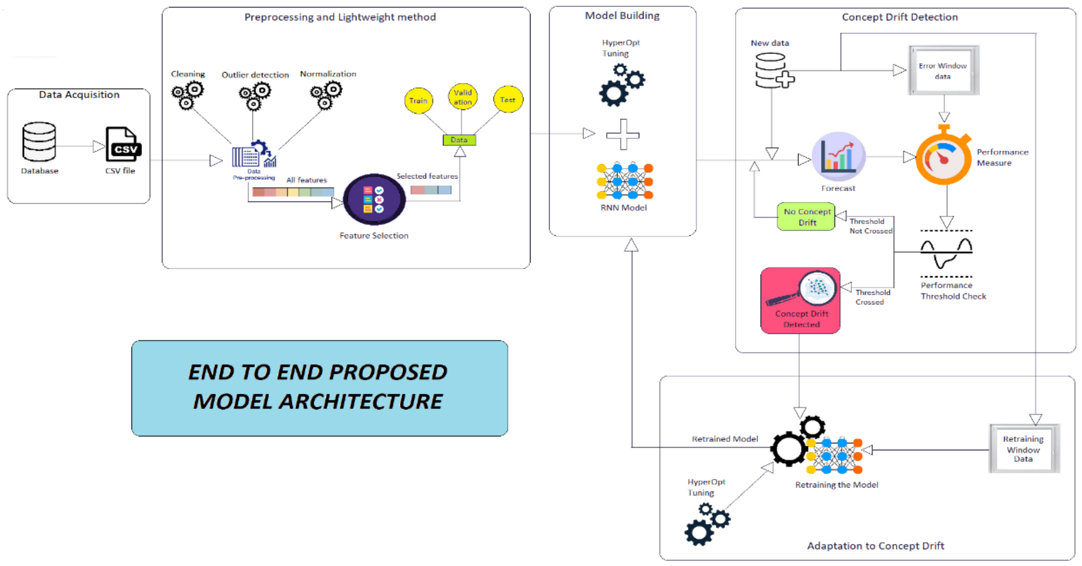

Figure 2. End to end model architecture

4.1 Lightweight method

Time series data typically has a high volume; hence, model training and retraining can take a lot of time. Thus, making a model lightweight is beneficial. Existing techniques to make a model lightweight include down-sampling, feature selection, segmentation, and stratification. All these techniques have their own advantages and drawbacks. However, out of them, feature selection is what we found to be most impactful on the quality of forecasting. Feature selection is a technique to choose only those features from the data that have a high influence on the target feature. In other words, it discards redundant and useless features from the data before it is fed to the model. This allows the model to learn from the selected quality data and thus potentially improve the forecasting quality.

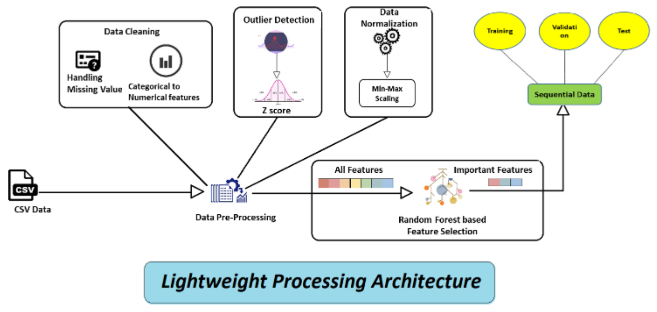

Figure 3. Lightweight model architecture

As shown in Figure 3, for the proposed model, we used a random forest-based feature selection technique to make the model lightweight.

4.1.1 Random forest based feature selection

Random forest-based feature selection is a popular technique used to select relevant features from data. The method works by constructing multiple decision trees based on a randomly selected subset of data and features. The algorithm assigns an importance score to each feature based on how well the feature separates the target variable. Features with higher importance are deemed more valuable. This process is carried out multiple times, and the average importance of each feature across the entire random forest is calculated and used to finally decide which features are going to be selected. By default, the algorithm selects those features with a higher feature importance than the average feature importance across all the features. The final selected features are fed to the forecasting model as input.

The computation of importance is measured by the metric Mean Squared Error (MSE), MSE is calculated as follows,

$$MSE = \frac{1}{N}\sum_{i=1}^{N}(y_i - \mu)^2 \qquad (1)$$

where, y is the value of target, N is the number of instances and u is the mean value of the target over all the data instances. For all the nodes of the trees in the Random Forest the MSE is calculated. Then the importance of each of the nodes is computed as follows.

$$Imp_j = W_j C_j - W_{lj} C_{lj} - W_{rj} C_{rj} \qquad (2)$$

where, Imp, stands for the importance of node j. W stands for weighted number of samples reaching node j. C is the MSE at node j, j and rj subscripts stand for left and right child after the split at node j respectively.

The importance of each individual feature is then calculated as follows.

$$FI_i = \frac{\sum_{j:node\ j\ splits\ on\ feature\ i} Imp_j}{\sum_{k\in all\ nodes} Imp_k} \tag{3}$$

where, FI_i stands for feature importance of ith feature and Imp_j is the importance of $node_j$. The feature importance is normalized in the range of 0 to 1. The feature importance is normalized in the range of 0 to 1 by dividing the feature importance values by the sum of all feature importance.

$$NormFI_i = \frac{FI_i}{\sum_{j\in all\ features} FI_j} \tag{4}$$

The average feature importance of each feature across all the trees in the Random forest is the final feature importance of the features,

$$RF\_FI_i = \frac{\sum_{j\in all\ trees} NormFI_j}{T} \tag{5}$$

where, RF_FI_i is the feature importance of feature i for the entire random forest, $NormFI_j$ is the normalized feature importance at the jth tree and T is the total number of decision trees in the forest.

By utilizing this feature selection process, the forecasting model gets only relevant and important features as input and not unwanted, noisy features, this helps improve the quality of the forecast and makes the model lightweight in nature.

4.2 Recurrent neural networks as base model

The base model algorithm is RNN. As shown in Figure 4, Recurrent neural networks are a kind of neural network that are capable of handling sequential data where the input data is a sequence of vectors.

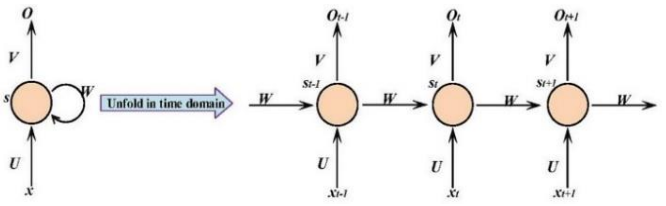

Figure 4. RNN algorithm flow

It can be text, speech, or time-series data. Unlike traditional neural networks that process each input piece of data independently, RNNs maintain an internal memory state that allows them to process sequences of arbitrary length. In simple words, RNNs can predict future values based on both current and past values, i.e., they have memory.

The RNN architecture consists of input X, hidden state h, and output Y. There are weights associated with each of them, namely, input weights Wx, recurrent or hidden layer weights Wh, and Output weights Wy, whereas bh and by are the biases corresponding to the hidden layer and output layer, respectively. At each time step 1, RNN takes the input vector Xt and the hidden state h(t-1) of the previous time step. The network then computes a new hidden state representation ht as a function of the current input and previous hidden state and generates an output Yt based on the new hidden state. Overall, RNNs and their more advanced versions like Long-Short-Term Memory (LSTM) and Gated Recurrent Unit (GRU) are powerful tools for time series forecasting and have been used successfully in a wide range of applications.

Mathematically, at each timestep, the computation of hidden state h and output Y is based on formulas as follows:

$$h_t=f(W_x{*}X_t+W_h{*}h_{t-1}+b_n) \tag{6}$$

$$Y_t=f(Wy{*}h_t+b_y) \tag{7}$$

where, f denotes the activation function, which is used to apply non-linear transformations to the data, which helps with training the neural networks. We used the TensorFlow Keras framework to build the RNN model in our study. RNN, like any neural network model, has a lot of hyperparameters to be set before training it. The complete information on the different hyperparameters of a RNN is provided in Table 1 as follows:

Table 1. Hyper-parameters of RNN

Hyper-Parameter	Description
Sequence length	The number of time steps that the RNN processes in each forward pass. Longer sequence lengths can capture more context but increase computational costs and memory requirements.
Number of hidden layers	Number of hidden layers of the RNN model.
Number of hidden units	Number of neurons in the hidden layer of the RNN. A larger number of hidden units can allow the network to learn more complex patterns, but with an increased risk of overfitting.
Activation	The function used to compute the output of each hidden unit. Activations can have a significant impact of the training procedure.
Batch size	Defines the size of the hatch of data to be processed before updating the weights during RNN training.
Epoch	Defines the number of passes through the entire dataset during training.
Learning rate	The step size used to update the weights of the RNN during training. A higher learning rate increases speed of convergence but also increases the risk of overshooting the optimal solution.
Optimizer	The algorithm used for updating the weights of the RNN while training.
Early stopping	To stop the training when the model's performance does not improve over epochs.

We train the model on a set of training data, and then to tune the hyperparameter, we use the Bayesian optimization algorithm. Bayesian optimization is a machine learning technique for tuning the hyperparameters of a given model. The idea behind it is to iteratively build a probability model of the objective function (RMSE of validation data). It then makes intelligent choices about where to sample the hyperparameter space next. Within predetermined parameters, the hyperparameter search space was defined. During optimisation, 50 trials were carried out in order to iteratively improve the performance of the model. By using the probability estimates of the model, Bayesian optimization can balance the exploration of new regions of the search space with the exploitation of promising regions that have already been sampled.

HyperOpt is a Python library that provides an implementation of Bayesian optimization for hyperparameter tuning. To use HyperOpt for hyperparameter tuning, the user defines a search space of possible hyperparameter values and an objective function (such as validation RMSE). HyperOpt then samples from the given search space, evaluates the objective function, and updates the probabilistic model. This process continues for a given number of trials.

Finally, after hyperparameter tuning, we get the final trained model with the best possible hyperparameters, ready to predict on the unseen data.

4.3 Concept drift detection

To address the issue of concept drift, As shown in Figure 5, the proposed model monitors its performance over a window of recent data. The window is called Error Window W and holds the latest w data points from the data stream. The model's performance (Accuracy for classification and RMSE or other regression metrics) is measured against the error window W. A defined error threshold is set beforehand based on the model's performance on the training data itself, and when the model's performance crosses the threshold, concept drift is detected, indicating a significant change in the data pattern. Once drift is detected, the model needs to be updated to adapt to the new data pattern and improve its performance. Because of its susceptibility to changes in prediction errors, RMSE was selected as the method for detecting concept drift. When used, it guarantees efficient model performance monitoring, identifying patterns in data deviations and triggering timely adjustments for increased predicting accuracy. So, assuming RMSE as the error metric, the concept drift detection algorithm is as follows:

```
Concept Drift Detection (ground_truth, pred, W, Th):
ground truth: Actual values
pred: predicted values
W: error window of size w
Th: Performance threshold
START
Running_error: RMSE over the latest window W, $\sqrt{\frac{1}{w}\sum_{i=1}^{w}(\text{pred}_i - \text{ground\_ruth}_i)^2}$; If (Running_error > Th):
Concept drift is detected;
Else:
Get Next model prediction;
Get next ground_truth from data stream;
Update the error window W;
Check for concept drift again;
END
```

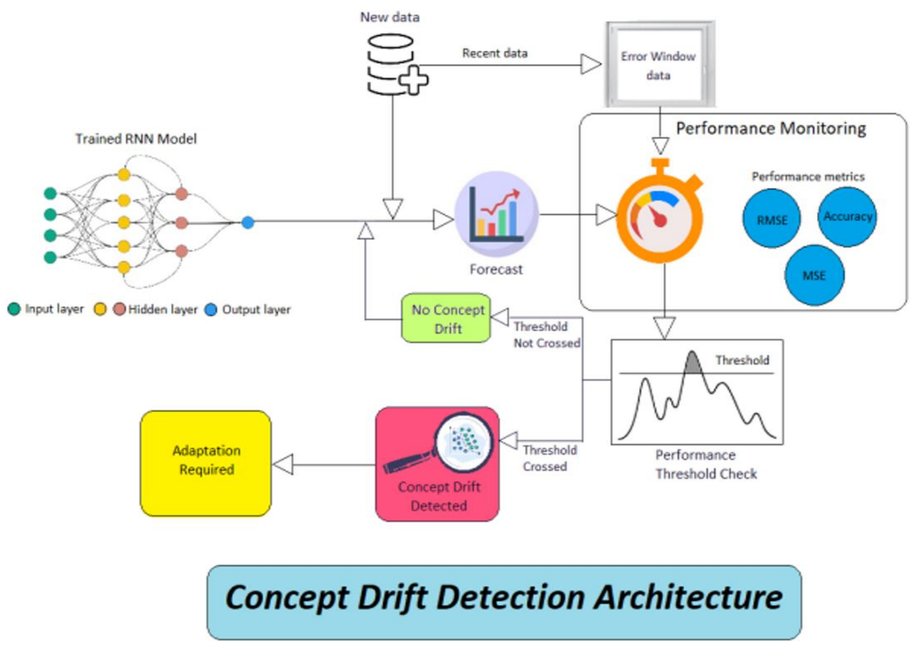

Figure 5. Concept drift detection method architecture

4.4 Adaptation to concept drift

Once concept drift is detected by the model, the next step is to make the model adapt to the new data pattern so that the degrading model performance can be improved as soon as possible. As shown in Figure 6, we propose retraining as the method to adapt new data patterns to the model. The complete proposed model adapts to concept drift after detecting it in the following ways:

•In the Online phase, the trained model forecasts the next values of the target feature as new data keeps coming from the data stream.

•We set a Retraining window R, that holds the most recent R data points from the data stream and is constantly updated with each new incoming data.

•As soon as Concept drift is detected, the model retraining phase starts.

•Retraining cannot be done on entire data as streaming data can be huge in size. We define a retraining technique that uses only the data in Retraining Window R to retrain itself and improve its performance. The retraining window size can be adjusted based on the frequency of drift and the available computational resources. This window-based retraining allows the model to adapt to the changing data distribution and ensures that it maintains high performance and reliability over time.

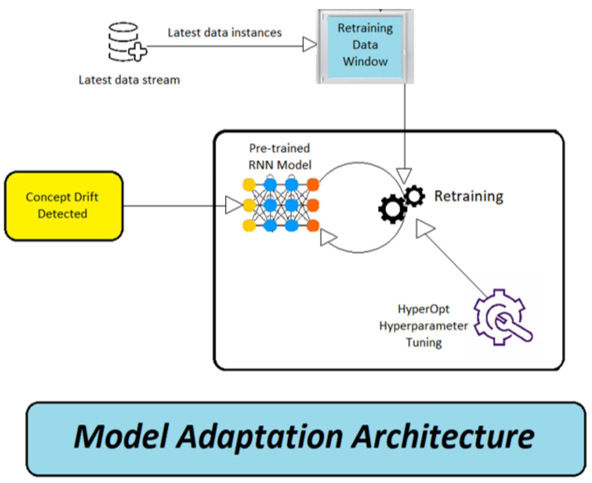

Figure 6. Adaptation to concept drift architecture

•While retraining the model, the hyperparameter learning rate is chosen to be smaller than the original learning rate used during the initial training phase. This is because the model has already learned some useful features from the previous trainings and needs to fine tune its parameters to learn new concepts in the data stream. If the learning rate used during retraining is too high, the model's parameters will be updated too quickly, leading to overfitting on the new data. This can result in poor generalization performance on the new, unseen data. This phenomenon is known as catastrophic forgetting. Hence, we set the learning rate and epoch to be smaller than what they were during initial training so that the model's parameters are updated more gradually, allowing it to adjust to the new data patterns without forgetting the previously learned information.

•During concept drift, it is not necessary that the old model hyperparameters are still the best hyperparameters. Hence, during retraining, we again utilize the HyperOpt hyperparameter tuning process. HyperOpt is a hyperparameter optimization tool chosen for its efficiency in fine-tuning model parameters. We split the retraining window into a 70-30% split and use the last 30% of the data as a validation set for the tuning process. We provide the search space for the necessary hyperparameters and set a trial number of 20 for speed and computation reasons. Once 20 trials are done, HyperOpt selects the best-performing set of hyperparameters, and finally the model is retrained on the entire retraining window using the best set of hyperparameters. However, it's important to note that the optimal set of hyperparameters may not be found within the defined number of trials, and it may be necessary to increase the number of trials for better results.So, the hyper-parameters of the online phase are provided in Table 2.

•As a retraining is completed, ideally the model performance should improve from before, but practically, if the data is noisy or if the concept drift is still going on, it may happen that even after one retraining session, the model performance is not better than the threshold. In that case, the model might go to another retraining session immediately, but the data for this retraining session will be almost the same as the last one, hence, it is redundant and not beneficial to do such retraining in quick succession. To address this, we make sure there is at least a difference of w data points between two retraining sessions.

Table 2. Hyper-parameters of retraining phase

Hyper-Parameter	**Significance**
Error Window size	Size of the window in which model performance is monitored to chock for concept drift
Retraining Window size	Size of the window that is used for retraining the model
Performance threshold	Critical value of the model's performance beyond which concept drift is detected
Retraining Learning rate	The rate of learning during retraining
Epoch	Number of times the retraining window data is process during a single retraining session
Batch Size	Retraining the batch size of the data based on which model weights are updated
Gap between two retraining sessions	A minimum amount of new data that has to be seen by the model between two consecutive concept drift detections

•Now as the retraining happen whenever the model detects concept drift, we save the versions of the model in memory. The benefit of this step is that whenever a concept drift is detected, if that newly detected concept has occurred before, it may happen that one of the previous versions of the model is capable of giving good predictions corresponding to this concept as that model has memorized that concept when it was trained. This will eliminate the unnecessary retraining process for this newly detected concept and thus save time. Also, it is way more efficient to save versions of models than to do a complete retraining.

•In our experiments we saved the latest 10 versions of the model and whenever a concept drift is detected we evaluate the performance of those last 10 models on the most recent error window. If any of those past models produce performance metric better than the set threshold, we simply replace the current model with the best performing past model and do not initiate retraining for this instance of concept drift detection

5. DATASET DESCRIPTION

For evaluating the proposed algorithm and comparing it against DSTP-RNN [9], we used one of the datasets used in the DSTP-RNN paper. The dataset name is SMI 2010. The data contains readings of temperatures from a monitor system in a domestic house. The sampling frequency of the data is per minute, but it is smoothed by using a 15-minute mean. The timespan of the original data is 40 days and divided into two folders, out of which we only use the first folder containing 30 days of data as the whole dataset. The room temperature is selected as the target the first 2000 data are used for training, and the rest of the 763 data are used as a holdout test set. The term "holdout test set" describes a subset of data that is only used to assess the performance of the model. In this instance, 763 data points from the latter half of the 40-day period were kept aside in order to evaluate the resilience and generalisation capacity of the model. This method guarantees an objective evaluation of the model on given data.

Data availability: The SML2010 data can be found in UCI Machine Learning Repository. Link: https://archive.ics.uci.edu/ml/datasets/SML2010. The details are provided in below Table 3.

Table 3. SML2010 data description

Hyper-Parameter	Significance
Date	Date: in UTC
Time	Time in UTC
Temperature Comedor Sensor	Indoor temperature (dining-room in A°C
Temperature Habitacion Sensor	Indoor temperature (room), in A°C
Weather Temperature	Weather forecast temperature, in Ã°C
CO_2 Comedor Sensor	Carbon dioxide in ppm (dining room)
CO_2 Habitacion Sensor	Carbon dioxide in ppm (cm)
Humedad Comedor Sensor	Relative humidity (dining room), in %
Humedad Habitacion Sensor	Relative humidity (roam), in %
Lighting Comedor Sensor	Lighting (dining room), in Lux
Lighting Habitacion Sensor	Lighting (room), in Lux
Precipitation	Rain, the proportion of the last
Meteo Exterior Crepusculo	Sun dusk
Meteo Exterior_Viento	Wind, in m/s
Merco Exterior Sol Oest	Sun light in west facade in Lux
Meteo Exterior Sol Est	Sun light in cast facade, in Lux
Metoo Exterior Sol Sud	Sun light in south facade; in Lux
Meteo Exterior Piranometro	Sun irradiance, in W/m^2
Exterior Entalpic	Enthalpic motor 1.0 or 1 (on-off)
Exterior Entalpic 2 Enthalpic Motor	2.0 or 1 (on-off)
Exterior Entalpic Turbo	Enthalpic motor turbo. 0 or 1 (on-off
Temperature Exterior Sensor	Outdoor temperature, in AC
Humedad Exterior Sensor)	Outdoor relative humidity, in %
Day of Wec	Day of the week (computed from the date), 1=Monday, 7=Sunday

6. RESULTS AND ANALYSIS

In this section, we present a detailed analysis of the performance metrics and provide insights into the strengths of the proposed model. We start by describing the experimental setup and the evaluation criteria used in this study Next, we showcase the time and memory consumption analysis of the proposed model, followed by discussion on concept drift detection and adaptation to the drift. We show how feature selection is important for achieving better results. We also compare the performance of the proposed model with a recent research work DSTP-RNN [9] on the SML2010 dataset to show that the proposed model outperforms it comprehensively.

6.1 Pre-processing and feature selection of data

Pre-processing is a necessary step to prepare the data in a way that can be fed to the model training phase. At first, we combined the data and time columns and set them as indexes for visualization purposes Since the SML 2010 dataset did not have missing values and all the features were numerical, additional steps to impute missing values or categorical feature encoding were not required.

We applied the random forest-based feature selection technique to the prepared data with a 70:30 split, 100 estimators, and a random seed of 0 for reproducibility, and the results were provided in Table 4.

Table 4. Results of random forest based feature selection

Dataset	Total Number of Features (Excluding Date and Time)	Number of Features Selected	Selected Features
SML2010	22	2	'Temperature_Habitacion_ Sensor' 'Temperature_Comedor_Sensor'

We checked for outlier in the dataset using Z score criteria (23) but the data does not contain any outliers.

6.2 Train-test split

To compare the results of the proposed model with those of the DSTP-RNN model, we set the train-test split the same as in that paper. The first 2000 instances are selected as the training set, and the last 763 data instances are selected as a holdout test set. During the training, 10% of the training set is used as a validation set for tuning the hyperparameters of the model. The RMSE is chosen as the evaluation metric for the experiments. For the purpose of evaluating the model, the choice to divide the data into training and test sets is essential. It makes evaluating the model's ability to generalise to new data possible. However, the robustness of the model may be limited by the use of just 763 data points for testing. Its performance in a variety of contexts would be better validated with a broader test set, reducing the possibility of overfitting to the particular holdout data.

6.3 Model specifications

A Recurrent Neural Network is used as the base model due to its effectiveness in time series forecasting and ability to find complex data patterns. The HyperOpt library is used for tuning the hyperparameters using the Bayesian Optimization technique. The final RNN model used for the SML 2010 dataset has one hidden layer with 12 units. The lag sequence parameter was chosen to be 12. An Adam optimizer with a learning rate of 0.001 is used for the training. The activation functions for the hidden and output layers are tanh and linear, respectively. A batch size of 256 is selected with 1500 epochs. L1 regularization of the value 0.0001 is used in the hidden layer to counter any possible overfitting. The model is trained to forecast the next 30 future values of the time series at every step.

6.4 Memory and time consumption analysis

To show the lightweight nature of the proposed model, we measured the time and memory consumption of the model in the training phase with and without the feature selection step. The results show that the feature selection step essentially helps the model run faster and requires less runtime memory. All the experiments were conducted on a machine with an 11th Gen Intel (R) Core (TM) i5-processor with a clock speed of 2.60 GHz, 16 GB of RAM, and 4 Core(s). The results are provided in Table 5 and Table 6.

Table 5. Memory consumption by the dataset with and without feature selection

Dataset Name	Dataset Size Before Feature Selection (KB)	Dataset Size After Feature Selection (KB)	Percentage Reduction of Size
SML2010	496	64.8	86.9

Table 6. Time consumption for training by the dataset with and without feature selection

Dataset Name	Time Without Feature Selection	Time with Feature Selection	Percentage Reduction of Time
SML2010	1 min 20 sec	1 min 07 sec	16.2

From the above results, we can see that the same RNN model takes more time and memory when all the features of the dataset are used compared to when only the important features selected by the Random Forest based feature selection algorithm are used. We see a huge 86.7% reduction in memory consumption, which can save a lot of memory as the time series data tends to have a high volume. Also, the reduction in time consumption is 16.2%, which, by intuition, we can say will increase significantly when the volume of the data increases.

6.5 Model performance with and without feature selection analysis

To show that feature selection helps the model learn from important relevant features and not get confused with redundant noisy features, we compared the results of the proposed model with and without feature selection with identical settings of hyperparameters. The result is provided in Table 7.

Table 7. Model performance on test dataset with and without Feature Selection

Dataset Name	RMSE Without Feature Selection	RMSE with Feature Selection	Percentage Improvement of RMSE
SML2010	0.433	0.0701	83.8

Hence, we get about an 83.8% improvement in RMSE. The reason behind that could be that the features that were dropped using the Random forest-based feature selection algorithm were noisy and did not contribute much new information to modelling the target feature. Hence, dropping those features helps the model perform better.

6.6 Concept drift detection and adaptation analysis

As discussed, the proposed model detects concept drift by monitoring the model's performance over the defined error window. Whenever the model performance RMSE goes above a set threshold, concept drift is detected. The threshold is set based on the model performance on the training set itself, i.e., the training RMSE as provided in Table 8.

Table 8. Training RMSE on the dataset

Dataset Name	Training RMSE
SML2010	0.1397

To have some flexibility in the drift detection, we set the RMSE threshold at 120% of the training RMSE as provided in Table 9.

Table 9. Performance threshold for the dataset

Dataset Name	Performance Threshold
SML2010	0.1696

The error window size is chosen as follows based on intuition and trials as shown in Table 10.

Table 10. Error window size for the dataset

Dataset Name	Error Window Size
SML2010	120

Because the SML 2010 dataset does not have any concept drift, we injected artificial drift into the test data to see how the proposed model detects it. We injected incremental drift of a magnitude of half the mean of the target from the 200th instance onwards. The test data before and after drift injection is as follows:

As the model detects concept drift, the immediate next action should be to retrain the model so that it can adapt to the new concepts in the data stream. We define a retraining window R that holds the most recent R data points from the data stream. The retraining uses only the data within window R. During the retraining to tune some of the hyper-parameters, we employed HyperOpt again. We split the retraining window 70:30 and use the last 30% as validation data for the tuning. Once tuning is done and the best-performing hyperparameters are found, we finally retrain the model for the entire retraining window using the best set of hyperparameters. To avoid too frequent retraining, we specify a minimum gap between two retraining equal to the size of the error window. Also after every drift detection we check if any of the past 10 saved models produce performance metric better than the threshold set. If yes, the best performing past model replaces the current model and there is no need for retraining for that particular concept drift detection instance. The retraining window size is selected as follows and is provided in Table 11.

The selection of 256 batches and 1500 epochs was determined by computational performance as well as empirical findings. Through experimentation, these parameters were found, striking a balance between computational resources and model convergence. The model's performance and resource usage on the specified hardware are optimized while effective training is ensured by the chosen settings. Next, to see how retraining helps with concept drift adaptation. We measured the model performance in terms of RMSE, with and without retraining on the drift injected test data of the SML 2010 dataset and the comparison is provided in Table 12.

Table 11. Retraining window size for the datasets

Dataset Name	Retraining Window Size
SML2010	120

Table 12. Performance measure RMSE of the proposed model before and after drift addition, with and without drift detection and retraining

Dataset	Model Performance on Test Set Before Drift was Added	Model Performance on Test Set Without Drift Detection and Retraining	Model Performance on Test Set After Drift Detection and Retraining
SML2010	0.0701	2.88	0.378

As evident from the table above, retraining helps the model to quickly adapt to the concept drift in the data and thus making the model performance significantly better. Now it is to note that at the start of concept drift in the data, the model performance cannot be good due to fluctuations and thus the average RMSE is slightly higher than the RMSE without the concept drift added. This is clear if we look at the graphs of prediction vs actual values in both with and without retraining case.

As shown in Figures 7 and 8, retraining helps the model adapt to the new data patterns quickly. We also show the actual values that we get as predictions from the model both with and without retraining on the test dataset.

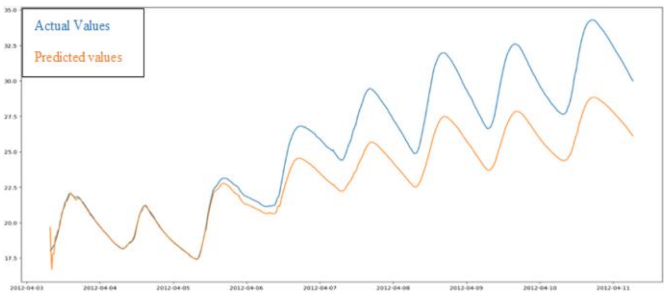

Figure 7. Prediction vs actual values on drift injected test set of SML2010 data without concept drift detection and adaptation

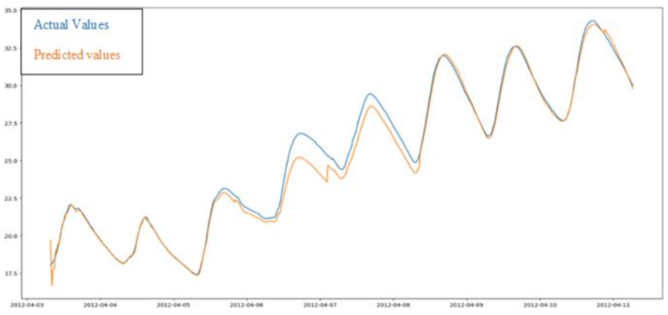

Figure 8. Predictions actual values on drift injected test set of smi 2010 data after concept drift detection and adaptation

As shown in Tables 13 and 14, we compare the predicted values with the actual ground truth values and see the difference in the quality of the prediction. Just so compare with the ground truth, here the predicted values are the first values of each prediction since the proposed model is trained to predict the next 30 values together.

Next, we compare the RMSE curves of both the cases of model prediction, Ie, while retraining is done and while it's not done. The RMSE curve is formed by using the RMSE values on the error window ay the model predicts continuously on the test data.

As shown in Figure 9, it is clear from the RMSE curves that with retraining, the RMSE of prediction stays low, which is ideal and means that the model is adapting according to the new data patterns. This proves our intuition and argument that retraining is indeed helping the model give better predictions even in the event of concept drift in the data stream.

Table 13. Sample of actual vs predicted values on the drift injected sml2010 test set when concept drift detection and retraining is not done

Date	Prediction	Original
4/11/2012 1:45	27.4	31.8
4/11/2012 2:00	27.3	31.7
4/11/2012 2:15	27.3	31.6
4/11/2012 2:30	27.2	31.5
4/11/2012 2:45	27.1	31.5
4/11/2012 3:00	27.1	31.3
4/11/2012 3:15	27.0	31.3
4/11/2012 3:30	27.0	31.2
4/11/2012 3:45	26.9	31.1
4/11/2012 4:00	26.8	31.0
4/11/2012 4:15	26.8	30.9
4/11/2012 4:30	26.7	30.8
4/11/2012 4:45	26.6	30.7
4/11/2012 5:00	26.5	30.6
4/11/2012 5:15	26.5	30.5
4/11/2012 5:30	26.4	30.4
4/11/2012 5:45	26.3	30.3
4/11/2012 6:00	26.3	30.1
4/11/2012 6:15	26.2	30.1
4/11/2012 6:30	26.1	30.0

Table 14. Sample of actual vs predicted values on the drift injected sml2010 test set with concept drift detection and retraining

Date	Prediction	Original
4/11/2012 1:45	32	31.8
4/11/2012 2:00	31.9	31.7
4/11/2012 2:15	31.8	31.6
4/11/2012 2:30	31.7	31.5
4/11/2012 2:45	31.6	31.5
4/11/2012 3:00	31.5	31.3
4/11/2012 3:15	31.4	31.3
4/11/2012 3:30	31.3	31.2
4/11/2012 3:45	31.1	31.1
4/11/2012 4:00	31.0	31.0
4/11/2012 4:15	30.9	30.9
4/11/2012 4:30	30.8	30.8
4/11/2012 4:45	30.7	30.7
4/11/2012 5:00	30.5	30.6
4/11/2012 5:15	30.4	30.5
4/11/2012 5:30	30.3	30.4
4/11/2012 5:45	30.2	30.3
4/11/2012 6:00	30.0	30.1
4/11/2012 6:15	29.9	30.1
4/11/2012 6:30	29.8	30.0

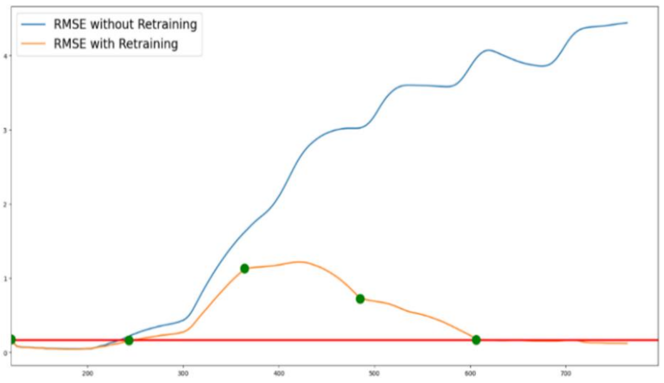

Figure 9. RMSE curves for SML2010 dataset with and without training. Red line indicates the RMSE threshold, and the green blobs indicate instances where drift was detected, and retraining happened

7. COMPARISON

Now, benchmarking is very important in the field of machine learning. To validate the performance of the proposed model, we compared it against a recent research paper, DSTP-RNN (9) DSTP RNN and DSTP-RNN. tl are dual stage attention-based RNN models that find the influence of non-target series on target series and use that information to forecast. We compare the RMSE of the proposed model on the SML 2010 dataset with any drift injection against both versions of DSTP-RNN and other benchmark forecasting algorithms as provided in 19) All the algorithms were trained to predict the next 30 predictions at a time. The result is provided in Table 15.

As we can see, the proposed model outperforms the previous best RMSE of DSTP-RNN by 28.9% and is clearly the best performing out of all the other algorithms involved. The suggested model performs better than the others, as evidenced by its lower RMSE, which shows that it is better able to capture complex time dependencies in the data. Its robustness is enhanced by its lightweight design and effective concept drift detection and adaptation, which distinguish it as the best-performing model among the compared methods.

Table 15. Comparison of benchmark algorithms and DSTP-RNN and the proposed model on the SML 2010 dataset with 30 step ahead prediction criteria

Method	RMSE
Arima	1.0631
SVR	0.6843
LSTM	0.7016
GRU	0.7084
Encoder-Decoder	0.2537
Input Att RNN	0.2144
Temp Att RNN	0.2406
DARNN	0.2080
GeoMAN	0.1310
DeepAttn	0.1647
DSTP-RNN	0.0987
DSTP-RNN-II	0.0993
The Proposed Model	0.0701

8. CONCLUSION AND FUTURE WORK

In conclusion, this paper presented a lightweight time series forecasting model that incorporates concept drift detection and adaptation. The proposed model employs a combination of machine learning algorithms, which makes it easy to implement and computationally efficient. The concept drift detection mechanism based on a sliding window architecture allows the model to adapt to changing patterns in the data, ensuring accurate forecasts over prolonged periods of time. Retraining based on recent windows makes it efficient in terms of storage requirements to keep old data.

Experimental results on the SML2010 dataset demonstrated the effectiveness of the proposed approach in detecting and adapting to concept drifts, leading to a significant improvement in forecasting quality over traditional approaches. Furthermore, the model's lightweight nature makes it suitable for deployment in resource- constrained environments. The proposed approach is suitable for time series forecasting in contexts with limited resources, which makes it useful in sectors like logistics, energy, and the Internet of things. The comparison against other established models clearly showcased the potential of the proposed model for time series forecasting. The future research scope includes utilizing more sophisticated techniques to improve the lightweight nature of the model and making new advancements in the concept drift detection approach to identify the type of drift and distinguish between noise and actual new data patterns. As the base model, the latest neural network-based algorithms can be tried out to see how they perform with more complex real-world time series data streams.

REFERENCES

[1] Lughofer, E., Pollak, R., Zavoianu, A.C., Pratama, M., Meyer-Heye, P., Zörrer, H., Eitzinger, C., Haim, J., Radauer, T. (2018). Self-adaptive evolving forecast models with incremental PLS space updating for on-line prediction of micro-fluidic chip quality. Engineering Applications of Artificial Intelligence, 68: 131-151. https://doi.org/10.1016/j.engappai.2017.11.001

[2] Guo, Y., Han, S., Shen, C., Li, Y., Yin, X., Bai, Y. (2018). An adaptive SVR for high-frequency stock price forecasting. IEEE Access, 6: 11397-11404. https://doi.org/10.1109/ACCESS.2018.2806180

[3] Sahraei, M.A., Duman, H., Çodur, M.Y., Eyduran, E. (2021). Prediction of transportation energy demand: Multivariate adaptive regression splines. Energy, 224: 120090. https://doi.org/10.1016/j.energy.2021.120090

[4] Fekri, M.N., Patel, H., Grolinger, K., Sharma, V. (2021). Deep learning for load forecasting with smart meter data: Online Adaptive Recurrent Neural Network. Applied Energy, 282: 116177. https://doi.org/10.1016/j.apenergy.2020.116177

[5] Qiu, X., Suganthan, P.N., Amaratunga, G.A. (2018). Ensemble incremental learning random vector functional link network for short-term electric load forecasting. Knowledge-Based Systems, 145: 182-196. https://doi.org/10.1016/j.knosys.2018.01.015

[6] Munkhdalai, L., Munkhdalai, T., Park, K.H., Amarbayasgalan, T., Batbaatar, E., Park, H. W., Ryu, K.H. (2019). An end-to-end adaptive input selection with dynamic weights for forecasting multivariate time series. IEEE Access, 7: 99099-99114. https://doi.org/10.1109/ACCESS.2019.2930069

[7] Choi, J.Y., Lee, B. (2018). Combining LSTM network ensemble via adaptive weighting for improved time series forecasting. Mathematical Problems in Engineering, 2018: 2470171. https://doi.org/10.1155/2018/2470171

[8] Du, Y., Wang, J., Feng, W., Pan, S., Qin, T., Xu, R., Wang, C. (2021). Adarnn: Adaptive learning and forecasting of time series. In Proceedings of the 30th ACM International Conference on Information & Knowledge Management, Virtual Event, Queensland, Australia, pp. 402-411. https://doi.org/10.1145/3459637.3482315

[9] Liu, Y., Gong, C., Yang, L., Chen, Y. (2020). DSTP-RNN: A dual-stage two-phase attention-based recurrent neural network for long-term and multivariate time series prediction. Expert Systems with Applications, 143: 113082. https://doi.org/10.1016/j.eswa.2019.113082

[10] Mohapatra, U.M., Majhi, B., Satapathy, S.C. (2019). Financial time series prediction using distributed machine learning techniques. Neural Computing and Applications, 31: 3369-3384. https://doi.org/10.1007/s00521-017-3283-2

[11] Xu, M., Han, M., Chen, C.P., Qiu, T. (2018). Recurrent broad learning systems for time series prediction. IEEE Transactions on Cybernetics, 50(4): 1405-1417. https://doi.org/10.1109/TCYB.2018.2863020

[12] Montiel, J., Mitchell, R., Frank, E., Pfahringer, B., Abdessalem, T., Bifet, A. (2020). Adaptive xgboost for evolving data streams. In 2020 International Joint Conference on Neural Networks (IJCNN), Glasgow, UK, pp. 1-8. https://doi.org/10.1109/IJCNN48605.2020.9207555

[13] Xie, H., Zhang, L., Lim, C. P. (2020). Evolving CNN-LSTM models for time series prediction using enhanced grey wolf optimizer. IEEE Access, 8: 161519-161541. https://doi.org/10.1109/ACCESS.2020.3021527

[14] Fields, T., Hsieh, G., Chenou, J. (2019). Mitigating drift in time series data with noise augmentation. In 2019 International Conference on Computational Science and Computational Intelligence (CSCI), Las Vegas, NV, USA, pp. 227-230. https://doi.org/10.1109/CSCI49370.2019.00046

[15] Song, Y., Lu, J., Liu, A., Lu, H., Zhang, G. (2021). A segment-based drift adaptation method for data streams. IEEE Transactions on Neural Networks and Learning Systems, 33(9): 4876-4889. https://doi.org/10.1109/TNNLS.2021.3062062

[16] Lu, Y., Park, Y., Chen, L., Wang, Y., De Sa, C., Foster, D. (2021). Variance reduced training with stratified sampling for forecasting models. In Proceedings of the 38th International Conference on Machine Learning, Online, pp. 7145-7155.

[17] Yang, L., Shami, A. (2021). A lightweight concept drift detection and adaptation framework for IoT data streams. IEEE Internet of Things Magazine, 4(2): 96-101. https://doi.org/10.1109/IOTM.0001.2100012

[18] Read, J. (2018). Concept-drifting data streams are time series; the case for continuous adaptation. arXiv preprint arXiv:1810.02266. https://doi.org/10.48550/arXiv.1810.02266