sustainability

MDPI

Article

Loss-Driven Adversarial Ensemble Deep Learning for On-Line Time Series Analysis

Hyungjin Ko [1], Jaewook Lee [1], Junyoung Byun [1], Bumho Son [1] and Saerom Park [1,2,*]

[1] Industrial Engineering, Seoul National University, Seoul 08826, Korea; hyungjinko@snu.ac.kr (H.K.); jaewook@snu.ac.kr (J.L.); quswns95@snu.ac.kr (J.B.); andymogul@snu.ac.kr (B.S.)

[2] Industrial and Mathematical Data Analytics Research Center, Seoul National University, Seoul 08826, Korea

* Correspondence: drsaerompark@gmail.com

Received: 28 May 2019; Accepted: 20 June 2019; Published: 25 June 2019

Abstract: Developing a robust and sustainable system is an important problem in which deep learning models are used in real-world applications. Ensemble methods combine diverse models to improve performance and achieve robustness. The analysis of time series data requires dealing with continuously incoming instances; however, most ensemble models suffer when adapting to a change in data distribution. Therefore, we propose an on-line ensemble deep learning algorithm that aggregates deep learning models and adjusts the ensemble weight based on loss value in this study. We theoretically demonstrate that the ensemble weight converges to the limiting distribution, and, thus, minimizes the average total loss from a new regret measure based on adversarial assumption. We also present an overall framework that can be applied to analyze time series. In the experiments, we focused on the on-line phase, in which the ensemble models predict the binary class for the simulated data and the financial and non-financial real data. The proposed method outperformed other ensemble approaches. Moreover, our method was not only robust to the intentional attacks but also sustainable in data distribution changes. In the future, our algorithm can be extended to regression and multiclass classification problems.

Keywords: ensemble deep learning; on-line learning; time series analysis; adaptive learning

1. Introduction

Ensemble methods have been developed to achieve robust and high performance in various tasks, such as image classification, on-line learning, financial data prediction, and clustering [1–7]. Such methods aim to construct a group of models and aggregate the results of the models, where a high diversity of models is preferred. Two important issues must be addressed in ensemble learning: selecting candidate models and aggregating the results of the models [3,4]. Although the selection of candidate models can have a greater impact than the aggregation strategy, the selection may require a difficult decision and depend on prior knowledge. Applying the aggregation strategy can have a similar effect as selecting models. These two components of ensemble learning are appropriate for applying to on-line learning scenarios because they help to adapt the entire model to changing input data.

On-line ensemble learning has become popular because ensemble learning cannot only increase the robustness of models for atypical events but also the predictive performance. In general, we can postulate that no single dominant model can be used for all unknown samples [8]. Various properties, such as seasonality, concept drift, and trend, which can change dominant models, should be considered in analyzing time series data. On-line ensemble learning has solved this problem by changing candidate models or adjusting weights on the basis of competence levels for new samples [2,3,9]. However, the cost of training a new model or retraining the existing models is too high to be applied to streaming

data, particularly in deep learning models. For example, financial data prediction requires data that are constantly utilized during trading sessions; thus, the only means to improve the prediction model is on-line learning, and the models can be retrained only if given sufficient time, such as nontrading days [7,10]. Benkeser et al. proposed an on-line cross-validation-based ensemble learning method to avoid retraining models with new samples; however, they only used simple base learners, such as bounded logistic regression models. Therefore, we developed a new on-line learning method to reduce the training and retraining costs of deep learning models. This method can act as a fundamental building block for a robust and sustainable system for time series data.

In ensemble deep learning, candidate models for an ensemble model are made of deep learning models. Training a deep learning model requires solving a high-dimensional nonconvex optimization problem, which can have multiple local minima [11,12]. Ensemble deep learning methods that use model averaging with multiple neural networks have won first places in various tasks, such as image classification, localization, and detection, in ImageNet Large-scale Visual Recognition Challenge 2012, 2014, and 2015 [13–15]. Such methods use a simple averaging strategy to combine multiple models and show more interests in designing the sophisticated network structures. However, this simple averaging is vulnerable to a small number of poor candidate models and non-adaptive to data. Thus, an ensemble deep learning method for aggregation is required to achieve refined weights [1]. Ju et al. developed a stacking-based ensemble deep learning method to compute the ensemble weights of neural network models; the weights were obtained by solving a constrained convex optimization problem or training the weights with a validation dataset [1]. Although training the weights exhibited good performances, the performance of this method can highly depend on the selection of validation data. An ensemble of deep learning models can significantly improve the performance of time series classification [16]. However, this ensemble model implicitly postulates off-line learning, and cannot provide the adaptability to the change of time series data. Fan et al. proposed on-line deep ensemble learning method for predicting citywide human mobility, which constructs the adaptive human mobility predictor and combines the pre-trained predictors instead of deep learning models [17]. This on-line deep ensemble learning method can be applied only to GRU-based deep learning model. Therefore, we aim to develop an efficient on-line ensemble deep learning method that adjusts the ensemble weights by using continuously incoming data and is applicable to any deep learning models minimizing loss function in the current study.

The fundamental objectives of this study are to propose an on-line learning procedure for deep learning models and to develop an adversarial on-line ensemble learning method by using the loss function value of the previous stage. In our scenario, off-line learning (i.e., training deep learning models) is computationally intensive, but the adaptability of the algorithm is necessary to dynamically adapt to changes in continuously incoming data. Therefore, we augment adaptability by introducing an aggregation strategy of ensemble deep learning for classification. We propose a new regret measure for our ensemble model and demonstrate that our algorithm can minimize regret based on adversarial assumption. We verified our on-line ensemble learning algorithm through extensive experiments with financial and non-financial time series data. We also conducted experiments to demonstrate the effectiveness of our algorithm from two aspects. The first aspect showed the robustness of the ensemble model when several classifiers deteriorated. The second aspect was about adaptability to time series data when the data distribution changed.

The remainder of this paper is organized as follows. In Section 2, we review ensemble deep learning and on-line learning research. In Section 3, we propose an on-line ensemble deep learning that updates ensemble weight on the basis of the loss value of streaming data, theoretically demonstrate the convergence of our algorithm, and provide an overall framework to apply the proposed algorithm to real-world problems that involve analyzing time series data. In Section 4, we present the verification of the effectiveness of our algorithm through various experiments using simulated data, financial time series, and non-financial time series. In Section 5, we discuss the robustness and the sustainability

of our algorithm to intentional attacks and the changes in the data distribution. Section 6 concludes this study.

2. Related Work

2.1. Ensemble Deep Learning

Deep learning models have been successfully used in various supervised tasks, such as computer vision, speech recognition, and natural language processing [13,15,18,19]. A representative example of deep learning models is the feed-forward deep neural network that builds a complex function by composing simple functions. Convolutional neural networks (CNN) are specialized neural networks for data with grid-like topology, such as time series and image data [13,20]. Recurrent neural networks (RNN) are designed for sequential data [21–24]. With the advent of the big data era, these deep learning models have achieved state-of-the art performance in many real-world applications. However, the models are based on the assumption that data distribution remains unchanged, and, consequently, changes in data distribution can deteriorate the performance of the models. Therefore, we need a more sustainable model.

Ensemble methods can be one of the most promising research directions to improve the sustainability of models. These methods have been used mostly to improve the performance in deep learning applications [1,13–15]. Candidate models of ensemble deep learning share the same network structure, but with different model checkpoints or with different initial parameters [1]. Although the ensemble of deep learning models with cross-validated unequal weights can improve robustness for noisy data, the deterioration caused by a change in data distribution in time series data is inevitable because ensemble weights are fixed [1,25,26]. In the case of time series data, detecting and dealing with a change in data distribution are important [4]. Therefore, we aim to propose an ensemble deep learning method for on-line time series analysis that is adaptable and sustainable in real-world applications.

2.2. On-Line Learning

The primary objective of on-line learning is to minimize the regret, which is the difference between the performance of the on-line learning algorithm in a streaming data setting and the off-line algorithm using all the data. In general, an on-line learning setting assumes that the algorithm receives an instance x_t and makes a prediction $\hat{y}_t \in \{0, 1\}$. Let an on-line setting have T rounds and M experts, where the prediction of the expert m at tth round is $\hat{y}_{t,m}$, the true label at tth round is y_t, and the loss at tth round is $\ell(\hat{y}_t, y_t)$. Regret is defined as:

$$\mathcal{R}_T = \sum_{t=1}^{T} \ell(\hat{y}_t, y_t) - \min_{m=1,\ldots,M} \sum_{t=1}^{T} \ell(\hat{y}_{t,m}, y_t). \quad (1)$$

The objective of on-line learning is to minimize regret $\mathcal{R}_T$. However, this regret postulates that the on-line algorithm should find the best expert among candidate learners. However, in the case of time series data, the best model can change due to various reasons, such as covariate shift, varying distribution, and seasonality. Thus, base learners are constructed by deep learning models based on diverse variables. Our algorithm keeps all experts active instead of finding the best expert. In this study, we propose a new regret measure that is compatible with our algorithm and demonstrate the convergence of our algorithm in Section 3.

The simplest algorithm for on-line learning is the Halving algorithm, which makes a prediction from the majority vote over all active experts. Incorrect models are filtered every round, and the remaining models are kept as the set of consistent experts $C_t = \{m : \hat{y}_{s,m} = y_s, \forall s = 1, \ldots, t-1\}$ for the next round t. However, the Halving algorithm strongly assumes that the best expert is perfect.

The exponential weighted average algorithm relieves this assumption by using weights based on past performance.

The exponential weighted average algorithm introduces the weights $\mathbf{w}_t = (w_1^t, \ldots, w_M^t) \in \mathbb{R}^N$. The prediction is given as $\hat{y}_t = \mathbf{1}_{f_t > 0.5}$ where

$$f_t = \frac{\sum_{m=1}^{M} w_m^t f_m^t}{\sum_{m=1}^{M} w_m^t}.$$

We denote the cumulative loss of model i up to time t as $L_m^t = \sum_{s=1}^{t} \ell(f_m^s, y_s)$ and the cumulative loss of the algorithm up to time t as $L_t = \sum_{s=1}^{t} \ell(f_s, y_s)$. The weights w_t is updated as $\tilde{w}_m^{t+1} \leftarrow w_m^t \exp(-\eta L_m^t)$, and $w_m^t = \tilde{w}_m^t / \sum_k \tilde{w}_k^t$. Although the Halving algorithm uses 0–1 loss, the exponential weighted average algorithm uses convex loss function $\ell : [0,1] \times \{0,1\} \to [0,1]$, such as squared loss and absolute loss. In the regret in Equation (1), $\hat{y}_t$ is replaced with f_t, as shown in the following equation:

$$\mathcal{R}_T = \sum_{t=1}^{T} \ell(f_t, y_t) - \min_{m=1,\ldots,M} \sum_{t=1}^{T} \ell(f_m^t, y_t). \tag{2}$$

In accordance with Theorem 7.6 in [27], the regret of this algorithm after T round can be bounded by $\sqrt{(T/2)\log M}$ for $\eta = \sqrt{8 \log M / T}$. However, this analysis is based on the external regret that compares the cumulative loss of the algorithm and the cumulative loss of the best expert among M models. Therefore, we propose and analyze a new regret based on the comparison between the cumulative loss of the algorithm and the cumulative loss of the ensemble model with the optimal weight in Section 3.

3. Proposed Method

In this study, we propose an on-line ensemble deep learning method for time series data. This method is adaptive to atypical events and robust to the attacks for several candidate models. Our method consists of off-line and on-line learning phases. In the off-line phase, we train candidate deep learning models with the accumulated time series data. In the on-line phase, the ensemble weights are updated depending on the performance of deep learning models with incoming data. In Section 3.1, we introduce the on-line ensemble deep learning algorithm and prove that the sequence of the ensemble weights of our algorithm can converge to the optimal solution that minimizes total expected loss. Section 3.2 describes the overall framework for analyzing time series data using the on-line ensemble deep learning method.

3.1. Loss-Driven Adversarial Ensemble Deep Learning

Training deep learning models requires determining the cost function and using the iterative gradient-based optimization algorithm to solve nonconvex optimization problems, even if the cost function is convex in the first argument (the predicted value). The on-line ensemble deep learning algorithm calculates the cost function for incoming data, and updates only the ensemble weights on the basis of the calculation. For a classification task, the cross entropy loss between the prediction and the training label is typically used. The binary cross entropy loss function $\ell_{CE} : [0,1] \times \{0,1\} \to \mathbb{R}^+$ is given as:

$$\ell_{CE}(\tilde{y}, y) = -y \log \tilde{y} - (1-y) \log (1-\tilde{y}). \tag{3}$$

In Section 2.2, on-line learning algorithms postulate that the best expert exists among candidates, and each round receives an instance and makes a prediction. In on-line ensemble deep learning, however, each round can receive a mini batch of instances and update ensemble weights based on the batch. In such cases, the cost function is calculated from the mini batch similar to training the deep learning models. We can also use the error rate of the batch as the loss function to update the ensemble

weights. This mini batch strategy can be appropriate for coping with the effect of abrupt jumps in time series data.

During the on-line phase, our algorithm combines the base classifiers $f_m(x), m \in \{1, ..., M\}$ with weight vector $\mathbf{w}_t = (w_1^{(t)}, \ldots, w_M^{(t)})$,. The prediction $f^{(t)}(x)$ at t is obtained by stacking the predictions of the base classifiers as $f^{(t)}(x) = \sum_{m=1}^{M} w_m^{(t)} f_m(x)$, as in [8,28]. The proposed method aims to obtain the on-line ensemble strategy, which can converge to an optimal weight that minimizes the average total loss for a series of instances while not training the candidate models.

After training candidate models, we set the initial weight $w_m^{(0)} = \frac{1}{M}$. We denote the loss matrix $\mathbf{A} \in [0,1]^{M \times N}$, which is defined by $A_{mi} = \ell(y_i, f_m(x_i))$, over the base classifiers $m = 1, \ldots, M$ and the data instance $i = 1, \ldots, N$, which is the number of data in the on-line phase. The loss matrix $\mathbf{A}$ is constructed by using the bounded convex loss function as indicated in Equation (2), where a monotonic transformation can be used for an unbounded loss function such as that in Equation (3). At each round $t = 1, \ldots, T$, a distribution $\mathbf{p}_t = (p_1^{(t)}, \ldots, p_M^{(t)}) \in \mathbb{R}^M$ over the base classifiers is obtained by $\tilde{p}_m^{(t+1)} = p_m^{(t)} \exp(-\eta \ell_m^{(t)})$, similar to the exponential weighted average algorithm in Section 2.2, and a vector $\boldsymbol{\ell}_t = \mathbf{A}\mathbf{e}_{i_t} = (\ell_1^{(t)}, \ldots, \ell_M^{(t)})$ represents an incurred loss vector, where $\ell_m^{(t)}$ is the loss associated with the classifier $f_m(x)$, $\mathbf{e}_i \in \mathbb{R}^N$ denotes the ith unit vector, and $\mathbf{e}_{i_t}$ is determined by the series of instances. Using this expression, we propose a new regret measure for our algorithm based on adversarial assumption, as presented in the following equation:

$$\mathcal{R}_T := \sum_{t=1}^{T} \mathbf{p}_t^T \mathbf{A}\hat{\mathbf{e}}_t - \min_{\mathbf{p}} \sum_{t=1}^{T} \mathbf{p}^T \mathbf{A}\hat{\mathbf{e}}_t, \tag{4}$$

where $\hat{\mathbf{e}}_t$ is defined in an adversarial manner as $\hat{\mathbf{e}}_t \in \arg\max_{\mathbf{e}_i} \mathbf{p}_t^T \mathbf{A}\mathbf{e}_i$, $i = 1, \ldots, N$. The regret in Equation (4) differs from the previous regrets in Equations (1) and (2) given that it considers the distribution over M candidate models instead of the single best model. Moreover, the regret in Equation (4) is slightly modified from regret measure in on-line optimization on the simplex [29] by introducing adversarial assumption to demonstrate the convergence in the worst case scenario. Therefore, the objective is to minimize the total cumulative loss that is incurred during the on-line phase (over T rounds) $\mathcal{L}_T = \sum_{t=1}^{T} L_t$ where $L_t = \sum_{m=1}^{M} p_m^{(t)} \ell_m^{(t)}$.

The following theorem shows that the weight vector $\mathbf{p}_t$ converges to a limiting vector. This process minimizes average total loss.

Theorem 1. *Assume that the loss function L is convex in its first argument and takes values within $[0,1]$. Then, $\mathbf{p}_t$ is generated by the aforementioned algorithm that converges to the limiting vector $\mathbf{p}_*$ for an appropriate selection of η. $\mathbf{p}_*$ is the optimal solution that minimizes total expected loss, $\min_{\mathbf{p}} \max_{\mathbf{q}} \mathbf{p}^T \mathbf{A}\mathbf{q}$.*

Proof. Let $\mathbf{p}_t$ be generated by the aforementioned algorithm with $\eta_t = \sqrt{(8 \ln M)/t}$. As $t \to \infty$, $\eta_t \to 0$, and $\hat{p}_m^{(t+1)} / p_m^{(t)} = \exp(-\eta_t \ell_m^{(t)}) \to 1$; therefore, $p_m^{(t+1)} = \hat{p}_m^{(t+1)} / \sum_{m=1}^{M} \hat{p}_m^{(t+1)} \to p_m^{(t)}$ for all $m = 1, ..., M$. Accordingly, $\mathbf{p}_t$ converges to the limiting vector $\mathbf{p}_*$. □

Approximately, for $T \geq 2^s - 1$, this selection of η_t consists of dividing time into periods $[2^k, 2^{k+1} - 1]$, $k = 0, ..., s$, and selecting $\eta_k = \sqrt{(8 \ln M)/2^k}$ in each period. By utilizing the proof of Theorem 7.7 in [27] and Corollary 6.4 in [30], we can show that

$$\frac{R_T}{T} := \frac{1}{T} \sum_{t=1}^{T} \mathbf{p}_t^T \mathbf{A}\hat{\mathbf{e}}_t - \min_{\mathbf{p}} \frac{1}{T} \sum_{t=1}^{T} \mathbf{p}^T \mathbf{A}\hat{\mathbf{e}}_t = O(\sqrt{\ln M / T})$$

which implies that the used on-line learning algorithm is a regret minimization algorithm; that is, $R_T/T \to 0$ as $T \to \infty$. Hence, the following holds:

$$\min_{\mathbf{p}} \max_{i=1,\ldots,N} \mathbf{p}^T \mathbf{A}\mathbf{e}_i \leq \max_{i=1,\ldots,N} \left(\frac{1}{T} \sum_{t=1}^{T} \mathbf{p}_t^T \mathbf{A}\mathbf{e}_i \right) \leq \frac{1}{T} \sum_{t=1}^{T} \max_{i=1,\ldots,N} \mathbf{p}_t^T \mathbf{A}\mathbf{e}_i = \frac{1}{T} \sum_{t=1}^{T} \mathbf{p}_t^T \mathbf{A}\hat{\mathbf{e}}_t$$

By definition of regret, the right-hand side can be expressed and bounded as follows:

$$\begin{aligned} \frac{1}{T} \sum_{t=1}^{T} \mathbf{p}_t^T \mathbf{A}\hat{\mathbf{e}}_t &= \min_{\mathbf{p}} \frac{1}{T} \sum_{t=1}^{T} \mathbf{p}^T \mathbf{A}\hat{\mathbf{e}}_t + \frac{R_T}{T} = \min_{\mathbf{p}} \mathbf{p}^T \mathbf{A} \left(\frac{1}{T} \sum_{t=1}^{T} \hat{\mathbf{e}}_t \right) + \frac{R_T}{T} \\ &\leq \max_{\mathbf{q}} \min_{\mathbf{p}} \mathbf{p}^T \mathbf{A}\mathbf{q} + \frac{R_T}{T} \end{aligned}$$

This implies that for the min-max of all $T \geq 1$ and $\lim_{T\to\infty} \frac{R_T}{T} = 0$, the following bound holds:

$$\min_{\mathbf{p}} \max_{i=1,\ldots,N} \mathbf{p}^T \mathbf{A}\mathbf{e}_i \leq \max_{\mathbf{q}} \min_{\mathbf{p}} \mathbf{p}^T \mathbf{A}\mathbf{q}$$

To demonstrate reverse inequality, the definition of min is adopted, and we have $\min_{\mathbf{p}} \mathbf{p}^T \mathbf{A}\mathbf{e}_i \leq \mathbf{p}^T \mathbf{A}\mathbf{q} \leq \max_{i=1,\ldots,N} \mathbf{p}^T \mathbf{A}\mathbf{e}_i$. Taking the maximum over $\mathbf{q}$ of both sides yields $\max_{\mathbf{q}} \min_{\mathbf{p}} \mathbf{p}^T \mathbf{A}\mathbf{e}_i \leq \max_{i=1,\ldots,N} \mathbf{p}^T \mathbf{A}\mathbf{e}_i$ for all $\mathbf{p}$, and subsequently, taking the minimum over $\mathbf{p}$ proves the inequality $\max_{\mathbf{q}} \min_{\mathbf{p}} \mathbf{p}^T \mathbf{A}\mathbf{q} \leq \min_{\mathbf{p}} \max_{i=1,\ldots,N} \mathbf{p}^T \mathbf{A}\mathbf{e}_i$. Therefore, we obtain

$$\min_{\mathbf{p}} \max_{i=1,\ldots,N} \mathbf{p}^T \mathbf{A}\mathbf{e}_i = \max_{\mathbf{q}} \min_{\mathbf{p}} \mathbf{p}^T \mathbf{A}\mathbf{q}$$

If we let $\hat{\mathbf{e}}_*$ maximize $\mathbf{p}_*^T \mathbf{A}\mathbf{e}_i$ over $i = 1, \ldots, N$, then from $R_T/T \to 0$, we derive

$$\mathbf{p}_*^T \mathbf{A}\hat{\mathbf{e}}_* = \min_{\mathbf{p}} \mathbf{p}^T \mathbf{A}\hat{\mathbf{e}}_* = \min_{\mathbf{p}} \max_{i=1,\ldots,N} \mathbf{p}^T \mathbf{A}\mathbf{e}_i = \max_{\mathbf{q}} \min_{\mathbf{p}} \mathbf{p}^T \mathbf{A}\mathbf{q} = \min_{\mathbf{p}} \max_{\mathbf{q}} \mathbf{p}^T \mathbf{A}\mathbf{q}$$

The last equality originates from von Neumann's minimax theorem. Therefore, $\mathbf{p}_*$ is the minimal solution for total expected loss.

3.2. On-Line Time Series Analysis

In the previous section, we focus on the on-line phase that updates the ensemble weights to analyze time series data when the candidate models are given. In this section, we introduce the overall framework for analyzing time series data using our algorithm. We conduct several experiments based on this framework in Section 4. Figure 1 illustrates the process that consists of the off-line and on-line phases.

When analyzing time series data, continuously incoming instances appear more realistic than the entire training data being given and fixed. Our algorithm is based on the former scenario. We cannot immediately utilize the incoming instances to learn the parameters of deep neural networks because training such networks requires adequate data. Therefore, we train the classifiers with the initial dataset, accumulate the incoming instances during the on-line phase, and update the training dataset with the accumulated instances. Our on-line ensemble deep learning algorithm enables reflecting the incoming instances to the ensemble model by updating the ensemble weights.

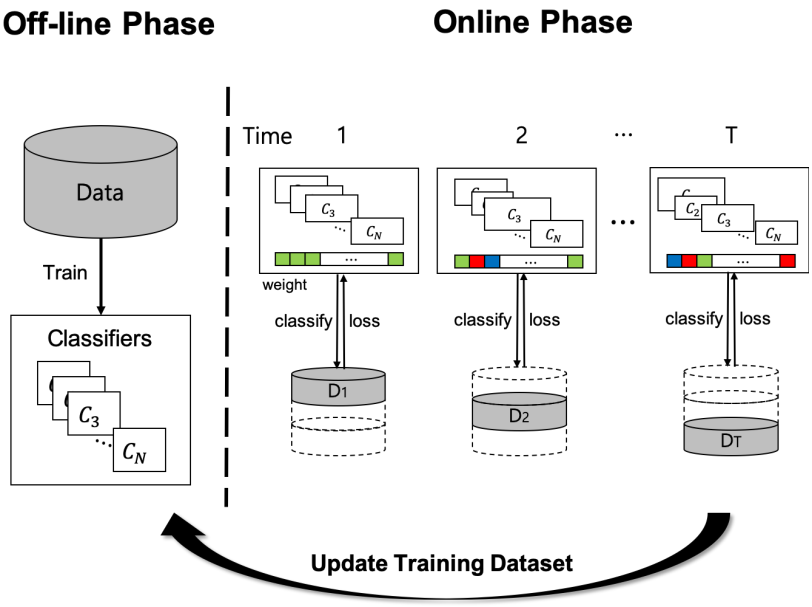


Figure 1. Overall framework for analyzing time series data using the on-line ensemble deep learning algorithm.

4. Experiment

4.1. Experimental Design

In this paper, we propose an on-line ensemble deep learning algorithm for a sustainable and robust analysis. This algorithm postulates continuously incoming training data. We verified the effectiveness of our algorithm by applying it to simulated and real-world time series data. Although our overall framework in Section 3.2 can be applied to continuously incoming time series data, we excluded the retraining phase and focused on one-time off-line and on-line phases. We generated simple time series data based on the sine function to explore the properties of our algorithm. We conducted experiments using financial and non-financial time series data to demonstrate the effect of our algorithm on various real-world examples. Financial time series examples consist of S&P 500, Nasdaq future, gold future, commodity future, and cryptocurrency. Non-financial time series data consist of temperature and power consumption.

The simulated and power consumption data are univariate time series. The others, namely S&P 500, Nasdaq future, gold future, commodity future, cryptocurrency, bankruptcy data and temperature, are multivariate time series. In this study, we concentrated on the classification problem of time series data even if most time series prediction models have focus on regression problems. Deep learning models have accomplished the state-of-the-art performances in classification problems, and up and down prediction can be effectively used. For univariate time series, we used x_t, x_{t-1}, ..., $x_{t-(k-1)}$, where k is the window size of the input to predict the target variable y_t, and constructed the base classifiers using the different window sizes from 1 to 6. The target variable of a univariate case was set to $y_t = 1$ if the time series value at time step x_{t+1} was greater than that at time step x_t to predict the trend (direction) at the next time step of the time series. For multivariate time series, we similarly constructed the base classifiers with $\mathbf{x}_t, \mathbf{x}_{t-1}, \ldots, \mathbf{x}_{t-k+1}$ where the input at time t, $\mathbf{x}_t$, is a vector. The target variable of the multivariate case was obtained by calculating y_t from one of the variables in $\mathbf{x}_t$. The base classifiers of the multivariate time series can use different variables, unlike the univariate case.

The experiments consisted of off-line and on-line phases. In the off-line phase, data from $t = 1$ to $t = l$ were used to train deep learning models, which were multilayer perceptrons with various configurations, such as the different network structures and the different input variables. We minimized cross entropy loss (Equation (3)) with an Adam optimizer [31]. During the on-line phase, our algorithm

adjusted ensemble weights depending on the losses of the base classifiers for a batch of D incoming instances from $t = l + 1$ to $t = T$ with $\eta = 10.0$. We divided the total data into 55 % of data for the off-line phase and 45 % of data for the on-line phase.

We measured the effectiveness of our algorithm in the on-line phase on the basis of accuracy, the precision, the recall, and area under the curve (AUC) of the receiver operating characteristics (ROC) curve because our algorithm was applied to the classification problems of time series data. We also examined the distribution of the ensemble weight to investigate the adaptability of the proposed algorithm.

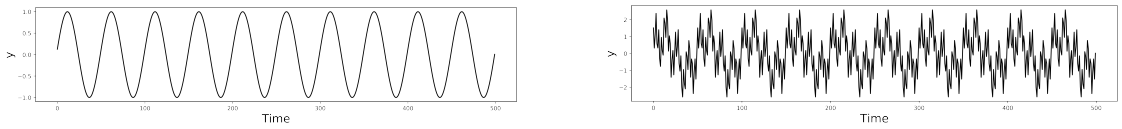
We compared our algorithm with other ensemble methods. The baseline method is a simple ensemble model of deep learning models with a fixed weight. We also learned tree-based ensemble methods, including random forest and gradient boosting. For implementation, we used the most popular machine learning library, namely, *scikit-learn* in Python, for the tree-based ensemble methods [32], and the *keras* library to train deep learning models [33].

4.2. Simulated Time Series Data

We utilized the simulated time series data before applying the proposed methodology to real data. The simulated time series was generated based on a sine functions, with a single sine function and a combination of sine functions with the different frequencies.

4.2.1. Data Description

We generated simple time series data from the function $x_t = \sin 0.04\pi t$ for $t = 1, \ldots, 10{,}000$. As mentioned in Section 4.1, we divided the data into the off-line phase $t = 1, \ldots, 5500$ and the on-line phase $t = 5501, \ldots, 10{,}000$. The target variable is nearly balanced, where the ratio of $y = +1$ is 47.9%. Figure 2a shows the generated time series of the simple sine function. We also generated a complex sine function, which is a combination of three simple sine functions with different cycles $x_t = \sin 0.04\pi t + \sin 0.16\pi t + \sin 0.64\pi t$. In total, 10,000 data were generated, similar to the simple sine function. The time periods of the off-line and on-line phases were the same as that of the simple sine case, but the class distribution of the target variable was changed to 32.2%. Figure 2b shows the generated time series of the combined sine function.

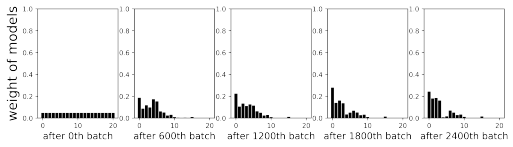


(**a**) Sine function. (**b**) Composite sine function.

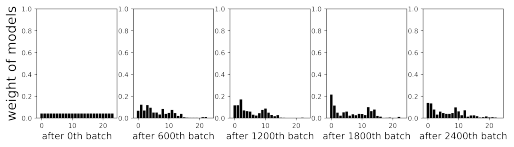
Figure 2. Sine function and composite sine function.

4.2.2. Results

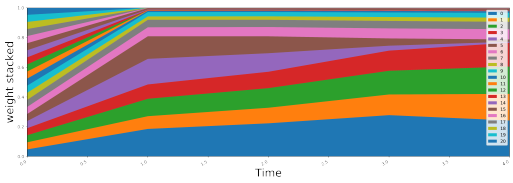
In this section, we present the experimental results of the simulated time series data to verify the significance of our algorithm. Figure 3 presents the illustrative change in ensemble weight and the quantitative performance measures, such as accuracy, precision, recall, and AUC, over time during the on-line phase. Figure 3a,b shows that the distribution of the ensemble weight changed even when the initial distribution was uniform. For both the simple sine and the combined sine, we found that the weights were adjusted to improve the performance of the ensemble models in Figure 3c–l.



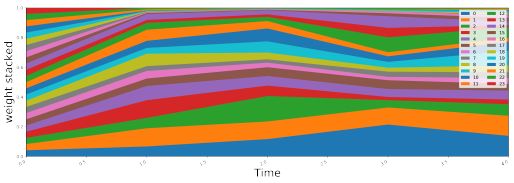
(**a**) Sine: Change of weight distribution



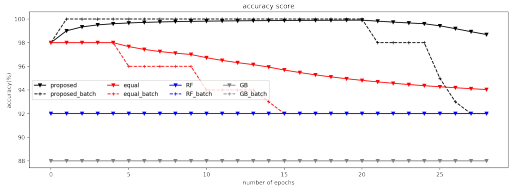
(**b**) Combined sine: Change of weight distribution



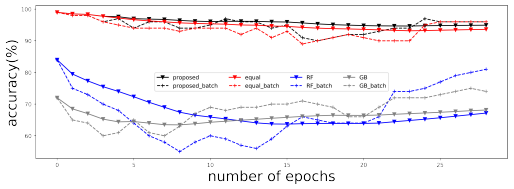
(**c**) Sine: Stack graph of weights over time



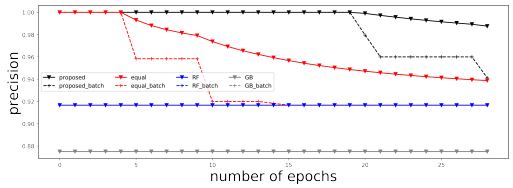
(**d**) Combined sine: Stack graph of weights over time



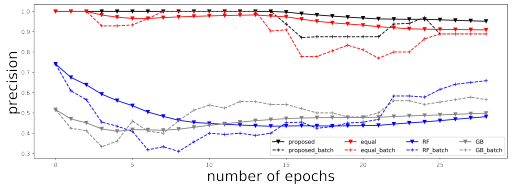
(**e**) Sine: Accuracy



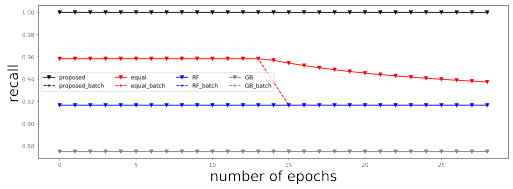
(**f**) Combined sine: Accuracy



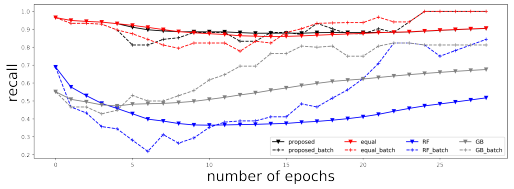
(**g**) Sine: Precision



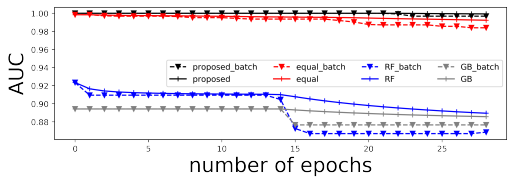
(**h**) Combined sine: Precision



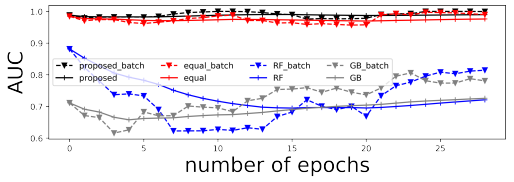
(**i**) Sine: Recall



(**j**) Combined sine: Recall



(**k**) Sine: AUC



(**l**) Combined sine: AUC

Figure 3. Performance measures of sine and combined sine.

For the simulated data, deep learning models with varying performance constitute the ensemble models. In Figure 3a,b, the models on the left exhibited better performance than the models on the right. Consequently, the weight of the models on the right decreased even when the distribution of the weights of the models on the right was also changed in accordance with the change in incoming instances. This tendency was evident for the simple periodic time series data, as shown in Figure 3a. Figure 3c,d presents the stack graphs of the ensemble weight over time for the sine and combined sine examples, respectively, where the height of the graph represents the weight of each classifier, and the sum of weights is 1. The proposed algorithm aims to increase the proportion of classifiers that perform efficiently over time and to reduce the weight of classifiers that exhibit poor performances

in adapting to incoming instances. Our ensemble algorithm can compensate for relatively inferior classifiers, whereas most ensemble models can be easily deteriorated by such classifiers.

Figure 3e,f shows the predicted and cumulative accuracies for the sine and combined sine cases, where the incoming instances were considered with an instance or a batch as a unit. In both examples, the difference between the cumulative batch accuracy of the proposed model and that of the equal weight model increased with time. That is, the proposed model exhibited a more stable performance than other methodologies. However, the accuracy and the precision of the proposed method also eventually decreased, as shown in Figure 3c–f. This result implies the necessity of our overall framework in analyzing time series data. The performances of random forest and gradient boosting remained unchanged because the simulated data were periodic, and the ensemble models were not updated. For the simulated data, the deep learning based ensemble models were significantly better than the tree-based ensemble models.

Figure 3g–j shows the precision and the recall calculated by type 1 and type 2 errors. The deep learning-based ensemble models had the higher precision and recall scores than the tree-based models, although they decreased in the later part. In the case of the combined sine example, the tree-based models had good recall score, but their accuracy and precision scores were considerably lower than those of the deep learning ensemble models. Figure 3k,l shows the AUC of sine and composite sine, respectively. In both cases, the AUC of the proposed model achieved the best performance, which was close to 1.

As mentioned in Section 4.1, we fixed the hyperparameter $\eta = 10$. We examined the role of η, which was used to adjust the effect of loss when updating the weight in $\tilde{p}_m^{(t+1)} = p_m^{(t)} \exp\left(-\eta \ell_m^{(t)}\right)$, to justify the fixed value. Figure 4 shows the accuracy of the proposed ensemble model for different ηs, with η changing from 0.001 to 100.0 in the log scale. Unexpectedly, the accuracy scores of $\eta = 0.001, \ldots, 1.0$ hardly changed even when η changed in the log scale. The accuracy score substantially increased when η increased from 1.0 to 10.0. Changing $\eta = 10.0$ to $\eta = 1000.0$ did not drastically improve the accuracy. Therefore, we set the hyperparameter η to 10.0 in all cases.

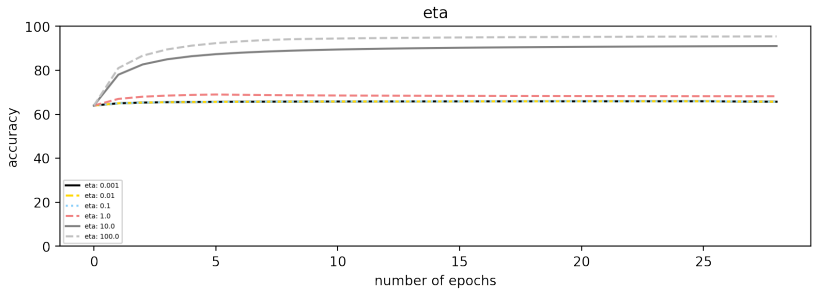


Figure 4. Role of η in the proposed model.

4.3. Financial Time Series Applications

We applied our algorithm to real financial time series data. Real financial time series data are typically aperiodic and complex given that they can exhibit a trend, a nonstationary property, or a sudden drift. Therefore, we expect that our algorithm can help stably analyze the time series data during the on-line phase.

4.3.1. Data Description

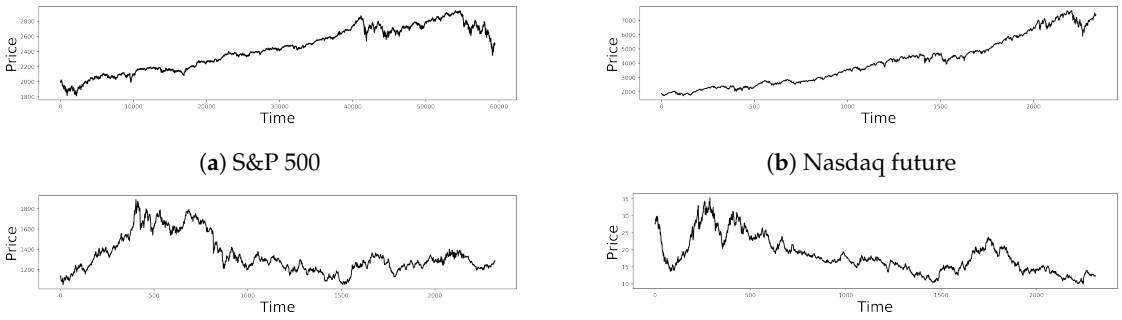
We used S&P 500, Nasdaq future, gold future, and sugar future datasets to verify the effectiveness of the proposed method for financial time series data. The S&P 500 data were collected at 5-min intervals from January 2016 to December 2018. The other datasets were collected at one-day intervals from January 2010 to December 2018. Nasdaq data were obtained from https://investing.com/. Table 1 summarizes the basic information of each dataset, such as the number of instances, frequency,

target, and predictive variables. For S&P 500, gold, and, sugar, the predictive variables contain the spot indexes, such as open price, low price, high price, close price and volume information. Eight additional variables were extracted from the time series of close price, namely, RDP (−5), MA (5), MA (10), EMA (5), OSCP, EOSCP, DISP (5), and EDISP (5), based on the work in [34]. In addition, for Nasdaq, nine variables were extracted from the time series, namely OBV, MA(5), BIAS(6), PSY(12), ASY(5), ASY(4), ASY(3), ASY(2), ASY(1), based on the work in [35]. The explanation of these variables is provided in Tables A1–A3 in Appendix A. The target variable was constructed using close price.

Table 1. Description of financial data.

Data	Predictor	Target	Freq	# of Data
S&P 500	13 variables in Appendix A	trend of price	5 min	59,453
gold future			daily	2246
commodity—sugar future				2252
Nasdaq future	9 variables in Appendix A			2297
cryptocurrency—Bitcoin	17 variables in Appendix A			2683
bankruptcy—savings bank	38 variables in Appendix A	default or not	quarterly	4225

Figure 5 shows the four financial time series datasets that we used. S&P 500 and Nasdaq future present increasing trends and sudden drops, with the irregular patterns mostly concentrated in latter periods. The ratios of $y = 1$ for S&P 500 and Nasdaq future in on-line phase are 51.3% and 56.0%, respectively. The gold future and sugar future datasets exhibit different properties from the previous examples because they do not demonstrate a consistent trend. However, they have sudden drops and rises mostly during the early periods. The ratios of $y = 1$ for gold future and sugar future in on-line phase are 49.3% and 47.2%, respectively.



(**a**) S&P 500 (**b**) Nasdaq future

(**c**) Gold future (**d**) Sugar future

Figure 5. Financial time series examples.

Bitcoin price data were obtained from September 2011 to August 2019 through an on-line source (https://Bitcoincharts.com/markets/). Bitcoin price was used at intervals of one day and was from bitstamp exchanges. Bitcoin price was averaged over daily open price, close price, lowest price and highest price to calculate the target variable. Using this price, we produced a binary class variable with a value of 1 if the price is increased the next day and 0 otherwise. The ratio of $y = 1$ for Bitcoin in on-line phase is 55.9%. We used 17 variables as input to predict the target variable, as explained in the Table A1 in Appendix A. The predictive variables were used to train the base learners of the proposed model during the off-line phase. Among these variables, crude oil, SSE, gold, VIX, and FTSE100 were obtained from https://finance.yahoo.com/; and USD/CNY, USD/JPY, and USD/CHF were obtained from https://ofx.com/en-au/forex-news/historical-exchange-rates/. We also used various blockchain-related variables explained in the Table A1 in Appendix A. Such data were obtained from https://blockchain.info/.

Figure 6 shows Bitcoin daily price, which exhibits considerable changes. For example, the value of 1 Bitcoin was approximately $5 in September 2011 but reached approximately $4000 in August 2017 when market volatility became exceptional. However, after hitting approximately $20,000, the price started to decrease with high volatility. Therefore, the later part is important in analyzing Bitcoin price.

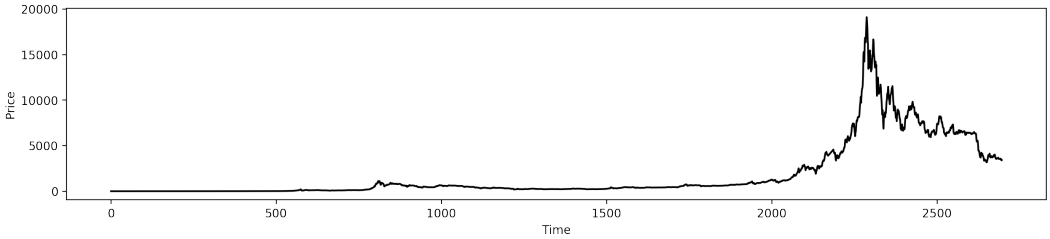


Figure 6. Bitcoin.

4.3.2. Results

Figure 7 shows the accuracy and the weight change over time for each financial example. The accuracy of the financial time series example (from 50% to 60%) was considerably lower than the accuracy result of the example of the toy data. In an efficient market, stocks follow a fairly random walk. Hence, a single base classifier would achieve a slightly better performance than a trivial classifier that predicts only one value. This can lead to the small improvement of ensemble model in the case of financial time series such as index, future, and asset. Nonetheless, to clarify the effectiveness of our ensemble algorithm, we compared all ensemble methods with the trivial classifier, whose final cumulative accuracy represents the ratios of y = 1 in Figure 7. Table 2 shows the average difference between the performance of the proposed method and the performance of a single base learner that predicts only one value for all the batches for all the examples. The performance of a single base learner exhibited a degree of imbalance in each example. By contrast, the proposed model achieved the best performance for all examples.

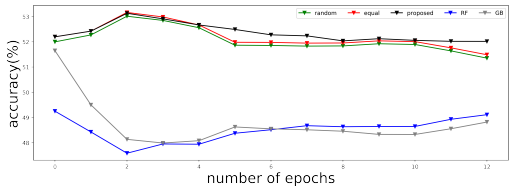
Table 2. The average difference (%).

	Sine	Combination Sine	S&P 500	Nasdaq	Gold	Sugar	Bitcoin	Temperature	Power Consumption
proposed	**46.69**	**27.12**	**0.66**	**0.59**	**0.06**	**0.73**	**0.86**	**6.83**	**18.66**
equal	42.03	25.81	0.13	0.29	−0.44	−1.07	−0.14	2.97	17.8
rf	39.99	−0.64	−0.46	−5.57	−0.66	−1.97	−5.47	3.15	−1.02
bg	35.99	0.33	−0.17	−1.17	−0.14	−2.97	−1.64	4.9	2.21

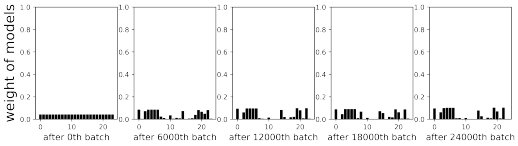
Figure 7a,c,e–g demonstrates the cumulative accuracies of the ensemble models and the trivial classifier. Our on-line ensemble algorithm outperformed the other ensemble models not only on all datasets but also at most time steps. For the S&P and Nasdaq datasets, deep learning based ensemble algorithms outperformed other ensemble methods, and our ensemble algorithm slightly outperformed the equal ensemble and the trivial classifier. In Appendix B, we add Figure A1 to compare our proposed method with the equal ensemble and the trivial classifier in more detail. As shown in Figure A1, we found that our algorithm had better performances in the later part of the S&P despite of high volatility, as shown in Figure 5. On the other hand, gold and sugar future datasets exhibited obvious improvements compared to the other methods. This shows that our method can efficiently adapt to the change of input distribution by updating the ensemble weights.

Figure 7b,d,f,h,j presents graphical representations of how weight changes over time for each financial example. Similar to the previous toy data example, the weight of each classifier changed with time to recognize the pattern change inherent in the data over time. In the case of the S&P 500, 5-min data were used, as described above, and the remaining examples used daily data. S&P 500 was intended as an example of high-frequency trading. Consequently, the weight assigned to each

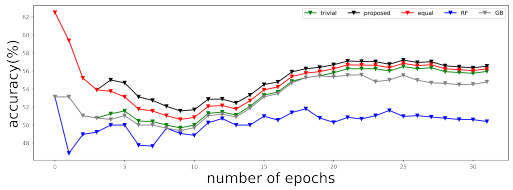
classifier changed with time in all the examples, regardless of whether it is a high-frequency trading scenario or a daily scenario. In particular, in the beginning of the on-line phase, a significant change in weight distribution occurred starting from the initial uniform distribution; thereafter, a slight weight distribution change was observed. In the case of financial time series, except for Bitcoin, the process of each time series followed a random walk, and a considerable change did not occur in weight distribution after finding the appropriate weight distribution by updating with the error of the initial state. Once the appropriate weight distribution is found, performance cannot improve through weight change. In the Bitcoin example, weight changed dramatically over time, presumably because the patterns inherent in Bitcoin data are more likely to vary with time than other financial time series data. That is, the Bitcoin process changed in a manner that follows a different type of distribution over time compared with other time series processes.



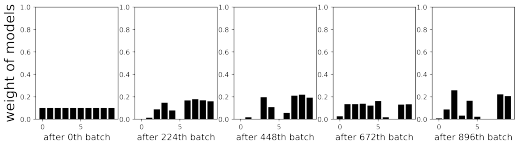
(**a**) S&P 500: Accuracy



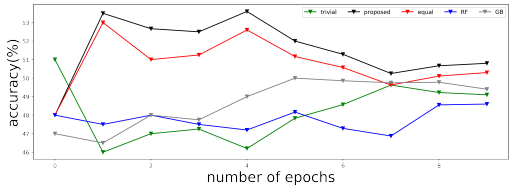
(**b**) S&P 500: Change of weight over time



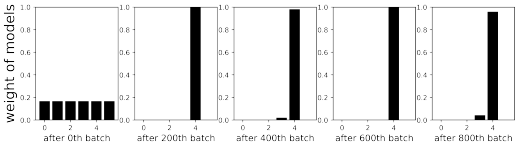
(**c**) Nasdaq: Accuracy



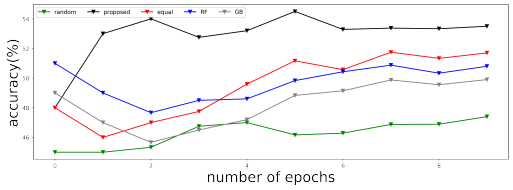
(**d**) Nasdaq: Change of weight over time



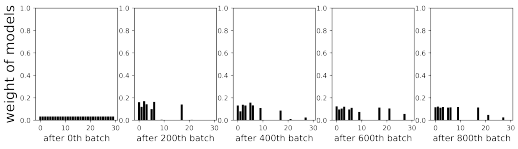
(**e**) gold: Accuracy



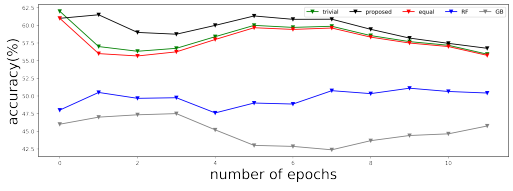
(**f**) gold: Change of weight over time



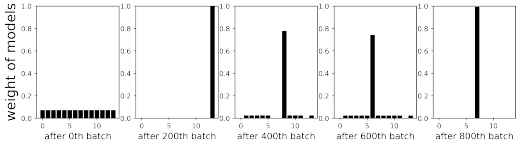
(**g**) sugar: Accuracy



(**h**) sugar: Change of weight over time



(**i**) Bitcoin: Accuracy



(**j**) Bitcoin: cChange of weight over time

Figure 7. Performance measures of financial time series data.

4.4. Non-Financial Time Series Applications

In this section, we explored time series data with different properties from the financial time series data in the previous section. We compared the performance of our algorithm with those of other ensemble algorithms during the on-line phase through a classification task.

4.4.1. Data Description

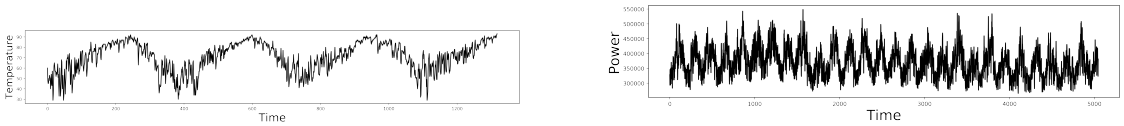
We tested our method on two non-financial time series datasets, namely temperature and power consumption, as shown in Table 3. This table presents the predictor, target, frequency and number of instances for non-financial data for each example. The temperature dataset represents the weather information for Austin, Texas and comprises data from December 2013 to July 2017. This dataset was obtained from https://www.kaggle.com/grubenm/austin-weather. Each instance included 18 numerical variables and two categorical variables, but we only used 15 numerical variables and excluded those that directly indicate temperature as input variables for the experiment. The predictive variables were divided into five categories: dew point, humidity, sea level pressure, visibility, and wind. The detailed information of the variables is shown in Table A1 in Appendix A.

Table 3. Description of non-financial data.

Data	Predictor	Target	Freq	# of Data
temperature	18 variables in Appendix A	trend of temperature	daily	1304
power consumption	historical value	trend of consumption		5055

The power consumption dataset is based on data from the American Electric Power (AEP) at https://www.kaggle.com/robikscube/hourly-energy-consumption. AEP is among the largest generators of electricity in the USA, and it delivers electricity to more than five million customers in 11 states. The dataset contains the power consumption from December 2004 to January 2018. Data are recorded in 1-h timestamp; hence, the data for the same day were summed to change the unit into day. As a univariate dataset, the power consumption values from the previous day and two days before were used as input. Therefore, the preprocessing step was similar to that of the simulated data. For both datasets, we created binary variables in the same manner as the financial datasets to use them as target variables. In addition, the ratios of $y = 1$ for temperature and power consumption in on-line phase are 59.7% and 52.8%, respectively.

Figure 8 illustrates temperature and power consumption data. They appear different from the previous financial time series datasets. They exhibit seasonality, but minimal trend, where the frequency of power consumption data is higher than that of temperature data. Therefore, the algorithms should adapt to these characteristics of time series data during the on-line phase. We did not use tricks such as removing or reflecting seasonality when training the models during the off-line phase. The experimental results of the datasets are provided in the following section.

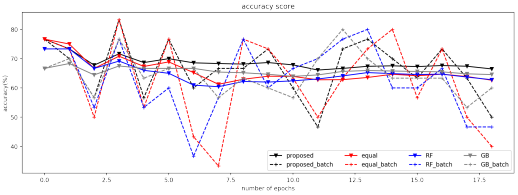


(**a**) Temperature (**b**) Power consumption

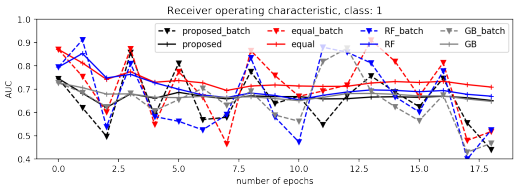
Figure 8. Temperature and power consumption.

4.4.2. Results

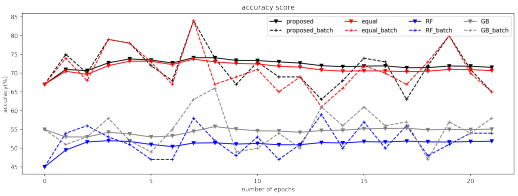
We tested the capability of our algorithm in predicting the direction of time series data. Figure 9 presents the comparison of prediction performance using accuracy and AUC. Figure 9a,c presents the accuracy scores of the temperature and power consumption datasets, respectively. Figure 9b,d corresponds to their AUC scores over time. As shown in Figure 9a,b, our algorithm was slightly better than the other algorithms for temperature data, even if the difference was minimal. In the case of power consumption data, our algorithm substantially outperformed the other algorithms, as shown in Figure 9b,d. Figure 8 depicts that power consumption data are volatile compared with temperature data. Thus, training the models to adapt well to a slow change in time series was easier than training the models for volatile time series during the off-line phase. The initial accuracies of temperature data were higher than the initial accuracies of power consumption data. In addition, deep learning-based models were better at predicting complex data compared with tree-based ensemble models.



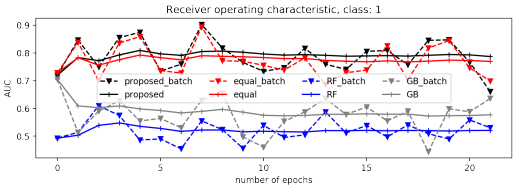
(**a**) Temperature: Accuracy



(**b**) Temperature: AUC



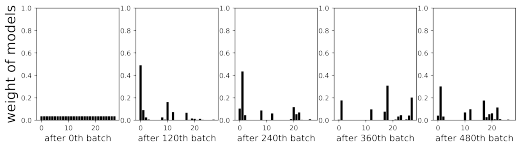
(**c**) Power: Accuracy



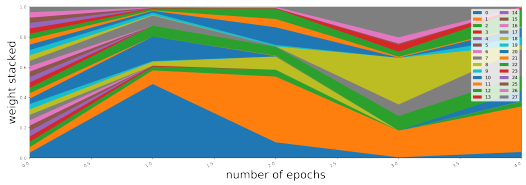
(**d**) Power: AUC

Figure 9. Performance measures of non-financial time series data.

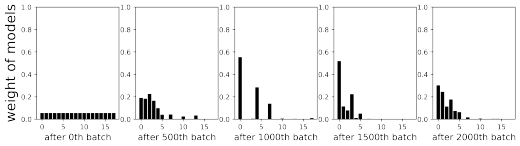
We visualize the change in ensemble weight over time in Figure 10 to examine the adaptiveness of our algorithm during the on-line phase. Figure 10a,b shows the ensemble weight of temperature data, and Figure 10b,d shows the ensemble weight of power consumption data. The distribution of the ensemble weight changed over time in both cases. The changes were helpful for adapting to the change in the characteristic of time series data without using other techniques.



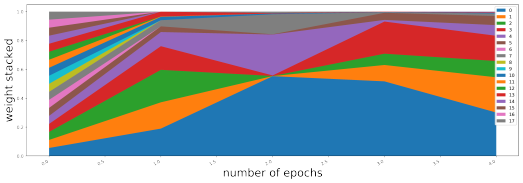
(**a**) Temperature: Change of weight distribution



(**b**) Temperature: Stack graph of weights over time



(**c**) Power: Change of weight distribution



(**d**) Power: Stack graph of weights over time

Figure 10. The change of the distribution of the ensemble weight for non-financial time series data (temperature and power consumption).

5. Discussion

As shown in Section 4, we verified our algorithm for real time series data with different properties, such as trend, seasonality, and sudden drift, by comparing it with other ensemble methods. In this section, we discuss the effectiveness of our algorithm by assuming specific useful scenarios. As mentioned in Section 3, our algorithm can be robust to intentional attacks and sustainable in data distribution change. We examined these properties of our algorithm through two illustrative examples.

5.1. Robustness for the Intentional Attacks

Many researchers have conducted studies in recent years on developing a system that is robust to external malicious attacks, which cause artificial intelligence to fail in machine learning applications. An adversarial attack is a representative study related to addressing the problem of fragile artificial intelligence with slight noises in an off-line environment. However, our study focused on developing a robust system when several classifiers are attacked during the on-line phase. This study can be important to applying deep learning system to real world applications that deal with time series or continuously incoming data. Therefore, we tested the robustness of the proposed model against malicious attacks when several base classifiers were attacked during the on-line phase. We used the sine example presented in Section 4.2 for verification. We postulated that the prediction of the attacked classifier was reversed to simplify the attacking scenario.

Figure 11 shows the effect of the number of attacked classifiers on the accuracy of the proposed model. The number of base deep learning models was 12, and we increased the number of the attacked models from 0 to 11 when the attack started in the 50th epoch. Until two attacked models, the accuracies of the ensemble model were maintained without significant degradation, while the accuracies decreased only slightly to five attacked models. The performance of the ensemble model sharply decreased when the number of attacked models reached six. Although performance initially dropped, accuracies recovered to a certain extent at eight attacked models. These results show that our algorithm could prevent the ensemble model from being deteriorated by attacked models during the on-line phase.

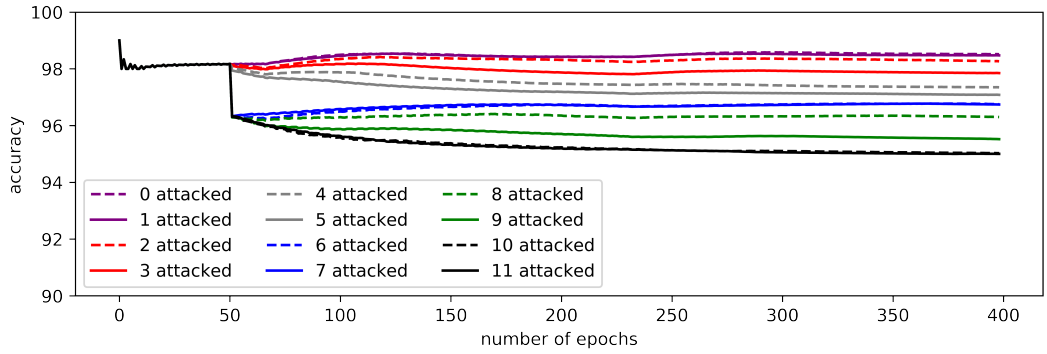
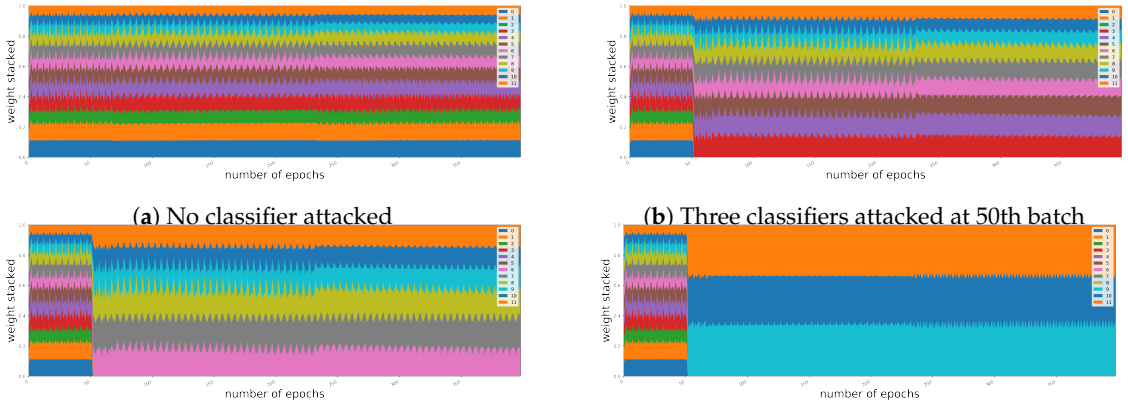


Figure 11. Accuracy depending on the number of attacked classifier.

Figure 12 illustrates the change in the distribution of ensemble weights over time, where Figure 12a–d presents zero, three, six and nine attacked classifiers, respectively. The intentional attacks resulted in a drastic change in weight distribution. The weights of the attacked models disappeared immediately after the attack in all cases because our attack scenario was strong. Our ensemble model could exclude the attacked classifiers, thus the on-line system could remain robust. This property can be helpful in detecting attacked models, and it can be reflected in the subsequent off-line phase.



(**a**) No classifier attacked

(**b**) Three classifiers attacked at 50th batch

(**c**) Six classifiers attacked at 50th batch

(**d**) Nine classifiers attacked at 50th batch

Figure 12. Accuracy and change of weight distribution of attacking scenario.

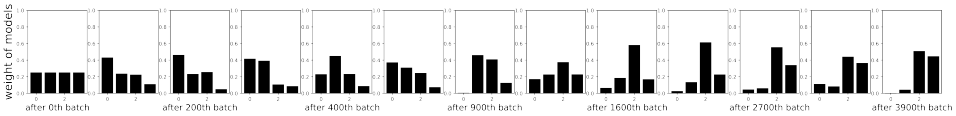
5.2. Sustainability for the Change of Target Distribution

In the previous experiments, we explored the effectiveness of our algorithm when the distribution of input variable changes over time. However, in an on-line scenario, a change in the distribution of the target variable is an important problem in addition to the change in input distribution. This problem is typically addressed using transfer learning methods, but our on-line approach can cope with it to a certain extent. Thus, we designed an experiment to verify the sustainability for the change in target distribution. We used the bankruptcy data explained in Table 1. The financial statement data of the Korean Savings Bank were obtained from the Korean Financial Supervisory Service (KFSS). The data of 133 Korean savings banks were collected from June 2004 to September 2016. In total, 38 variables were collected based on the work in [36] and used as input variables that reflect the various financial statuses of savings banks. The 38 variables were divided into seven categories: stability, profitability, growth, productivity, capital adequacy, liquidity, and asset quality. We attempted to estimate the default risk of savings banks through these variables because they are considered appropriate variables for learners to learn the default risk of a savings bank. This is because the aforementioned variables, as classified into seven categories, can represent the default risk of savings banks by reflecting their financial soundness and various statuses. We constructed the target variable with whether a savings bank breaks down after three steps. The data for the target class $y = 1$ (bankruptcy) was oversampled: the ratio of $y = 1$ data instances to the total data is 20%, 30%, 40%, and 50%. Data augmentation was performed with SMOTE [37].

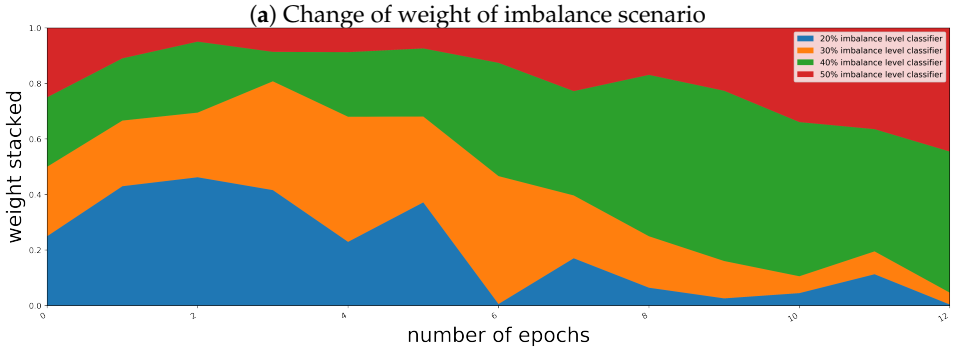
In the previous experiments, we qualitatively verified that our algorithm can adapt to a change in data distribution by analyzing the change of the ensemble weights. From these results, a question arises: Can the proposed model remain sustainable in target distribution change? In this experiment, we proved this feature by artificially changing the target distribution of bankruptcy data. The experimental design is as follows. The target variable is binary; hence, we adjusted the imbalance degree the of target variable to 20%, 30%, 40%, and 50%. During the on-line phase, we changed the target distribution by changing the degree of data imbalance. We postulated that the base classifiers of the ensemble model are diverse, similar to other ensemble methods. During the off-line phase, four classifiers were learned by training data with different distributions that have imbalance levels of 20%, 30%, 40% and 50%. During the on-line phase, the transition of target distribution was 20%, 30%, 40%, and 50% imbalanced. We found that, when the weight of the classifier corresponding to the imbalance level increased and the others decreased, the imbalance level changed in an on-line manner.

Figure 13a illustrates that the weight of a classifier at 20% level increased first from the initial equal weight. Then, the weight of the 30% level classifier increased, and the weight of the 40% level classifier also increased. Finally, the weight of the classifier at 50% level increased. In Figure 13b, the weight of the 20% level classifier gradually decreased with time. By contrast, the weight of the classifier at the

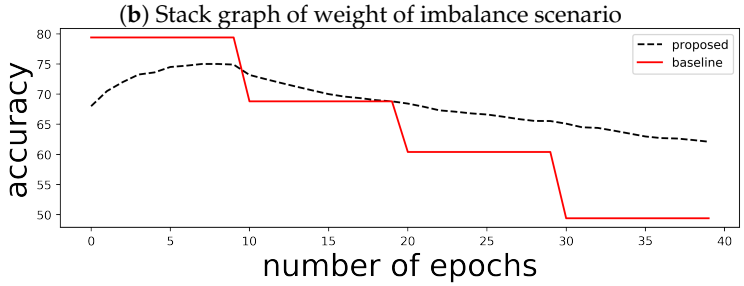
50% level increased over time. Consequently, when distribution changed, the weight assigned to each classifier in the ensemble model was appropriately adjusted. Therefore, the proposed model could make an accurate prediction. Figure 13c shows the change in the accuracy of the proposed model and the baseline. This result indicates that imbalance level changed over time. Although the accuracy of the proposed method was lower than the accuracy of the baseline for the first 20% level, its accuracy was higher than the accuracy of the baseline in other levels. Therefore, even when the distribution or pattern of data inherent in the time series data changed with time, the proposed ensemble model could determine the changed distribution and assign appropriate weights to the classifiers for the entire ensemble model to achieve sustainability in data distribution change. Accordingly, our on-line ensemble algorithm could reflect the change in data distribution to the ensemble model when base classifiers are diverse. This result also implies the importance of the composition of base deep learning models.



(**a**) Change of weight of imbalance scenario



(**b**) Stack graph of weight of imbalance scenario



(**c**) Accuracy of imbalance scenario

Figure 13. Performance measure of imbalance scenario.

6. Conclusions

In this paper, we propose an on-line ensemble learning algorithm that changes the weight distribution of the ensemble model on the basis of loss value to adapt to a change in the properties of incoming instances. Our algorithm can be used as a general framework for aggregating deep learning models to analyze time series data because it is applicable to all cases that learn a model by minimizing the loss function. We selected deep learning models as base classifiers because they minimize the loss function without constraints but with a regularizer in the loss function. We developed our algorithm with motivation from on-line learning, devised a new regret measure based on the adversarial assumption for our algorithm, and proved that our algorithm can make the distribution of ensemble weight converge to the limiting vector that minimizes total loss. In addition, we suggest an overall framework to apply our algorithm to real-world systems to analyze time series data. This framework enables systems to remain sustainable with continuously incoming big data. In the

experiment, we demonstrated the effectiveness of our algorithm by focusing on the on-line phase. We applied our algorithm to the simulated data, financial time series data, and non-financial time series data, which exhibit various characteristics, such as high volatility, periodicity, trend, and sudden drift. The ensemble method based on deep learning models outperformed other models in most cases, and the visualization of the weight distribution illustrated how our algorithm works. We also discussed the effect of our algorithm on special scenarios related to the robustness and sustainability of the algorithm for extreme cases. The adjustment of ensemble weight on the basis of the loss function can detect and address the degradation problem to a certain extent.

We conducted experiments for classification tasks, but our algorithm can be extended to regression problems when using deep learning models with appropriate loss functions. We used simple multilayer perceptrons as base classifiers, but deep learning models that consider the sequence of instances, such as RNNs, can improve the prediction performance of our algorithm. Moreover, the use of RNNs would enable expanding our method to apply to time series classification problems that predict the target class using sequences of instances as inputs. In the future, we aim to extend our algorithm to unsupervised problems, such as anomaly detection, by constructing the appropriate loss functions.

Author Contributions: S.P. and J.L. conceived and designed the analysis. S.P. and J.L. developed and elaborated the theoretical analysis. H.K., J.B. and B.S. collected the data. H.K. carried out the experiment. S.P. took the lead in writing the manuscript. S.P. and H.K. discussed the results and contributed to the final manuscript. S.P. supervised the findings of this work.

Funding: This work was supported by the National Research Foundation of Korea (NRF) grant funded by the Korean government (MEST) (No.2018R1D1A1A02085851 and 2016R1A2B3014030). In addition, part of this research was performed while the authors (Saerom Park and Jaewook Lee) were visiting the institute for Pure and Applied Mathematics (IPAM), which is supported by the National Science Foundation.

Conflicts of Interest: The authors declare no conflict of interest. The funders had no role in the design of the study; in the collection, analyses, or interpretation of data; in the writing of the manuscript, or in the decision to publish the results.

Appendix A. Predictive Variables

Table A1 provides the detailed explanations for predictive variables used in the multivariate time series. Thirteen variables were used in the S&P 500, gold, and sugar datasets, as shown in Table A2, and nine variables were used in Nasdaq dataset, as shown in Table A3. Tables A2 and A3 not only present the detailed description of predictive variables but also the formula used to obtain the variables. These variables are based on the work in [34,35]. In the Bitcoin example, 17 variables, including the blockchain variable, were used. The savings bank and temperature datasets have 38 and 18 predictive variables, respectively, among which the variables appropriate for a given task were considered.

Table A1. The detailed explanation of predictive variables.

Data	Predictors
S&P 500, gold, sugar	Open price, Low price, High price, Close price, Volume, RDP (−5), MA (5), MA (10), EMA (5), OSCP, EOSCP, DISP (5), EDISP (5)
Nasdaq	OBV, MA (5), BIAS (6), PSY (12), ASY (5), ASY (4), ASY (3), ASY (2), ASY (1)
cryptocurrency—Bitcoin	Crude oil, SSE, Gold, VIX, FTSE100,USD/CNY, USD/JPY, USD/CHF, Historical value of target variable, Trading volume, Average block size, Median confirmation time, Hash rate, Miners revenue, Cost per transaction, Confirmed transactions per day, The number of transaction excluding popular addresses
Savings bank data	Total credit, Total credit growth rate, Loan growth rate, Allowance for loan losses, Total subordinated loans to total loans, Allowance for bad debts to subordinated loans, Net non-performing loans ratio, Overdue ratio, Petty overdue ratio, Non-performing loans, Loan deposit ratio, Net loans to total assets, Net loans against liquid liabilities, Available funds ratio, Operating expenses to operating income, Operating return on assets, Current account ratio, BIS capital adequacy ratio, Net loan-to-equity, BIS simple capital adequacy ratio, Tangible common equity ratio, Total investment efficiency, Labor income share, Total asset growth rate, Tangible asset growth rate, Net income growth rate, Equity capital growth rate, Total net income, Return on assets, Return on equity, Interest coverage ratio, Capital ratio, Debt ratio, Liquidity ratio, Quick ratio, Payables dependency, Total assets, Debt repayment coefficient
Temperature	High temperature, Average temperature, Low temperature, Highest wind speed gust, High dew point, Low visibility, High wind speed, Average wind speed, Low sea level pressure, High visibility, Average visibility, Average dew point, Low dew point, High humidity, High sea level pressure, Low humidity, Average sea level pressure,Average humidity

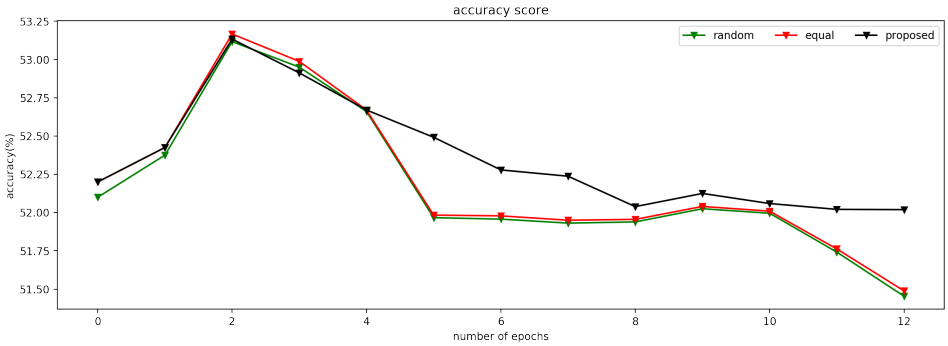
Table A2. The detailed description and formula of financial examples except for Nasdaq.

Variables	Description	Formula
RDP ($-k$)	Relative difference prices between k times before and now in percentage.	$\frac{p(t)-p(t-k)}{p(t-k)} \times 100$
MA (k)	Moving average of k times.	$\frac{\sum_{i=0}^{k-1} p(t-i)}{k}$
EMA (k)	Exponential moving of k times.	$\alpha \sum_{i=0}^{k} (1-\alpha)^i p(t-i)$ where $\alpha = \frac{2}{k+1}$
DISP (k)	k-time disparity: the relative difference in percentage between k-time moving average and the current price.	$\frac{p(t)-MA(k)}{MA(k)} \times 100$
EDISP (k)	]@c@k-time disparity with exponential moving average and the current price.	$\frac{p(t)-EMA(k)}{EMA(k)} \times 100$
OSCP	Price oscillator: the relative difference between 5-time moving average and 10-time moving average.	$\frac{MA(5)-MA(10)}{MA(5)}$
EOSCP	Price oscillator with exponential moving average: the difference between 5-time exponential moving average and 10-time exponential moving average.	$\frac{EMA(5)-EMA(10)}{EMA(5)}$
RSI (k)	Relative strength index for k-times	$1 - \frac{1}{1+\sum_{i=0}^{k-1} \frac{u(t-i)}{k} \sum_{i=0}^{k-1} \frac{d(t-i)}{k}}$ where $u(t)$ is 1 if there is upward-price change at time t and is 0 otherwise, and $d(t)$ is 1 if there is downward -price change at time t and is 0 otherwise.

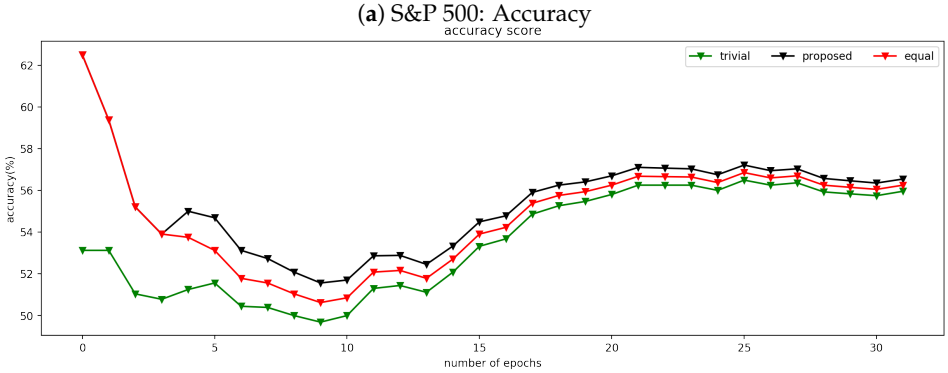
Table A3. The detailed description and formula of Nasdaq.

Variables	Description	Formula
OBV	On Balance Volume. V_t is volume at time t	$OBV_t = OBV_{t-1} + \theta \cdot V_t$
MA_5	Moving average of 5 times.	$MA_5 = \frac{\sum_{i=1}^{5} C_{t-i+1}}{5}$
$BIAS_6$	C_t is close price at time t	$BIAS_6 = \frac{C_t - MA_6}{MA_6} \cdot 100\%$
PSY_{12}	The ratio of the number of rising periods over the n day. A is the number of rising days in the last n days.	$PSY_{12} = \frac{A}{12} \cdot 100\%$
ASY_5	The average return in the last 5 days. SY_t is log return	$ASY_5 = \frac{\sum_{i=1}^{5} SY_{t-i+1}}{5}$
ASY_4	The average return in the last 4 days.	$ASY_4 = \frac{\sum_{i=1}^{4} SY_{t-i+1}}{4}$
ASY_3	The average return in the last 3 days.	$ASY_3 = \frac{\sum_{i=1}^{3} SY_{t-i+1}}{3}$
ASY_2	The average return in the last 2 days.	$ASY_2 = \frac{\sum_{i=1}^{2} SY_{t-i+1}}{2}$
ASY_1	The average return in the last 1 days.	$ASY_1 = SY_{t-1}$

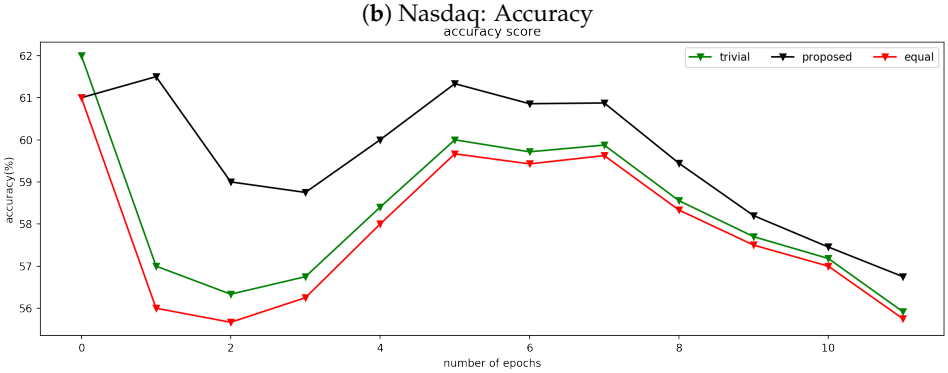
Appendix B. Additional Figures



(**a**) S&P 500: Accuracy



(**b**) Nasdaq: Accuracy



(**c**) Bitcoin: Accuracy

Figure A1. Accuracy of S&P500, Nasdaq, and Bitcoin.

References

1. Ju, C.; Bibaut, A.; van der Laan, M. The relative performance of ensemble methods with deep convolutional neural networks for image classification. *J. Appl. Stat.* **2018**, *45*, 2800–2818. [CrossRef]
2. Cruz, R.M.; Sabourin, R.; Cavalcanti, G.D. Dynamic classifier selection: Recent advances and perspectives. *Inf. Fusion* **2018**, *41*, 195–216. [CrossRef]
3. Krawczyk, B.; Cano, A. Online ensemble learning with abstaining classifiers for drifting and noisy data streams. *Appl. Soft Comput.* **2018**, *68*, 677–692. [CrossRef]
4. Krawczyk, B.; Minku, L.L.; Gama, J.; Stefanowski, J.; Woźniak, M. Ensemble learning for data stream analysis: A survey. *Inf. Fusion* **2017**, *37*, 132–156. [CrossRef]
5. Hinton, G.; Vinyals, O.; Dean, J. Distilling the knowledge in a neural network. *arXiv* **2015**, arXiv:1503.02531.
6. Park, S.; Hah, J.; Lee, J. Inductive ensemble clustering using kernel support matching. *Electron. Lett.* **2017**, *53*, 1625–1626. [CrossRef]

7. Barbosa, J.; Torgo, L. Online ensembles for financial trading. In *Practical Data Mining: Applications, Experiences and Challenges*; ECML/PKDD: Berlin, Germay, 2006; p. 29.
8. Wolpert, D.H.; Macready, W.G. No free lunch theorems for optimization. *IEEE Trans. Evol. Comput.* **1997**, *1*, 67–82. [CrossRef]
9. Kolter, J.Z.; Maloof, M.A. Dynamic weighted majority: An ensemble method for drifting concepts. *J. Mach. Learn. Res.* **2007**, *8*, 2755–2790.
10. Park, S.; Lee, J.; Son, Y. Predicting Market Impact Costs Using Nonparametric Machine Learning Models. *PLoS ONE* **2016**, *11*, e0150243. [CrossRef]
11. Mosca, A.; Magoulas, G.D. Distillation of deep learning ensembles as a regularisation method. In *Advances in Hybridization of Intelligent Methods*; Springer: Berlin/Heidelberg, Germany, 2018; pp. 97–118.
12. Choromanska, A.; Henaff, M.; Mathieu, M.; Arous, G.B.; LeCun, Y. *The Loss Surfaces of Multilayer Networks*; Artificial Intelligence and Statistics: New York, NY, USA, 2015; pp. 192–204.
13. Krizhevsky, A.; Sutskever, I.; Hinton, G.E. Imagenet classification with deep convolutional neural networks. In Proceedings of the Advances in Neural Information Processing Systems, Lake Tahoe, NV, USA, 3–6 December 2012; pp. 1097–1105.
14. Simonyan, K.; Zisserman, A. Very deep convolutional networks for large-scale image recognition. *arXiv* **2014**, arXiv:1409.1556.
15. He, K.; Zhang, X.; Ren, S.; Sun, J. Deep residual learning for image recognition. In Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition, Las Vegas, NV, USA, 27–30 June 2016; pp. 770–778.
16. Fawaz, H.I.; Forestier, G.; Weber, J.; Idoumghar, L.; Muller, P. Deep Neural Network Ensembles for Time Series Classification. *arXiv* **2019**, arXiv:1903.06602.
17. Fan, Z.; Song, X.; Xia, T.; Jiang, R.; Shibasaki, R.; Sakuramachi, R. Online Deep Ensemble Learning for Predicting Citywide Human Mobility. *Proc. ACM Interact. Mob. Wearable Ubiquit. Technol.* **2018**, *2*, 105. [CrossRef]
18. Graves, A.; Mohamed, A.R.; Hinton, G. Speech recognition with deep recurrent neural networks. In Proceedings of the 2013 IEEE International Conference on Acoustics, Speech and Signal Processing, Vancouver, BC, Canada, 26–31 May 2013; pp. 6645–6649.
19. Goodfellow, I.; Bengio, Y.; Courville, A. *Deep Learning*; MIT Press: Cambridge, MA, USA, 2016.
20. Kalchbrenner, N.; Grefenstette, E.; Blunsom, P. A convolutional neural network for modelling sentences. *arXiv* **2014**, arXiv:1404.2188.
21. Pineda, F.J. Generalization of back-propagation to recurrent neural networks. *Phys. Rev. Lett.* **1987**, *59*, 2229. [CrossRef]
22. Hochreiter, S.; Schmidhuber, J. Long short-term memory. *Neural Comput.* **1997**, *9*, 1735–1780. [CrossRef] [PubMed]
23. Gers, F.A.; Schmidhuber, J.; Cummins, F. Learning to forget: Continual prediction with LSTM. In Proceedings of the 9th International Conference on Artificial Neural Networks IET, Edinburgh, UK, 7–10 September 1999.
24. Mikolov, T.; Karafiát, M.; Burget, L.; Černockỳ, J.; Khudanpur, S. Recurrent neural network based language model. In Proceedings of the Eleventh Annual Conference of the International Speech Communication Association, Chiba, Japan, 26–30 September 2010.
25. Benkeser, D.; Ju, C.; Lendle, S.; van der Laan, M. Online cross-validation-based ensemble learning. *Stat. Med.* **2018**, *37*, 249–260. [CrossRef] [PubMed]
26. Van der Laan, M.J.; Polley, E.C.; Hubbard, A.E. Super learner. *Stat. Appl. Genet. Mol. Biol.* **2007**, *6*, 25. [CrossRef]
27. Mohri, M.; Rostamizadeh, A.; Talwalkar, A. *Foundations of Machine Learning*; MIT Press: Cambridge, MA, USA, 2018.
28. Breiman, L. Stacked regressions. *Mach. Learn.* **1996**, *24*, 49–64. [CrossRef]
29. Bubeck, S. *Introduction to Online Optimization*; Lecture Notes; Princeton University: Princeton, NJ, USA, 2011; pp. 1–86.
30. Schapire, R.E.; Freund, Y. *Boosting: Foundations and Algorithms*; MIT Press: Cambridge, MA, USA, 2012.
31. Kingma, D.P.; Ba, J. Adam: A method for stochastic optimization. *arXiv* **2014**, arXiv:1412.6980.
32. Pedregosa, F.; Varoquaux, G.; Gramfort, A.; Michel, V.; Thirion, B.; Grisel, O.; Blondel, M.; Prettenhofer, P.; Weiss, R.; Dubourg, V.; et al. Scikit-learn: Machine Learning in Python. *J. Mach. Learn. Res.* **2011**, *12*, 2825–2830.

33. Chollet, F. Keras. 2015. Available online: https://keras.io (accessed on 22 February 2019).
34. Son, Y.; Noh, D.j.; Lee, J. Forecasting trends of high-frequency KOSPI200 index data using learning classifiers. *Expert Syst. Appl.* **2012**, *39*, 11607–11615. [CrossRef]
35. Qiu, M.; Song, Y. Predicting the direction of stock market index movement using an optimized artificial neural network model. *PLoS ONE* **2016**, *11*, e0155133. [CrossRef] [PubMed]
36. Kim, S.; Jo, K.; Ji, P. The analysis on the causes of corporate bankruptcy with the bankruptcy prediction model. *Mark. Econ. Res.* **2011**, *40*, 85–106.
37. Chawla, N.V.; Bowyer, K.W.; Hall, L.O.; Kegelmeyer, W.P. SMOTE: Synthetic minority over-sampling technique. *J. Artif. Intell. Res.* **2002**, *16*, 321–357. [CrossRef]