The Journal of Supercomputing (2023) 79:4622–4659
https://doi.org/10.1007/s11227-022-04834-4

A Bayesian-based classification framework for financial time series trend prediction

Arsalan Dezhkam[1] · Mohammad Taghi Manzuri[1] · Ahmad Aghapour[1] · Afshin Karimi[1] · Ali Rabiee[1] · Shervin Manzuri Shalmani[2]

Accepted: 13 September 2022 / Published online: 29 September 2022

Abstract
Financial time series have been extensively studied within the past decades; however, the advent of machine learning and deep neural networks opened new horizons to apply supercomputing techniques to extract more insights from the underlying patterns of price data. This paper presents a tri-state labeling approach to classify the underlying patterns in price data into up, down and no-action classes. The introduction of a no-action state in our novel approach alleviates the burden of denoising the dataset as a preprocessing task. The performance of our labeling algorithm is experimented with using machine learning and deep learning models. The framework is augmented by applying the Bayesian optimization technique for the selection of the best tuning values of the hyperparameters. The price trend prediction module generates the required trading signals. The results show that the average annualized Sharpe ratio as the trading performance metric is about 2.823, indicating the framework produces excellent cumulative returns.

Keywords Trend prediction · Machine learning · Deep learning · Financial time series · Feature engineering · Classification

1 Introduction

Although the outbreak of the COVID-19 pandemic triggered a sharp decline in stock prices across the financial market, a closer look at the recent figures on the market indices and stock prices illustrates that the trends are showing a pattern of great acceleration. Financial markets' data form in time series and have been studied by researchers within the past decades, though the main objective of these studies is to find more insight into the underlying market trends. The more insight extracted

✉ Mohammad Taghi Manzuri
manzuri@sharif.edu

Extended author information available on the last page of the article

from the market behavior; the better asset pricing is likely to be achieved. This is precisely the most significant aspect of portfolio formation part of the investment process. However, according to the EMH,[1] it is impossible to forecast the prices for future time intervals because the information propagation across the markets rapidly results in updating prices. On the other hand, many studies have been conducted proving that financial markets are indeed a combination of efficient and non-efficient markets; therefore, the stock prices are, to some extent, predictable [31].

Prediction is the process of finding the next plausible outcome based on past experiences and observations. Hence, feeding the most relevant observations to the prediction model is a crucial task to improve the accuracy of the desired output. This process is called feature engineering. When it comes to financial time series, substantial considerations should be noted. In the case of image, text, and speech observations, the input signal has almost all the required information for modeling the prediction process. On the other hand, asset pricing is a complex problem influenced by multiple endogenous and exogenous factors including but not limited to systematic risk, market behavior, interdependence between markets, macroeconomic variables, firm-specific information, investors' sentiment, and news. Authors [4] proposed a comprehensive taxonomy of input features prevalent among financial market researchers. According to their literature review, the authors have shown that the technical indicators have higher prediction power, while the informative signals from social media could boost the models' performance. Hence, the feature selection and engineering process is a stone step toward building an overarching portfolio formation and optimization model.

Researchers have tackled the feature extraction and engineering process using techniques from the time–frequency domain, statistical methodologies, traditional machine learning approaches, and recently deep learning frameworks. For example, the autoregressive integrated moving average (ARIMA) has extensively been used by researchers in financial time series analysis. The ARIMA model is a linear nonstationary model based on the autoregressive moving average (ARMA) model, including a new difference operator to convert nonstationary series to stationary and take the volatility clustering into account [19, 47]. Both ARMA and ARIMA belong to the univariate class of statistical analysis approaches since the only input variable is time series. There are other statistical methods in the same category, such as the generalized autoregressive conditional heteroscedastic (GARCH), and the Smooth Transition Autoregressive (STAR). Researchers in [47] also mentioned the second class of multivariate statistical methodologies, including linear discriminant analysis (LDA), quadratic discriminant analysis (QDA), linear regression (LR), and support vector machines (SVM).

Among time–frequency techniques, discrete wavelet transform (DWT) has been broadly exploited for feature extraction from financial time series. Authors [45] applied wavelet decomposition to the crude oil time series, turning the time series into different forecasting horizons. Applying the DWT to average monthly crude oil prices, they framed their procedure to compartment the whole signal into low- and high-frequency parts. The coarse scales follow the main trends, while the finer

[1] Efficient market hypothesis.

scales' seasonal fluctuations, singular events, and noise appear. Based on the power of wavelet transform in extracting features from various types of data, researchers have always shown specific interest in applying DWT as a preprocessing stage on the financial time series combining it with other frameworks like quantile regression, neural networks, and other applicable methods [1, 10, 12, 20, 23, 29, 42, 43].

Dimensionality reduction for extracting the abstract and high-level features to feed the subsequent modules of the prediction models has been studied in many research works such as [39, 46]. In [46], Zhang et al. applied principal component analysis (PCA) to perform dimensionality reduction and extract the abstract and high-level features to feed the next module of their framework, an LSTM predicting the next trading day's close price. They took the first four principal components of the cumulative contribution rate of the Shanghai Composite Index as the training sample data fed into LSTM. The authors [7] designed a framework to extract features from 24 randomly selected stocks in the SSE 50 index (Shanghai Stock Exchange), using a hybrid method based on the XGBoost and IFA. The generated features are then used in a mean–variance model for portfolio formation.

Around the second decade of the twenty-first century, the winter of neural network applications across the science and technology realm turned to spring. Accordingly, deep learning models found their way into the financial market analysis to find better solutions for complex problems such as asset pricing, stock price prediction, contagion between financial markets, spillovers, and other problems. A noteworthy application of deep learning models is feature extraction and engineering due to their multi-layer cascading non-linear units, enabling them to capture non-linear dependencies and underlying trends in data. Although most of the early works in the context of financial market analysis are based on long short-term memory, LSTM, there is also a rise in applying other architectures like RL units, Q-learning, ensemble learning, transformer networks, and recently generative adversarial networks (GAN) [5, 13, 22, 23]. The authors [25] investigated the contribution of additional information from the US stock market to South Korea's stock prediction. They exploited a multimodal deep learning framework to capture the cross-modal correlation at different levels and showed that deep multimodal networks can leverage the complementarity of stock data and provide more robust predictions. The authors [27] proposed a two-phase solution for the structural break problem in stock markets using deep reinforcement learning and continuous wavelet CNN. To estimate the occurrence probability of a structural break, within the first phase, they combined the time-domain and frequency-domain extracted by LSTM and continuous wavelet CNN, respectively, and after that, the pairs trading strategy in the next phase is optimized using deep Q-learning. Authors [40] applied a preprocessing stage based on the genetic algorithm, GA, on a train and test dataset and then trained a back propagation neural network to predict the closing price of the Shanghai and Shenzhen 300 index for the next 100 trading days.

To enable neural networks to receive the input vector sequentially, recurrent neural networks (RNN) emerged from the traditional feedforward networks. However, these models have severe problems dealing with long input sequences since they can only handle a few steps back. Hence, a developed variation of RNNs named LSTM was introduced to tackle these problems by adding three gates to the original RNN

architecture: (1) a forget gate to control what information requires to be thrown away from the LSTM memory; (2) an input gate to indicate if new information will be added into the memory, and (3) an output gate controlling the output state. To introduce a threshold-based portfolio [24], the authors built three architectures, S-RNN, LSTM, and GRU, to forecast 1-month-ahead stock returns and then used the last business day OHLCV of each month for building portfolios. Tian et al. [36] proposed a hybrid deep learning model based on multilayer bidirectional LSTM networks to solve the stock price prediction problem. They first analyzed the attributes of 10 different stocks using the Pearson correlation coefficient and then applied the LSTM model to forecast the retained attributes after the analysis.

There have been various attempts to tailor the structure of deep learning networks to the observations' characteristics across different contexts. For example, the authors [9] proposed a model to place another attention mechanism over the document-level attention. The so-called attention-over-attention reader model was exploited to provide a solution to the cloze-style reading comprehension task. The authors [14] crafted a CNN-bLSTM deep learning model for improving the performance of conversational speech recognition tasks. Although increasing the number of layers in multi-layer deep models results in the enhanced learning ability of the network, it turns the model to face the problems such as exploding and vanishing gradients. To tackle the incurred problems, researchers proposed a handful of techniques such as dropout, batch normalization, and residual [2, 15, 16, 18, 35]. There are attempts to extend the sentiment analysis techniques and apply the results to price prediction models to enhance the performance of the task. In a recent study [38], the authors augmented a Bidirectional Encoder Representations from Transformers, BERT, with CNN structure to capture important local information in the financial texts. Inspired by the word vectorization technique in natural language processing, the authors [30] introduced stock vectors and proposed two LSTM architectures for dimension reduction and price prediction, one with an embedded layer and the other based on an automatic encoder. Their experimentation for Shanghai A-shares composite index showed that the deep LSTM with the embedded layer performs 0.3% better in terms of the accuracy metric.

In recent decades, much research has been done based on price prediction as a regression task. However, researchers have shown that trend prediction as a classification task can dramatically improve machine learning and deep learning model predictions [32, 37, 44]. The task of labeling financial times series has a significant impact on the prediction model's performance, though the problem has not been widely studied in the literature. Recently, Wu et al. [41] proposed a price data labeling method to extract the continuous trend features of financial time series data and group them into two upward and downward classes. However, sometimes the market has unpredictable fluctuations; in this situation, investors risk losing their money. Thus, the labeling algorithm should have an extra state that shows these unpredictable and risky situations to prevent investors from investing their money in that period. Hence, proposing a price data labeling algorithm to help produce a more informative input feature vector for the market trend prediction task is of great significance. Accordingly, in this study, we introduce a novel tri-state labeling algorithm that significantly improves the quality of predictions by introducing three

states that show upward and downward trends, besides the risky situation with the no-action state.

The rest of the paper is organized as follows: In Sect. 2, we cover the methodology of our research work. It contains eight sub-sections starting from Sect. 2.1, which is the complete description of our proposed tri-state labeling algorithm. Since our experimentation will be conducted on financial time series, in Sect. 2.2, we have thoroughly covered the reasons for using embargoed purging cross-validation instead of the traditional K-fold CV. Then, we included the required background on the machine learning and deep networks we aim to use as our predictive machines. We first explain support vector machines in Sect. 2.3 and then continue the ML methods with XGBoost in Sect. 2.4. The basics of LSTM and GRU are also discussed in Sects. 2.5 and 2.6. In Sect. 2.7, the reader will be refreshed with our approach to the performance evaluation of the classification task. The hyperparameters' value tuning using Bayesian optimization is discussed in Sect. 2.8. Section 2.9 provides readers with all our steps to design and evaluate our trading system. The reader can find all information about our experimentation and the associated results and discuss the findings within Sect. 3. Finally, Sect. 4 concludes the research findings and the further possibilities for future research works.

2 Methodology

2.1 Proposed labeling algorithm

The first part of our proposed framework is a labeling algorithm to extract continuous upward and downward trends from daily close price time series. Our input is the close price time series denoted by X, where x_t is close price at time t. The algorithm finds x_t such that:

$$\left| x_t - x_{t_0} \right| \geq \tau x_{t_0} \tag{1}$$

where x_{t_0} is the close price at the time t_0 and τ denotes a threshold value which is a hyperparameter. According to Eq. (1) if the condition is satisfied then the trend is labeled as upward, otherwise, the direction of the changes is downward. Once the overall direction of the price changes is found, in the second phase of the algorithm, the labeling algorithm will deal with directional changes as follows: Suppose that the labeling algorithm reads x_{t_j}, the price at t_j, and the direction at time t_i has been labeled as upward. However, the algorithm should decide the exact time of changing direction while keeping the upward trend until the next prices are still at higher levels. Therefore, the following three cases are determined:

Case 1

$$x_{t_j} > x_{t_i} \tag{2}$$

In this case, we still label the trend with $+1$ as upward since the coming price at the time t_j is at a higher level compared with the price at the time t_i.

Case 2

$$t_j - t_i > w \tag{3}$$

where w is the window size for the period with no price fluctuations with much larger or smaller values than x_{t_i}, the last updated price shows an upward trend. This means the upward trend has ended and we now can label all the coming time instances with 0, as a "no-action" trend. It is worth mentioning that w is also a hyperparameter that needs to be determined.

Case 3

$$x_{t_i} - x_{t_j} \geq \tau x_{t_i} \tag{4}$$

This means that the upward trend ends and we now should change the direction to downward. This means while we label all instances with $+1$, the state space will be ready to follow the downward direction.

On the other hand, if the state space shows the no-action, label (denoted by 0), the algorithm is adjusted to handle the corresponding three cases:

Case 1

$$x_{t_j} \geq x_{t_i} + \tau x_{t_i} \tag{5}$$

In this situation, the trend shows the start of an upward direction.

Case 2

$$x_{t_j} \leq x_{t_i} - \tau x_{t_i} \tag{6}$$

This means the current price at the time t_j is lower than x_{t_i}, the price at the last no-action state, with a factor of τ. Hence, the algorithm detects a change from a no-action state to falling prices and starts to label the past time instances with label 0 and prepares the state space for the coming downward trend follow-up.

Case 3

$$t_j > t_{zero} + w \tag{7}$$

Since the price time series is still fluctuating, we continue to update the no-action trend with label 0.

The third and final state for our proposed tri-state labeling algorithm is -1 or the "downward," for which exist three cases: (1) remaining at the current state, (2) changing to an upward direction, or (3) putting the system to the no-action state.

Case 1

$$x_j < x_i \tag{8}$$

where x_i is a previous time instance of a downward trend, hence we only update the current downward trend.

Case 2

$$t_j - t_i > w \tag{9}$$

where t_i is the latest time instance of the price at an upward trend, hence, the prices are still fluctuating which means the bullish trend has ended and we are ready to enter a no-action trend while the labeling process of the past trend with -1 is done.

Case 3

$$x_j \geq x_i + \tau x_i \tag{10}$$

where x_i is the last time instance at which the trend has been labeled with a downward direction. Therefore, the direction is changed to upward. This means we must set the state space for the start of a new upward direction, while the past instances are labeled with -1, as a downward trend.

2.2 Combinatorial purged K-fold cross-validation

For a machine learning algorithm to properly learn the general structure of the data and prevent it from the extreme fidelity to the data, we usually split observations into two training and test sets, where the cross-validation (CV) technique is used to prevent overfitting. K-fold CV is widely used among machine learning researchers among popular CV methods. However, for two reasons, this cross-validation method produces undesired results when applied to financial time series.

First, financial time series do not possess the properties of an independent and identically distributed (IID) process. Finance observations are serially correlated, meaning that the feature at time t is highly correlated with the feature at time $t + 1$. Therefore, the prediction process from overlapping data points results in a label at time $t + 1$ which is derived from overlapping features from time t. A second reason for CV's failure in finance is the multiple testing and selection bias. The solution for the second problem is to purge all overlapping labeled samples from the training and test sets. For the serial correlation problem between financial features, the solution is to embargo those samples in the series that immediately follow another sample in the test set. This purging and embargoing cross-validation technique is known as PURGED K-FOLD CV [28]. As shown in Fig. 1a [28], within one partition of the K-fold cross-validation, two overlapping regions need to be purged to prevent data leakage between training and test sets. As shown in Fig. 1b [28], the embargo process is imposed on training samples directly after a test set to bolster leakage prevention between training and test observations.

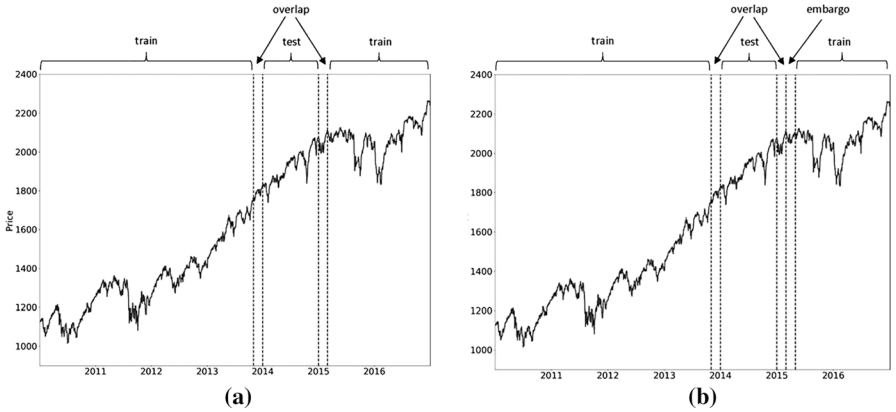

Fig. 1 **a** Purging overlap in the training set; **b** Embargo of post-test train observations [28]

Algorithm 1. PURGED K-FOLD CV

consider a label Y_j that is a function of some observations

$Y_j = f(t);\ t \in [t_{j,0}, t_{j,1}]$

Purging:

For every two consecutive observations

If $t_{j,0} \leq t_{i,0} \leq t_{j,1}$ then

Drop the observation

If $t_{j,0} \leq t_{i,1} \leq t_{j,1}$ then

Drop the observation

If $t_{i,0} \leq t_{j,0} \leq t_{j,1} \leq t_{i,1}$ then

Drop the observation

Embargoing:

Define a hyper-parameter, embargo period h

drop the observations that take place immediately after the test,

$t_{j,1} \leq t_{i,0} \leq t_{j,1} + h$

2.3 Support vector machines (SVM)

SVMs are a set of widely used supervised learning algorithms for classification, regression, and outlier detection tasks through finding the optimal hyperplane using margin maximization. The basic idea behind SVM is to apply a non-linear transformation to map the input vector $\boldsymbol{x}$ into a high-dimensional feature space. Suppose the input feature is $x_{i;i=1..n} \in R^p$, where p is the total number of data patterns, and the

corresponding target is $y_i \in R$. Then, the SVM computes a decision function of the following form:

$$y(x) = w^T \phi(x) + b \tag{11}$$

The objective is to maximize the margin plane parameterized by w and b. Class labels are then assigned by sgn function:

$$label = sgn(y(x)) \tag{12}$$

The parameters w and b are estimated by solving the following minimization problem:

$$\begin{aligned} &\underset{w,b,\zeta}{\text{Minimize}} \frac{1}{2} w^2 + C \sum_{i=1}^{n} \left(\zeta_i + \zeta_i^*\right) \\ &subject\ to \begin{cases} d_i - w\phi\left(\boldsymbol{x}_i\right) - b_i \leq \epsilon + \zeta_i \\ w\phi\left(\boldsymbol{x}_i\right) + b_i - y_i \leq \epsilon + \zeta_i^* \\ \zeta_i, \zeta_i^* \geq 0 \end{cases} \end{aligned} \tag{13}$$

where C is the penalty parameter, and ζ_i, ζ_i^* are the slack variables. The above-mentioned optimization problem is solved by the Lagrangian method:

$$\begin{aligned} &Max_\alpha \sum_{i=1}^{n} d_i\left(\alpha_i - \alpha_i^*\right) - \epsilon \sum_{i=1}^{n} \left(\alpha_i + \alpha_i^*\right) - \frac{1}{2} \sum_{i=1}^{n} \sum_{j=1}^{n} \left(\alpha_i - \alpha_i^*\right)\left(\alpha_j - \alpha_j^*\right) K\left(x_i, x_j\right) \\ &subject\ to \ \sum_{i=1}^{n} \alpha_i = \sum_{i=1}^{n} \alpha_i^*; \quad 0 \leq \alpha_i, \alpha_i^* \leq C, \quad i = 1, 2, \ldots, n \end{aligned} \tag{14}$$

where $K\left(x_i, x_j\right)$ is the kernel function:

$$K\left(x_i, x_j\right) = \phi\left(x_i\right) \odot \phi(x_j) \tag{15}$$

where the $\odot$ is the inner product operator. The solution for α_i determines the parameters w and b for the optimal hyperplane.

2.4 Extreme gradient boosting (XGBoost)

In the context of machine learning, a weak learner is a classification model that can perform marginally better than random guessing. Authors [33] developed boosting in a successful attempt to answer the question "Can a set of weak learners create a single strong learner?" proposed by [21]. The main idea behind most of the boosting algorithms is to iteratively apply a weak learner to training data and assign more weights to misclassified observations to find a new decision stump for them. Finally, all learned models are aggregated to form a strong learner able to classify all training samples correctly. Therefore, a decision tree ensemble model with K additive functions is used to predict the target.

$$\hat{y}_i = \phi(\boldsymbol{x_i}) = \sum_{k=1}^{K} f_k(\boldsymbol{x}_i), f_k \in F \tag{16}$$

In Eq. (16), $\boldsymbol{x}$ is the m-dimensional input feature vector, y is the one-dimensional target vector forming the n cardinality sample space $D = \{(\boldsymbol{x}_i, y_i); |D| = n, \boldsymbol{x}_i \in \mathbb{R}^m, y_i \in \mathbb{R}\}$. The space of classification and regression trees (CART) with T leaves in each tree is also indicated by $F = \{f(\boldsymbol{x}) = w_{q(\boldsymbol{x})}; q : \mathbb{R}^m \to T, w \in \mathbb{R}^T\}$ where f_k represents an independent tree structure q whose leaf weights are w. To classify the observations, the decision rules in the trees are applied to calculate the predicted target by summing up all w_i, the weights in the corresponding leaves. Equation (17) shows the objective function used to learn the set of functions used in the model.

$$l(\phi) = \sum_i l(\hat{y}_i, y_i) + \sum_k \Omega(f_k) \tag{17}$$

where $\Omega(f) = \gamma T + \frac{1}{2}\lambda \|w\|^2$.

As it can be seen from Eq. (17), the model is trained in an additive manner instead of using traditional optimization methods in the Euclidean space. Hence, while the adaptive boosting techniques try to weigh misclassified samples more, in gradient boosting, base learners are generated sequentially so that the current model is always more effective than the previous one by ameliorating a loss function. Therefore, the objective function to be optimized is modified to include greedily adding f_t.

$$\begin{aligned} l^{(t)} &= \sum_{i=1}^{n} l\left(y_i, \hat{y}_i^{(t-1)} + f_t(\boldsymbol{x}_i)\right) + \Omega(f_t) \\ \Omega(f) &= \gamma T + \frac{1}{2}\lambda \|w\|^2 \end{aligned} \tag{18}$$

XGBoost [6] is a highly enhanced version of gradient boosting, and it mainly aims at increasing computation speed and efficiency since the gradient boosting algorithm analyzes the datasets sequentially resulting in a very low rate performance. Furthermore, XGBoost supports parallelization by creating decision trees in a parallel way like the random forest. It also exploits distributed computing methods to evaluate large and complex models and uses Out-of-Core computation to analyze large and varied datasets. Using cache optimization is another technique used in XGBoost to achieve a higher level of optimal resource utilization. Therefore, XGBoost, as a simple to utilize and interpretable prediction model, has been widely used and has outperformed most modern and state-of-the-art deep learning methods for classification and clustering of tabular data [34].

2.5 Long short-term memory (LSTM)

Learning in the human brain is an accumulative process that prevents it from restarting thinking and learning from scratch every second. This is in contrast to what happens in traditional neural networks, as they cannot consider the previous events to infer

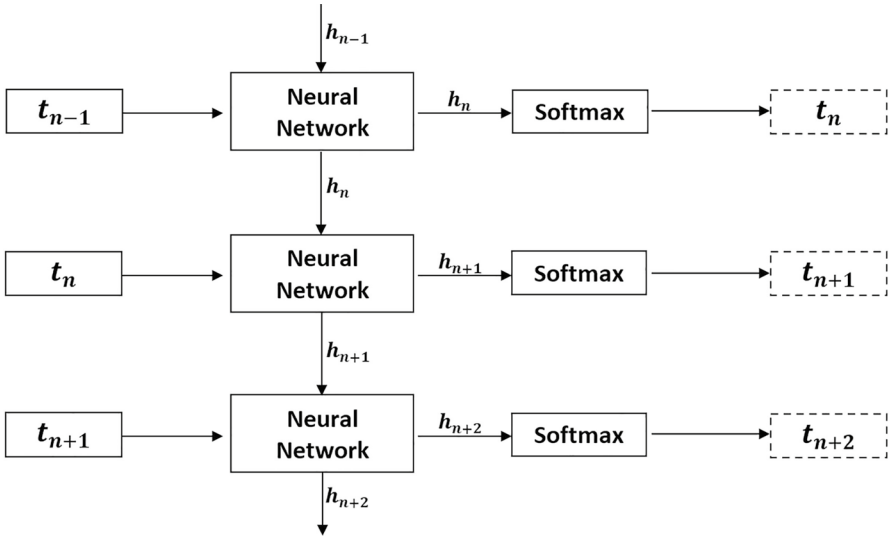

Fig. 2 RNN architecture

about the later ones. To overcome this shortcoming, recurrent neural networks (RNNs) have been widely adopted concerned with applications in time series and sequential data such as price prediction, speech recognition, and image recognition. As it is shown in Fig. 2, the goal of the RNN is to have a model which can take the n current observations in the associated vector t_n, send it into a neural network and using the knowledge from the previous stage as the hidden vector h_n and predict the next target t_{n+1}.

Vanishing gradient, exploding gradient, long-term dependency, and unidirectionality are the major drawback of the RNN model. The long short-term memory, LSTM, model is one way to solve these problems. LSTM has introduced a memory unit called the cell into the network. Different researchers adopted various architectures for the LSTM model, but the original design incorporates a forget gate, an input gate, and an output gate with a peephole connection. The mathematical equations according to the connections and gates in the LSTM architecture, Fig. 3, are expressed as follows:

$$\begin{aligned}
F_t &= \sigma\left(W_{HF}H_{t-1} + W_{XF}X_t + P_F \odot C_{t-1} + b_F\right),\\
I_t &= \sigma\left(W_{HI}H_{t-1} + W_{XI}X_t + P_I \odot C_{t-1} + b_I\right),\\
\tilde{C}_t &= \tanh\left(W_{HC}H_{t-1} + W_{XC}x_t + b_C\right),\\
C_t &= F_t \odot C_{t-1} + I_t \odot \tilde{C}_t,\\
O_t &= \sigma\left(W_{HO}H_{t-1} + W_{XO}X_t + P_O \odot C_t + b_O\right),\\
H_t &= O_t \odot \tanh\left(C_t\right)\\
Y_t^{LSTM} &= \sigma\left(W_{HY}H_t + b_Y\right)
\end{aligned} \tag{19}$$

where F_t, I_t, and O_t are the forget gate, input gate, and output gate at time t, respectively. W s and b s in Eq. (19) represent the corresponding weight matrices and

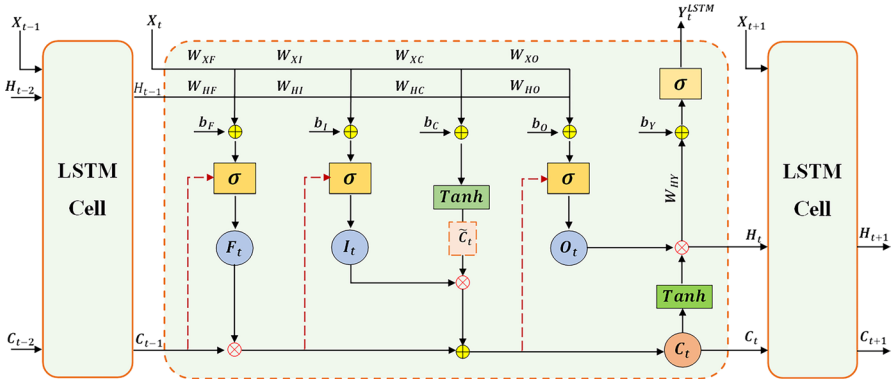

Fig. 3 LSTM architecture

bias terms in each equation. $\sigma(\bullet)$ indicates a sigmoid activation function, Tanh$(\bullet)$ is a hyperbolic tangent function, and $\odot$ is an element-wise multiplication operator. The forget gate decides what information from the past cell state C_{t-1} is propagated to the updating process of the cell state at time t. If $F_t = 1$, it keeps the information received, while a value of 0 for F_t means the information is discarded. P_F, P_I, and P_O are the peephole weights for the respective forget, input, and output gates. The peephole connections are a mechanism to enable the LSTM cell for inspecting its current internal states resulting in learning unsupervised stable and precise timing algorithms [11]. Since LSTM is a special variant of the RNN model, the same process for RNN weight updates and hyperparameter optimization methods can be exploited within LSTM networks [3].

2.6 Gated recurrent unit (GRU)

Although the LSTM cell has an outstanding learning capacity in comparison to the traditional RNN, its computational complexity is higher than RNN because of the extra parameters of the model. To reduce the number of parameters, the authors [8] introduced the GRU cell by integrating the forget and input gates into an update gate. Having only two gates, reset and update, the GRU removes a gating signal, and the associated parameters compared with the LSTM cell. Since GRU has fewer parameters to be learned it has fewer tensor operations which in turn results in slightly reduced computational time. Equation (20) represents the mathematical expressions of the GRU cell, and the corresponding architecture is visualized in Fig. 4.

$$\begin{aligned} R_t &= \sigma\left(W_{HR}H_{t-1} + W_{XR}X_t + b_R\right), \\ Z_t &= \sigma\left(W_{HZ}H_{t-1} + W_{XZ}X_t + b_Z\right), \\ \tilde{H}_t &= \tanh\left(W_{H\tilde{H}}\left(R_t \odot H_{t-1}\right) + W_{X\tilde{H}}X_t + b_Z\right), \\ H_t &= (1 - Z) \odot H_{t-1} + Z_t \odot \tilde{H}_t \end{aligned} \tag{20}$$

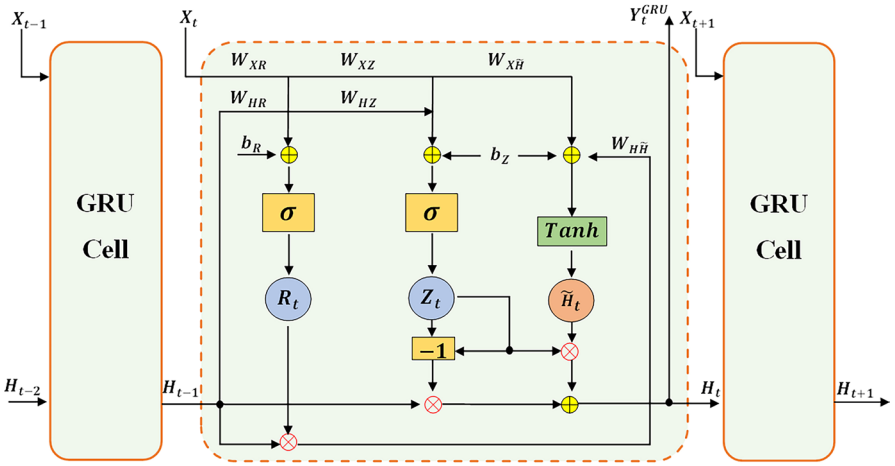

Fig. 4 GRU cell. Forget and input gates in LSTM are now integrated into the update gate in the GRU model

2.7 Classification task metrics

The AUC, or area under the receiver operating characteristic curve, is the most versatile and common evaluation metric used to judge the quality of a binary classification model. It is simply the probability that a randomly chosen positive data point will have a higher rank than a randomly chosen negative data point for the learning problem. So, a higher AUC means a more sensitive, better-performing model. When dealing with multi-class classification problems, it is common to use the accuracy score and to look at the overall confusion matrix to evaluate the quality of a model. Accuracy is the most straightforward to evaluate the overall performance of the classification task. Since financial time series are almost equally weighted across all classes, so all the individual dataset elements have approximately the same weight and contribute equally to the accuracy value. Therefore, the higher accuracy, the higher the probability that the model prediction is correct. Assigning up, down, and no-action labels to the market trends is a multi-class classification task in which the upward trends are positive, downward trends are negative, and non-essential fluctuations are shown with 0. Therefore, the performance of the labeling task can be assessed by computing the confusion matrix of the corresponding three classes. For each class, the classification task should be assessed for the performance of recognizing the class label correctly (true positives, TP), the number of correctly classified corresponding labels (true negatives, TN), the number of observations that were either incorrectly assigned to the class (false positives, FP) or those that incorrectly were not assigned to the class (false negatives, FN). Hence, the performance of the models for each class can be evaluated using precision, recall, and F1-Score metrics, and the overall performance of the classification task is assessed based on accuracy. Precision is defined as the ratio between the TP area and the area in the ellipse in Fig. 5. The recall is computed by finding the ratio between the TP area and the area

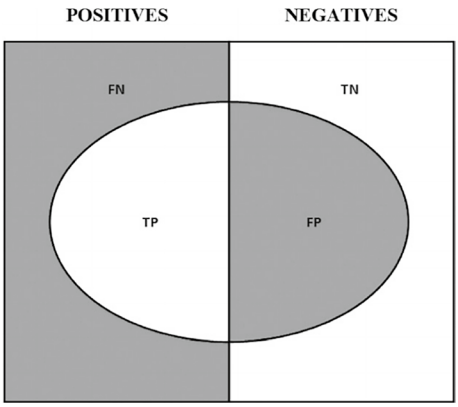

Fig. 5 Confusion matrix. *Source*: [28]

in the left rectangle. Accuracy is the sum of the TP and TN areas divided by the whole area (square). In the context of the stock market prediction, we would like to predict and capture as many as up-trends to maximize the profit, however, we want to be very sure about our prediction. It means we need to maximize both precision and recall. On the other hand, decreasing the FP area comes at a cost of increasing the FN area, because higher precision typically means fewer calls, hence the lower recall. A trade-off between recall and precision is obtained by taking the harmonic mean of them, which is known as the F1-score.

$$\begin{aligned} Accuracy &= \frac{TP + TN}{TP + TN + FP + FN} \\ Precision &= \frac{TP}{TP + FP} \\ Recall &= \frac{TP}{TP + FN} \\ F1 &= \frac{2}{\frac{1}{Precision} + \frac{1}{Recall}} \end{aligned} \tag{21}$$

2.8 Hyperparameter optimization

A common practice for a learning machine to produce the desired output is to become optimized from the perspective of the parameters that control the learning process. These controlling measures are called hyperparameters. Hyperparameter optimization is a widespread technique that is done by maximizing or minimizing an objective function with a performance or loss metric to find a tuple of parameters that result in an optimal model. There are many techniques for this task including grid search, random search, gradient-based optimization, and so on. When it comes to problems with expensive-to-evaluate functions, Bayesian optimization is the first choice. It has the advantage of no predefined

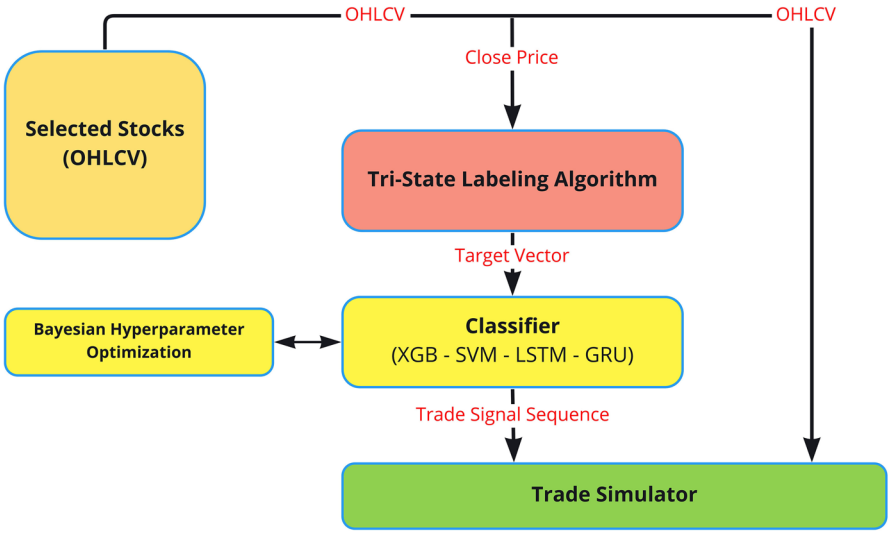

Fig. 6 Trend prediction and trading framework pipeline

assumption for the functional form of problems. If $x \in X$, and the problem under the study is $f(x)$ then the Bayesian hyperparameter optimization yields the result by Eq. (22):

$$x^* = \arg_{direction} f(x) \tag{22}$$

where $direction \in \{Maximize, Minimize\}$.

In Eq. (22), x belongs to the hyperparameter search space X, and direction shows either minimization when the goal is to minimize the loss function or maximizing toward higher values of some performance metrics such as accuracy.

Figure 6 illustrates the whole work of the prediction machine and trading simulation. The input vector of the machine includes a vector of various stock indices which is processed by the proposed labeling algorithm to indicate the ups and downs of the market. The labeled stock prices vector is fed into four independent classification models discussed earlier. The models are trained separately and are used for predicting the next close price trend from the unseen test data. These models are optimized by Bayesian optimization in a probabilistic search space. To run Bayesian optimization, we assume that for our classifier model function, $f(x)$, the performance of the model for a specific combination of hyperparameters is known as prior information. Then, we form the posterior probability function and exploit it to enhance the performance metric based on finding a better estimation of a new combination of hyperparameters. This procedure continues until it stops with no more improvement on the performance metric and the best tuning parameters for maximum performance are reported in the last stage, where the parameters are saved to be used for model preparation.

Date	1/28/2020	1/29/2020	1/30/2020	1/31/2020	2/3/2020	2/4/2020	2/5/2020	···	4/15/2021	4/16/2021	4/19/2021	4/20/2021	···
Price	50.529999	47.509998	48.779999	47	48.02	49.45	49.84	···	83.010002	82.15	81.11	79.27	···
Label	1	-1	-1	-1	-1	1	1	···	1	0	0	-1	···
Signal	Buy	Sell				Buy		···	Buy			Sell	···
# Shares	197902.24	0	0	0	0	190138.2	190138.2	···	111086.64	111086.6	111086.6	0	···
Capital	10000000	9402335.037	9402335.037	9402335.037	9402335	9402335	9476489	···	9221302.5	9125768	9010238	8805838	···

Fig. 7 A snapshot of the trading system using predicted labels. Buy and sell positions are taken upon receiving the appropriate signal. The first '1' indicates a buy position and the next '−1' is the sell position. '0' labels are indicating the volatile market

2.9 Trading system

Trading strategies in financial markets are vital to profits, avoiding emotions and behavioral finance biases. Therefore, traders should decide when to sell an asset or security to be less likely to succumb to the disposition effect, which causes them to hold on to stocks that have lost value and sell those that rise in value. Hence, traders need to carefully look for trade signals, buy an asset and then sell it at an opportune moment. Trade signals can be composed of complex indicators, including but not limited to technical signals, fundamental analysis, sentiment measures, macroeconomic indicators, and even inputs from other trading signal systems. However, it is recommended to provide traders with a simple trading module using only a handful of inputs. The advantage of the proposed architecture in this paper is its ability to extract upward and downward trends from the market based on its behavior and turn the buy or sell signal on. To evaluate the model's performance using the predicted labels for test datasets, we conducted experiments using markets studied in this research. An initial capital has been assumed to be available at the time t_0 to be used for trading. As soon as the system receives the buy signal at the time t_i, the whole initial capital is used to purchase as many shares as possible from the target asset. The system puts aside the remaining capital in the balance and waits for the next coming sell signal. At time t_j, the system detects a downward market and issues the sell signal. Therefore, the whole shares are sold at the time t_j and the total amount of the trade is summed into the available balance forming the new capital. This process is continued until the end of the study period. A sample run of this trading mechanism is shown in Fig. 7.

The primary goal of every trading system is to maximize the returns while considering risk and time spent with capital invested in the market. To compare the performance of the system using predicted labeled series from ML/DL methods used in this study, we use the metrics for the rate of return per day (RoR/day), and risk-return ratio (RRR). The RoR/day indicates the daily profit obtained in the market. Hence, the RoR_t for some time $t \in [i, j]$ is defined as follows:

$$RoR_t = \frac{Capital_j - Capital_i}{Capital_i} \times 100 \tag{23}$$

And the daily RoR is computed by taking into account the whole days in between.

$$RoR/day = \frac{RoR}{NumberofDaysinMarket} \tag{24}$$

To calculate the *RRR*, we need to compute the maximum drawdown. The maximum drawdown (*MDD*) is the maximum amount of loss from a peak to a trough in a specific period before a new peak is attained, which indicates the downside risk over the period.

$$\begin{aligned} DD_t &= Max_{k\in(i,t)}\left(RoR_k\right) - RoR_t \\ MDD &= Max_{t\in(i,j)}\left(DD_t\right); i < k < j \end{aligned} \tag{25}$$

Having defined *RoR*, and *MDD*, it is now straightforward to calculate *RRR*:

$$RRR = \frac{RoR}{MDD} \tag{26}$$

The third metric is the Sharpe ratio which is used as a measure of the performance of the system in making profitable trades while minimizing the risk. The Sharpe ratio is defined as the ratio of the excess expected return to the standard deviation of the return.

$$Sharpe\ Ratio = \frac{\mu - R^f}{\sigma} \tag{27}$$

where the R^f is the risk-free rate, μ is the mean of the one-period simple-return of an asset between dates $t-1$ and t, and σ is the corresponding standard deviation.

3 Results and discussion

To test the proposed framework, we consider some stock indices from S&P500 since this market is widely used in computational finance literature. We have selected stocks based on their systematic risk compared with the market's risk. Advanced Micro Devices, Inc., AMD, Apple Inc., AAPL, The Clorox Company, CLX, Macy's Inc., M, Seagate Technology Holdings plc, STX, and Walmart Inc., WMT, are the selected stocks for further experimentation using our proposed framework. As can be seen from Table 1, Beta values range from 0.17 to 2.09 indicating less risk for stocks with a Beta smaller than 1. As Beta takes larger values than 1, the corresponding asset is considered to have more risk than the market. Daily close prices for the selected assets are obtained from Yahoo finance. Trained models are back-tested with out-of-sample data for the last two years from January 16, 2020, to November 23, 2021. For computation simplicity, we have assumed that the risk-free rate is zero and the transaction cost is zero throughout our simulations.

Descriptive statistics for the dataset are also listed in Table 1. The large values of Chi-Square and zero for the p value of the Jarque–Bera test confirm that the null hypothesis for all series to be normally distributed is rejected. The skewness values for most indices are negative, indicating these markets are downward most of the time and experience negative returns. The series skewed with high excess kurtosis, indicating the presence of high peaks and heavy tails. As a statistical measure, kurtosis shows the degree of presence of outliers in the underlying distribution. This

Table 1 Descriptive statistics for datasets; the statistics are computed for the log return series

Stock	Min	Max	Mean	Std	Skewness	Kurtosis	Jarque_Bera test	Beta	Observations
CLX	−0.1777	0.1246	0.0005	0.015	−0.3623	11.1595	(40,634.0, 0.0)	0.17	7809
WMT	−0.1074	0.1107	0.0004	0.016	0.1196	5.1631	(8677.52, 0.0)	0.52	7808
M	−0.2244	0.1921	0.0002	0.0274	−0.0893	7.4364	(17,299.2, 0.0)	2.09	7515
AAPL	−0.7312	0.2869	0.0008	0.028	−2.2186	65.8689	(1,416,089.97, 0.0)	1.2	7808
STX	−0.2807	0.2458	0.0006	0.0292	−0.6957	11.4471	(26,417.66, 0.0)	1.06	4779
AMD	−0.4769	0.4206	0.0005	0.039	−0.3329	10.4935	(35,916.69, 0.0)	1.93	7808

The values in the parentheses in the Jarque–Bera column indicate Chi-Square value and p value, respectively

measure plays a crucial role in determining the associated risk in an asset in financial markets. High values of kurtosis imply there are high probabilities of extreme returns. Hence, these assets are risky and the returns experience a lot of outliers.

To train the models, we first need to construct the required input features. As described in Sect. 2, we first pass the adjusted daily close prices into the labeling algorithm to form the input features for the classification models. The result of applying our labeling algorithm to the stock price time series is shown in Fig. 8. As shown in Fig. 8a, c, the labeling algorithm follows the upward and downward trends in the AMD stock price while also annotating a wait signal, the '0' label, whenever the market is slightly fluctuating hence it is hard to determine the exact direction of the trend. The same scenario is plotted in Fig. 8b, d for STX stock.

Combinatorial Embargoed Purging cross-validation process applied to the data set is shown in Fig. 9. The blue and red bars indicate the 8 training and 2 validation folds. As it is discussed in Sect. 2.2, when the validation set places before the training, the extracted overlapping region is doubled. The labels generated by the proposed tri-state labeling algorithm are presented along the Target Label row.

The hyperparameters selected for tuning the models, along with the range of their corresponding possible values, are explained in Table 2. For the recurrent neural networks used in this research, LSTM and GRU, three neurons are in the output layer. This is because the problem under the study is a multi-class classification with three classes. Dropout value, the number of units in the hidden layer, and the learning rate are three hyperparameters that must be optimized for the maximum accuracy score of the classification task. The configuration for LSTM and GRU simulation consists of 20 trials of running networks with 100 epochs. This configuration is repeated 25 times and each time the parameters' values for the best trial are collected. The objective function is set in the maximization direction to optimize the performance metrics, F1-Score and accuracy. Extreme gradient boosting, XGBoost, has a long list of hyperparameters that could be considered for optimization purposes. We have selected some of the most significant parameters of XGBoost, such as the number of boosting stages, n_estimator, the maximum depth of the individual estimators, and the subsample ratio of columns when constructing each estimator, colsample_bytree. L1 and L2 regularization terms on weights in the list of hyperparameters are shown as reg_alpha and reg_lambda, respectively. Support vector

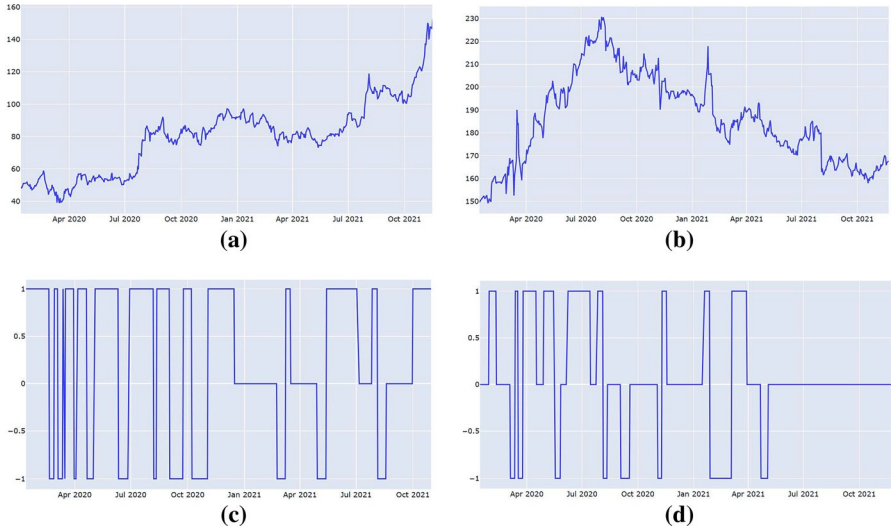

Fig. 8 Automatic labeling of AMD and CLX time series using the proposed algorithm. **a**, **b** Are price time series for AMD and CLX, respectively, while **c**, **d** show their continuous trend labeling. The vertical axis in (**a**, **b**) is in US dollars. In labeling diagrams, up-trends are shown by 1, and the down-trend is -1, while 0 stands for no deterministic trend due to high volatility between two up and down situations. For both time series, the threshold value is set to 0.05

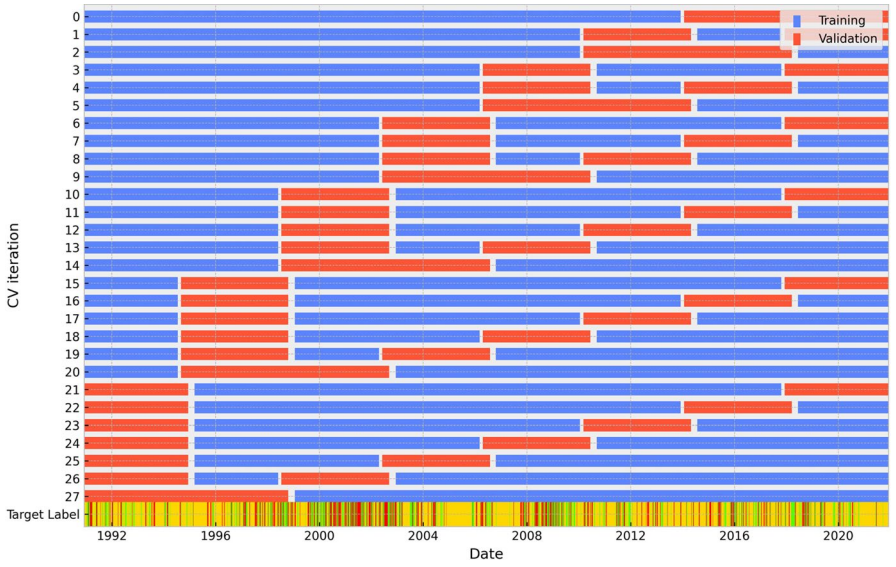

Fig. 9 Combinatorial embargoed purging K-fold CV. The blue and red bars indicate the training and validation sets, respectively. Target label is the labels extracted from price data using proposed framework. Green bars are $+1$, Reds show -1 and fluctuating periods are marked with yellow bars. Price data as input feature vector have been cross-validated with 8 training and 2 validation folds. Resource: research simulations

Table 2 Selected hyperparameters for optimization

Name	Description	Range
LSTM and GRU		
units	Specifies the number of units in each dense layer	24, 32,48, 64
Dropout	Float between 0 and 1. Fraction of the units to drop for the linear transformation of the inputs	0.2, 0.25, 0.3, 0.35, 0.4
learning rate (l.r.)	How quickly the network updates its parameters	[0.001, 0.1]
XGBoost		
objective	Learning objective function	multi:softprob, multi:softmax
n_estimators	The number of trees in our ensemble. Equivalent to the number of boosting rounds. The value must be an integer greater than 0. Default is 100	50–1000
max_depth	The maximum depth per tree. A deeper tree might increase the performance, but also the complexity and chances to overfit. The value must be an integer greater than 0. Default is 6	3–18
sampling_method	Used only by gpu_hist tree method	'uniform,' 'gradient_based'
colsample_bytree	The fraction of columns to be randomly sampled for each tree. It might improve overfitting. The value must be between 0 and 1. Default is 1	0.1–0.99
learning_rate	The step size at each iteration, while the model optimizes toward its objective	0.01–0.2
reg_alpha	L1 regularization on the weights (Lasso Regression). When working with a large number of features, it might improve speed performance. It can be any integer. Default is 0	0.00001–0.01
reg_lambda	L2 regularization on the weights (Ridge Regression). It might help to reduce overfitting. It can be any integer. Default is 1	0.00001–0.01
gamma	A pseudo-regularization parameter (Lagrangian multiplier) that depends on the other parameters. The higher Gamma is, the higher the regularization. It can be any integer. Default is 0	0.1–0.99
SVM		
kernel	Specifies the kernel type to be used in the algorithm. If none is given, 'rbf' will be used	poly, rbf, sigmoid
C	Regularization parameter. The strength of the regularization is inversely proportional to C. Must be strictly positive. The penalty is a squared l2 penalty	[0.01, 1]
gamma	Kernel coefficient for 'rbf,' 'poly,' and 'sigmoid'	1, 2, 3, 4, 5, 6, 7, 8, 9
degree	Degree of the polynomial kernel function ('poly'). Ignored by all other kernels	2, 3, 4

machine classification task is conducted using Python's scikit-learn package. The major examined hyperparameters for SVC are the regularization parameter, C, the kernel type to be used in the algorithm, Kernel, the degree of the polynomial kernel function, degree, and the kernel coefficient, gamma. The results show that the model with the radial basis function (RBF) kernel with the regularization parameter close to 1 and of degree 3 produces the best results. Therefore, we choose the same values to further conduct our simulation for future trading strategy experimentations. To search for the best hyperparameters we apply Bayesian optimization to search for values of hyperparameters maximizing the overall classification accuracy. Tables 3, 4, 5, 6, 7, 8, 9 and 10 represent the results of the hyperparameter optimization procedure along with the corresponding performance metrics values.

Having found the best hyperparameters' values for our problem and the dataset that we study, it is time to perform full experimentation including both the model training and then performing trading simulation using the labels predicted by the model. In Table 11, we have shown the report of classification metrics used in our experimentation for CLX stock. It can be seen that for all models, the overall performance in terms of accuracy shows satisfying numbers.

From Table 12, we can see that the XGBoost algorithm, in most cases, outperformed the other three algorithms based on the Sharpe ratio (SR) and maximum drawdown (MDD) values. The SR shows the excess return from the excess risk taken by the trading algorithm. The ratio indicates a good and very good investment in case the SR is in order of 1 or 2, respectively, and if it reaches 3 or higher, investment performance is considered excellent. Our tri-state labeling algorithm could produce SR values greater than 2 for each of the assets under management while maintaining a very low MDD value for each of them. As it can be seen from Table 12, the percentage of maximum drawdown in most cases does not exceed 10. The average performance of our proposed framework is shown in Table 13. First, it can be seen from the Proposed Framework column that XGBoost has outperformed other algorithms in our experimentation in terms of annualized Sharpe ratio. The "Comparison Frameworks" column in Table 13 compares our proposed framework with more sophisticated frameworks. Based on Convolutional Neural Networks (CNN), Hoseinzade and Haratizadeh [17] introduced 2D-CNNpred and 3D-CNNpred frameworks to extract features for market trend prediction automatically. 2D-CNNpred predicts future market trends based on its historical performance, while 3D-CNNpred does the task by incorporating other markets' historical information. The authors exploited 82 variables for their input feature vector. Their annualized SR values for 2D-CNNpred and 3D-CNNpred are reported in Table 13. Their highest score is 2.257, belonging to 2D-CNNpred, which is about 25% below our best SR value, 2.823. Kim and Khushi [22] introduced a Deterministic Policy Gradient with 2D Relative-attentional Gated Transformer (DPGRGT) model in the combination of historical OHLCV data to maximize the portfolio optimization reward using deep reinforcement learning. Their model could achieve an annualized SR equal to 0.6418 which is 339.9% less than what our model achieved. In another sophisticated model to attempt market trend prediction, Picasso et al. [32] exploited both technical and sentiment analysis to solve the problem as a classification task. They combined 10 technical indicators as information from historical stock data

Table 3 Hyperparameter optimization of recurrent neural networks for AMD stock

LSTM					GRU				
Accuracy	*F*1-Score	Dropout	Units	l.r	Accuracy	*F*1-Score	Dropout	Units	l.r
0.91209	**0.90832**	**0.2**	**24**	**0.0013**	0.88571	0.87882	0.2	48	0.00102
0.9033	0.89778	0.4	64	0.00165	0.91209	0.90491	0.35	32	0.00923
0.9033	0.90412	0.3	48	0.00421	0.88352	0.87358	0.4	64	0.00503
0.88791	0.88588	0.3	24	0.00146	0.91209	0.90408	0.2	64	0.00274
0.9033	0.89748	0.35	64	0.00168	0.90549	0.90176	0.2	64	0.00813
0.89451	0.89012	0.4	64	0.03355	0.90769	0.89919	0.25	48	0.00127
0.8989	0.89013	0.25	64	0.00108	0.9033	0.89012	0.4	48	0.00242
0.87033	0.8709	0.3	32	0.00252	0.89451	0.87688	0.3	24	0.00343
0.88132	0.88102	0.25	48	0.02173	0.89451	0.88433	0.35	48	0.00101
0.89451	0.87768	0.4	64	0.02368	0.82857	0.75089	0.35	48	0.01142
0.9011	0.89575	0.3	32	0.00979	0.8967	0.89996	0.25	32	0.00372
0.90549	0.89988	0.2	48	0.00117	0.82857	0.75089	0.35	48	0.00211
0.90989	0.90269	0.2	48	0.02547	0.90549	0.89981	0.35	48	0.00935
0.89231	0.88145	0.25	32	0.0182	0.89451	0.8785	0.3	24	0.00259
0.88571	0.87547	0.2	32	0.0246	0.82857	0.75089	0.2	64	0.00112
0.9011	0.89999	0.4	48	0.00131	0.91429	0.90919	0.2	32	0.00339
0.88791	0.87228	0.3	32	0.00776	0.87692	0.85466	0.35	64	0.00402
0.8967	0.88018	0.35	48	0.00194	0.89011	0.87852	0.2	24	0.00425
0.86813	0.84878	0.3	24	0.02053	0.9033	0.89882	0.2	24	0.0072
0.89011	0.88068	0.2	64	0.00106	0.9033	0.89793	0.25	24	0.00611
0.88791	0.88289	0.4	24	0.00166	**0.91648**	**0.91218**	**0.2**	**24**	**0.00183**
0.90769	0.89961	0.3	48	0.00132	0.89231	0.87961	0.35	64	0.00233
0.90769	0.90329	0.4	48	0.02314	0.91429	0.91066	0.35	32	0.00394
0.89231	0.88607	0.3	32	0.03195	0.89231	0.8795	0.25	64	0.01113
0.86813	0.87035	0.3	24	0.03019	0.89451	0.88695	0.25	32	0.00204

Best tuning values and the respective performance metrics are shown in bold

with textual financial news about the stock under study. Their highest annualized SR was reported for the case they had applied the dictionary of Loughran and Mcdonald [26] (L&Mc) to textual data for feature extraction. As can be seen from Table 13, L&Mc (News) and L&Mc (News and Price) are 1.235 and 0.756, while our model's performance is 128.6% and 273.4% higher than their models, respectively. Our comparison with some recent sophisticated studies in terms of both models used to predict the market and the feature engineering process exploited indicates that our proposed labeling algorithm could have successfully extracted more effective buy and sell opportunities resulting in higher annual trading performance.

In Fig. 10, we have plotted each stock's RoR to illustrate the evolution of our trading system's return to measure how much profits are made using our system through an investment, over time. The comparison charts prove that our labeling algorithm successfully achieves a high positive return even with a stock such

Table 4 Hyperparameter optimization of recurrent neural networks for CLX stock

LSTM					GRU				
Accuracy	*F*1-Score	Dropout	Units	l.r	Accuracy	*F*1-Score	Dropout	Units	l.r
0.85403	0.83438	0.25	32	0.00305	0.91068	0.90667	0.2	64	0.00107
0.91285	**0.90024**	**0.2**	**24**	**0.00189**	0.90196	0.8878	0.4	32	0.00264
0.90414	0.89121	0.25	64	0.00269	0.87582	0.8568	0.4	32	0.00169
0.90196	0.89477	0.25	24	0.02077	0.9085	0.89966	0.2	32	0.00167
0.89107	0.87656	0.25	32	0.0029	0.91068	0.90265	0.2	32	0.00216
0.84096	0.79521	0.4	32	0.03533	0.90414	0.89104	0.2	32	0.00171
0.89542	0.88591	0.25	64	0.0021	0.9085	0.8975	0.25	64	0.00103
0.90285	0.90176	0.25	24	0.00335	0.88671	0.87622	0.25	48	0.00232
0.88017	0.86976	0.2	48	0.02179	0.89107	0.88824	0.25	24	0.00234
0.9085	0.90066	0.2	48	0.0129	0.81481	0.73167	0.3	48	0.0058
0.90196	0.89181	0.2	64	0.00236	**0.93028**	**0.9234**	**0.2**	**24**	**0.00191**
0.89978	0.88549	0.2	32	0.02723	0.90632	0.89653	0.35	32	0.00499
0.90414	0.89392	0.4	32	0.0026	0.81481	0.73167	0.4	64	0.00142
0.88453	0.87549	0.3	32	0.01627	0.89325	0.87644	0.35	32	0.00179
0.8976	0.88662	0.2	24	0.01153	0.88453	0.87057	0.3	32	0.00222
0.90196	0.89527	0.25	24	0.00406	0.91068	0.90066	0.25	48	0.00128
0.85839	0.83964	0.25	32	0.00245	0.90414	0.89726	0.2	32	0.00153
0.91503	0.90515	0.25	32	0.0027	0.90414	0.89101	0.25	48	0.00117
0.90632	0.89613	0.3	64	0.00358	0.84532	0.7925	0.2	32	0.00182
0.85621	0.82603	0.2	48	0.00223	0.90196	0.89281	0.2	48	0.00202
0.87364	0.86368	0.3	64	0.00437	0.9085	0.89956	0.2	32	0.00118
0.87582	0.8611	0.2	64	0.03391	0.92157	0.91621	0.2	32	0.00148
0.91068	0.90329	0.4	24	0.00407	0.89978	0.88628	0.4	64	0.00109
0.89542	0.89053	0.3	24	0.01018	0.89325	0.88721	0.25	32	0.00436
0.85839	0.83285	0.2	24	0.00366	0.91503	0.90694	0.25	24	0.00354

Best tuning values and the respective performance metrics are shown in bold

as Macy's Inc., M, which has a high systematic risk value with a high likelihood to show extreme negative returns. As it can be seen from Fig. 10-, although the trained GRU model has failed to show a positive return on our labeled test set, XGBoost has surprisingly shown a high percentage of return, i.e., greater than 150% at the end of the test period. In Fig. 11, we have depicted the pick-to-trough decline during our investment test period. Drawdowns are considered a measure of downside volatility and are essential in monitoring the trading performance. Since volatile markets and large drawdowns are problematic for most investors, they usually choose to avoid values greater than 20 percent. In most cases, the drawdown value does not exceed 10 percent. This is a significant improvement that comes from our proposed labeling algorithm since when a trading strategy keeps the investor out of trouble, it results in starting to compound at a higher level.

Table 5 Hyperparameter optimization of recurrent neural networks for M stock

LSTM					GRU				
Accuracy	*F*1-Score	Dropout	Units	l.r	Accuracy	*F*1-Score	Dropout	Units	l.r
0.86822	0.84763	0.3	24	0.00467	0.86047	0.86183	0.35	24	0.00199
0.83915	0.76576	0.2	32	0.01753	0.87597	0.85821	0.3	64	0.01056
0.8469	0.78649	0.35	48	0.00597	0.83915	0.76576	0.35	24	0.03173
0.8469	0.78338	0.4	32	0.01301	0.8469	0.83781	0.25	48	0.01737
0.87209	0.86186	0.4	32	0.0162	0.81395	0.8018	0.25	24	0.01998
0.8469	0.84341	0.3	24	0.00853	0.85078	0.84806	0.25	32	0.01251
0.85078	0.8074	0.35	48	0.01066	0.85078	0.85066	0.2	64	0.00283
0.78295	0.80698	0.35	48	0.02937	0.87597	0.87195	0.2	64	0.00781
0.85853	0.85415	0.4	32	0.00521	0.86434	0.85796	0.25	32	0.01924
0.87209	0.84343	0.35	64	0.01607	0.83333	0.83791	0.2	48	0.00374
0.85465	0.84037	0.35	48	0.01052	0.86047	0.85982	0.35	24	0.00381
0.83527	0.78557	0.2	64	0.01132	0.87016	0.86487	0.35	24	0.00274
0.86434	0.84591	0.4	64	0.01385	0.83915	0.76576	0.35	64	0.01673
0.83915	0.76576	0.2	32	0.00811	0.8469	0.78338	0.35	32	0.01054
0.85078	0.84877	0.3	24	0.01377	0.85465	0.85722	0.2	64	0.00661
0.82752	0.79296	0.35	24	0.00533	0.85271	0.85549	0.25	32	0.00704
0.83915	0.76576	0.4	64	0.00476	0.87403	0.86877	0.2	24	0.0026
0.85078	0.8462	0.4	48	0.00704	0.83915	0.76576	0.25	48	0.01857
0.83915	0.76576	0.2	24	0.01712	0.83333	0.84691	0.3	32	0.0135
0.83915	0.76576	0.35	24	0.01963	0.85853	0.84222	0.2	48	0.00768
0.85271	0.80062	0.3	24	0.02079	0.84884	0.84699	0.2	32	0.00239
0.87582	0.8611	0.2	64	0.03391	**0.91403**	**0.8702**	**0.2**	**24**	**0.00636**
0.91068	**0.90329**	**0.2**	**24**	**0.00193**	0.8469	0.81967	0.4	24	0.01267
0.89542	0.89053	0.3	24	0.01018	0.83915	0.76576	0.25	32	0.00389
0.85839	0.83285	0.2	24	0.00366	0.86434	0.82619	0.3	48	0.0185

Best tuning values and the respective performance metrics are shown in bold

4 Conclusion

In this paper, we designed a multi-class classification framework to tackle the price trend prediction problem. The framework was implemented using two machine learning models, SVM and XGBoost, and two recurrent neural networks, LSTM and GRU. The reason behind exploiting different classification models in our trend prediction module is to show that regardless of the classifier used, the tri-state labeling algorithm extracts more profitable buy and sell opportunities from price data. This study contributed to the market trend prediction problem in four ways. First, our tri-state labeling algorithm helps filter low-confidence states of the market. For example, when the classification machine is not confident enough about whether the trend is upward or downward, it changes the system status to the idle state (denoted by 0). This is a safe position that is taken by the machine not to pose a threat to the investor's capital in a highly volatile market. Second, in the model training part of the work, the combinatorial purged

Table 6 Hyperparameter optimization for SVM classification

SVC						
Accuracy	*F*1-Score	*C*	Kernel	Degree	gamma	Time (s)
0.88791	0.86462	0.97	rbf	4	6.35122	34.08338
0.88791	0.86267	0.74	rbf	4	4.61141	34.47854
0.83516	0.76597	0.6	rbf	3	4.1091	34.66601
0.89011	0.86669	0.92	rbf	2	8.8002	33.99457
0.88791	0.86374	0.69	rbf	2	5.24261	33.50706
0.89011	0.86669	0.7	rbf	4	5.92944	33.40672
0.89011	0.86665	0.97	rbf	4	2.56692	35.21639
0.88791	0.86271	0.56	rbf	4	4.81323	34.29156
0.89451	0.87525	1	rbf	3	8.67797	32.86624
0.89011	0.86665	0.99	rbf	4	8.42303	33.2351
0.85714	0.80872	0.85	rbf	4	4.69465	34.28548
0.88571	0.86038	0.94	rbf	3	2.38	34.4737
0.84615	0.78804	1	rbf	4	7.12775	34.51734
0.89231	0.87146	0.95	rbf	4	8.49369	33.89479
0.88571	0.85976	0.92	rbf	2	5.8358	34.21499
0.89011	0.86665	0.88	rbf	2	6.29374	32.85993
0.89451	**0.8742**	**0.96**	**rbf**	**3**	**6.2801**	**33.20176**
0.81538	0.8109	0.9	rbf	4	3.55488	32.62426
0.88352	0.85675	0.69	rbf	3	3.60119	34.24339
0.88571	0.8619	0.92	rbf	3	6.67293	33.9439
0.86374	0.82297	0.89	rbf	4	6.7315	33.09472
0.86593	0.82634	0.88	rbf	3	7.93163	34.81558
0.84396	0.78431	0.92	rbf	3	8.93898	33.51085
0.88791	0.86271	1	rbf	4	8.9172	34.22566
0.89011	0.86669	0.59	rbf	4	3.71744	34.04539

The values are reported for AMD stock

Best tuning values and the respective performance metrics are shown in bold

K-Fold cross-validation is applied to the training and test dataset splitting task to prevent data leakage and look-ahead bias. This type of cross-validation assures less bias in the prediction because of data leakage between training and validation chunks of the input vector. This is in marked contrast to most of the previous works applying the common K-fold cross-validation technique which is ill-suited to deal with data leakage and look-ahead bias. Third, we have trained the final model with the best tuning hyperparameters found by applying a Bayesian hyperparameter optimization process. Finally, we have successfully back-tested our framework on selected stocks from the S&P-500 market and showed through extensive experiments that our training regime is applicable to different models, resulting in high performance of the trading task.

The proposed study has some limitations that can be considered as anchor points for further extensions and improvements. While the study provides valuable insights on trend following based on price changes, the labeling mechanism

Table 7 Hyperparameter optimization for XGBoost

XGBoost											
Accuracy	*F*1-Score	max_depth	gamma	reg_alpha	reg_lambda	colsample_bytree	min_child_weight	sampling_method	n_estimators	objective	Time
0.8637	0.8215	18	1.1051	41	0.3018	0.7678	10	gradient_based	696	multi:softmax	63.4289
0.8637	0.8289	11	3.969	40	0.6491	0.5095	2	gradient_based	781	multi:softprob	55.0986
0.8747	0.8458	15	3.4775	70	0.37	0.9825	7	uniform	355	multi:softmax	50.6152
0.8879	0.8697	17	2.8832	42	0.8175	0.8962	10	uniform	454	multi: softmax	51.8069
0.8703	0.8411	12	1.4598	87	0.9679	0.8299	10	gradient_based	318	multi:softprob	56.6415
0.8725	0.8442	7	1.2186	41	0.0171	0.9797	7	uniform	967	multi:softmax	50.2201
0.8725	0.8439	18	8.1591	43	0.2704	0.7073	6	gradient_based	486	multi:softmax	65.4552
0.8703	0.8409	14	3.0756	42	0.1931	0.8954	10	gradient_based	502	multi:softprob	57.5294
0.8813	0.8576	15	2.7026	56	0.7277	0.7741	7	uniform	423	multi:softprob	55.8899
0.8791	0.8537	11	1.0769	40	0.531	0.9712	9	uniform	667	multi:softmax	60.1367
0.8879	0.8683	15	1.2257	47	0.5404	0.6589	8	gradient_based	273	multi:softmax	59.3627
0.8747	0.8463	18	1.2111	43	0.241	0.5436	10	uniform	450	multi:softmax	65.7168
0.8835	0.8603	11	2.2105	62	0.7019	0.986	7	gradient_based	682	multi:softprob	42.5147
0.8857	0.8644	14	1.0571	40	0.498	0.6547	10	gradient_based	683	multi:softmax	65.4253
0.8791	0.8536	18	7.089	40	0.9822	0.7151	3	uniform	990	multi:softprob	83.399
0.8593	0.818	14	1.0921	40	0.2623	0.6218	0	uniform	611	multi:softmax	66.2685
0.8945	**0.8768**	**7**	**1.1738**	**40**	**0.7286**	**0.5933**	**4**	**uniform**	**640**	**multi:softmax**	**53.2042**
0.8791	0.8526	9	2.2144	41	0.0674	0.6766	10	gradient_based	501	multi:softmax	58.5607
0.8615	0.8276	14	1.3054	58	0.2625	0.5855	2	gradient_based	613	multi:softprob	49.1764
0.8791	0.8537	4	1.9456	59	0.7058	0.976	4	uniform	351	multi:softprob	50.5104
0.8769	0.851	16	3.8586	54	0.4508	0.8698	3	uniform	478	multi:softmax	74.8684

Table 7 (continued)

XGBoost											
Accuracy	*F*1-Score	max_depth	gamma	reg_alpha	reg_lambda	colsample_bytree	min_child_weight	sampling_method	n_estimators	objective	Time
0.8791	0.8529	7	1.7158	40	0.9097	0.6642	4	gradient_based	786	multi:softprob	45.9424
0.8769	0.851	4	4.0713	41	0.0022	0.9273	6	uniform	574	multi:softmax	43.6041
0.8769	0.851	8	5.3256	41	0.6899	0.8038	1	uniform	948	multi:softmax	47.8001
0.8725	0.8429	14	7.1177	42	0.6682	0.8005	8	gradient_based	334	multi:softmax	63.6851

The values are reported for AMD stock

Best tuning values and the respective performance metrics are shown in bold

Table 8 Hyperparameter optimization for XGBoost

XGBoost											
Accuracy	*F*1-Score	max_depth	gamma	reg_alpha	reg_lambda	colsample_bytree	min_child_weight	sampling_method	n_estimators	objective	Time
0.817	0.7741	6	5.931	41	0.3727	0.7007	0	gradient_based	372	multi:softmax	24.0615
0.7974	0.7075	7	2.7059	113	0.0845	0.9575	9	uniform	198	multi:softprob	28.5653
0.8366	0.812	18	1.3742	41	0.4262	0.9842	3	gradient_based	191	multi:softprob	37.0794
0.8105	0.7604	18	6.8847	40	0.6793	0.6817	7	uniform	305	multi:softprob	30.7957
0.8279	0.7962	9	1.0936	40	0.5142	0.7101	3	uniform	299	multi:softprob	30.1246
0.8366	0.8141	18	2.1214	44	0.4773	0.815	5	uniform	359	multi:softmax	33.8445
0.8061	0.7334	5	5.6337	50	0.8566	0.7466	3	uniform	295	multi:softmax	22.1428
0.8214	0.7825	10	4.2535	42	0.0339	0.7193	2	uniform	328	multi:softmax	31.4881
0.8844	0.8699	16	2.4258	40	0.0621	0.9548	8	uniform	210	multi:softprob	27.5668
0.902	**0.890**	**18**	**8.9265**	**40**	**0.6958**	**0.9409**	**4**	**gradient_based**	**183**	**multi:softmax**	**32.7001**
0.8148	0.7685	10	4.2891	50	0.0452	0.6634	4	uniform	344	multi:softmax	24.2424
0.8301	0.7982	12	4.4569	41	0.9636	0.9874	6	uniform	283	multi:softprob	31.7568
0.7974	0.7075	13	7.0061	41	0.4065	0.9421	4	gradient_based	298	multi:softprob	32.9238
0.8279	0.7947	12	8.8421	59	0.3582	0.5612	6	gradient_based	304	multi:softmax	30.7019
0.8366	0.8148	11	2.5667	40	0.3796	0.976	2	uniform	266	multi:softprob	30.2273
0.8126	0.7601	6	4.0282	44	0.6599	0.4823	1	uniform	399	multi:softmax	23.8804
0.8061	0.7539	7	1.6378	40	0.4469	0.3806	0	gradient_based	386	multi:softprob	23.8486
0.8279	0.7895	4	1.8386	53	0.7457	0.9378	1	gradient_based	249	multi:softmax	25.9997
0.8622	0.8535	18	8.976	40	0.0084	0.8736	10	uniform	265	multi:softmax	30.6596
0.8912	0.875	13	3.3235	40	0.1619	0.9776	6	uniform	323	multi:softprob	25.5677
0.8061	0.7328	5	1.6994	46	0.7916	0.983	10	uniform	181	multi:softmax	23.8973

Table 8 (continued)

XGBoost											
Accuracy	*F*1-Score	max_depth	gamma	reg_alpha	reg_lambda	colsample_bytree	min_child_weight	sampling_method	n_estimators	objective	Time
0.8301	0.7949	4	6.3871	41	0.9904	0.8966	1	uniform	327	multi:softprob	23.898
0.8192	0.7805	12	2.8584	40	0.9966	0.9889	8	uniform	400	multi:softprob	33.197
0.8279	0.7932	7	4.7116	94	0.7571	0.9516	3	gradient_based	240	multi:softmax	28.2843
0.7974	0.7307	14	7.9455	42	0.5891	0.9898	10	uniform	274	multi:softprob	32.7641

The values are reported for CLX stock

Best tuning values and the respective performance metrics are shown in bold

Table 9 Hyperparameter optimization for XGBoost

XGBoost											
Accuracy	*F*1-Score	max_depth	gamma	reg_alpha	reg_lambda	colsample_bytree	min_child_weight	sampling_method	n_estimators	objective	Time (s)
0.8574	0.8011	14	2.8253	43	0.1571	0.8231	13	uniform	520	multi:softmax	22.5147
0.8438	0.7722	6	4.4761	41	0.707	0.983	10	gradient_based	698	multi:softprob	22.8963
0.8438	0.7722	8	1.0467	44	0.4469	0.8078	13	gradient_based	667	multi:softmax	24.2549
0.8574	0.8011	8	2.2016	73	0.2234	0.8758	20	gradient_based	644	multi:softmax	23.5862
0.8438	0.7722	13	1.0282	70	0.2868	0.9875	20	uniform	589	multi:softprob	21.5718
0.8555	0.7975	6	5.8813	68	0.4518	0.417	8	uniform	694	multi:softmax	20.2554
0.8652	**0.8155**	**14**	**1.0441**	**83**	**0.3727**	**0.982**	**10**	**uniform**	**819**	**multi:softmax**	**23.6984**
0.8438	0.7722	14	4.6023	73	0.9216	0.2635	16	gradient_based	257	multi:softprob	18.0163
0.8574	0.8011	9	2.6645	74	0.0395	0.391	14	gradient_based	568	multi:softprob	20.9973
0.8633	0.8111	13	3.233	87	0.6541	0.8851	2	gradient_based	288	multi:softmax	28.0208
0.8574	0.8011	12	2.0782	73	0.3246	0.2529	1	gradient_based	626	multi:softmax	19.2962
0.8438	0.7722	7	2.0008	79	0.3725	0.9389	20	uniform	352	multi:softprob	23.9215
0.8438	0.7722	18	3.8969	40	0.764	0.981	10	uniform	369	multi:softmax	21.9027
0.8438	0.7722	18	3.1102	40	0.7071	0.8959	8	gradient_based	288	multi:softmax	29.7171
0.8438	0.7722	15	4.6847	87	0.0333	0.843	20	gradient_based	749	multi:softprob	21.9142
0.8438	0.7722	12	1.0602	62	0.2624	0.7675	5	gradient_based	751	multi:softmax	25.447
0.8594	0.8045	13	1.3293	41	0.6298	0.7549	5	gradient_based	523	multi:softprob	23.9069
0.8574	0.8011	7	7.2568	62	0.5774	0.8555	3	uniform	702	multi:softmax	25.7756
0.8574	0.8011	15	2.2939	60	0.7819	0.5265	0	uniform	615	multi:softmax	18.968
0.8438	0.7722	14	4.0672	61	0.4329	0.9167	16	gradient_based	896	multi:softprob	22.0706
0.8438	0.7722	7	1.0534	79	0.4175	0.76	20	gradient_based	527	multi:softmax	22.7151

Table 9 (continued)

XGBoost											
Accuracy	*F*1-Score	max_depth	gamma	reg_alpha	reg_lambda	colsample_bytree	min_child_weight	sampling_method	n_estimators	objective	Time (s)
0.8633	0.8111	6	1.3747	67	0.9937	0.6049	9	gradient_based	700	multi:softprob	21.7321
0.8594	0.8045	17	4.4469	87	0.3973	0.662	20	uniform	258	multi:softmax	22.6639
0.8652	0.8142	11	1.0097	40	0.3281	0.8083	11	gradient_based	343	multi:softmax	27.3033
0.8438	0.7722	16	1.1164	44	0.8867	0.9735	20	gradient_based	944	multi:softprob	22.6539

The values are reported for M stock

Best tuning values and the respective performance metrics are shown in bold

Table 10 Hyperparameter optimization for SVM classification

SVC						
Accuracy	*F*1-Score	*C*	Kernel	Degree	gamma	Time (s)
0.54773	0.42636	0.96	rbf	2	8.96451	68.55751
0.58864	0.51771	0.6	rbf	3	3.23122	61.5789
0.58182	0.50193	1	rbf	4	8.94659	63.0378
0.61591	0.56178	0.75	rbf	4	5.24323	60.19616
0.69318	0.66328	0.44	rbf	4	8.99035	63.81414
0.70909	0.68182	0.98	rbf	2	8.99531	62.34934
0.70455	0.67627	0.4	rbf	4	8.71733	62.89979
0.87115	0.85123	0.99	rbf	3	7.43513	64.88739
0.71136	0.68461	0.97	rbf	4	8.86893	62.36296
0.68636	0.65382	0.44	rbf	2	8.9409	62.44242
0.52045	0.35631	0.69	rbf	3	8.8603	63.15386
0.53409	0.39643	0.8	rbf	3	8.83122	60.07399
0.70227	0.67503	0.93	rbf	2	8.81325	60.41432
0.70909	0.68228	0.84	rbf	3	8.87425	62.6169
0.71136	0.68461	0.96	rbf	3	8.6247	64.30415
0.88691	**0.86695**	**1**	**rbf**	**3**	**7.332**	**60.75189**
0.70455	0.67738	0.68	rbf	2	8.98146	63.31783
0.70909	0.68228	1	rbf	4	7.34932	60.10106
0.69318	0.66289	0.82	rbf	4	3.2027	62.91097
0.68864	0.66082	0.68	rbf	4	8.98826	63.06187
0.70227	0.67419	0.71	rbf	3	8.27532	62.57946
0.70455	0.67712	1	rbf	3	8.57879	60.85747
0.59318	0.51773	0.98	rbf	3	8.96061	63.29737
0.625	0.60501	0.7	rbf	4	7.19191	60.17934
0.69091	0.66149	0.71	rbf	4	8.82945	61.37867

The values are reported for CLX stock

Best tuning values and the respective performance metrics are shown in bold

can be further enhanced to show more robustness against higher levels of volatility. Besides, we believe that there is also more room for improvement in labeling the trends using an adaptive threshold. Such an adaptive parameter optimization may lead the system through more volatile periods to capture trends better. Finally, the activity at each price level results in changes in volume, which affects the price. This means we need to study the mutual effect of price and volume in future works to extract the trends better.

Table 11 Classification metrics report for CLX stock

Class label	BB-XGBoost				BB-SVM			
	Precision	Recall	*F*1-score	Support	Precision	Recall	*F*1-score	Support
− 1	0.870	0.455	0.597	44	0.941	0.364	0.525	44
0	0.914	0.992	0.951	374	0.894	0.997	0.943	374
1	0.767	0.561	0.648	41	0.720	0.439	0.545	41
Summary								
Accuracy			0.902	459			0.887	459
Macro avg	0.850	0.669	0.732	459	0.852	0.600	0.671	459
Weighted avg	0.896	0.902	0.890	459	0.883	0.887	0.867	459
Class label	BB-LSTM				BB-GRU			
	Precision	Recall	*F*1-score	Support	Precision	Recall	*F*1-score	Support
− 1	0.692	0.409	0.514	44	0.950	0.432	0.594	44
0	0.883	0.992	0.935	374	0.917	0.981	0.948	374
1	0.692	0.220	0.333	41	0.744	0.707	0.725	41
Summary								
Accuracy			0.867	459			0.904	459
Macro avg	0.756	0.540	0.594	459	0.870	0.707	0.756	459
Weighted avg	0.848	0.867	0.841	459	0.905	0.904	0.894	459

The threshold and window size are 0.05 and 11, respectively

Table 12 Performance comparison of LSTM, GRU, XGBoost, and SVM

Stock	Algorithm	SR	MDD	Time (s)	Stock	Algorithm	SR	MDD	Time (s)
AMD	LSTM	2.66	7.22	211.84	STX	LSTM	1.94	2.16	127.58
	GRU	2.44	2.32	241.81		GRU	1.96	17.27	145.10
	XGBoost	**3.55**	**6.98**	**4.155**		XGBoost	**2.55**	**2.16**	**1.204**
	SVM	2.56	7.22	2.565		SVM	2.16	2.16	0.873
WMT	LSTM	1.28	7.38	213.89	M	LSTM	0.73	2.98	327.65
	GRU	− 1.98	8.97	236.885		GRU	2.39	1.79	325.11
	XGBoost	**3.10**	**4.03**	**3.668**		XGBoost	**2.52**	**1.79**	**3.897**
	SVM	1.37	7.71	3.498		SVM	2.34	2.61	1.925
AAPL	LSTM	2.61	10.37	215.11	CLX	LSTM	0.81	2.76	218.7
	GRU	2.52	10.38	238.48		GRU	2.72	2.76	237.30
	XGBoost	**2.67**	**4.24**	**4.657**		XGBoost	**2.55**	**1.83**	**5.528**
	SVM	2.57	6.25	2.179		SVM	1.04	2.56	1.302

Best tuning values and the respective performance metrics are shown in bold

Table 13 Average performance comparison

Proposed framework		Comparison frameworks	
Algorithm	SR	Algorithm	SR
LSTM	1.672	2D-CNNpred [17]	2.257
GRU	1.675	3D-CNNpred [17]	2.243
XGBoost	**2.823**	DPGRGT [22]	0.642
SVM	2.007	L&Mc (News) [32]	1.235
		L&Mc (News and Price) [32]	0.756

Sharpe ratio for all studies is reported in annualized rate

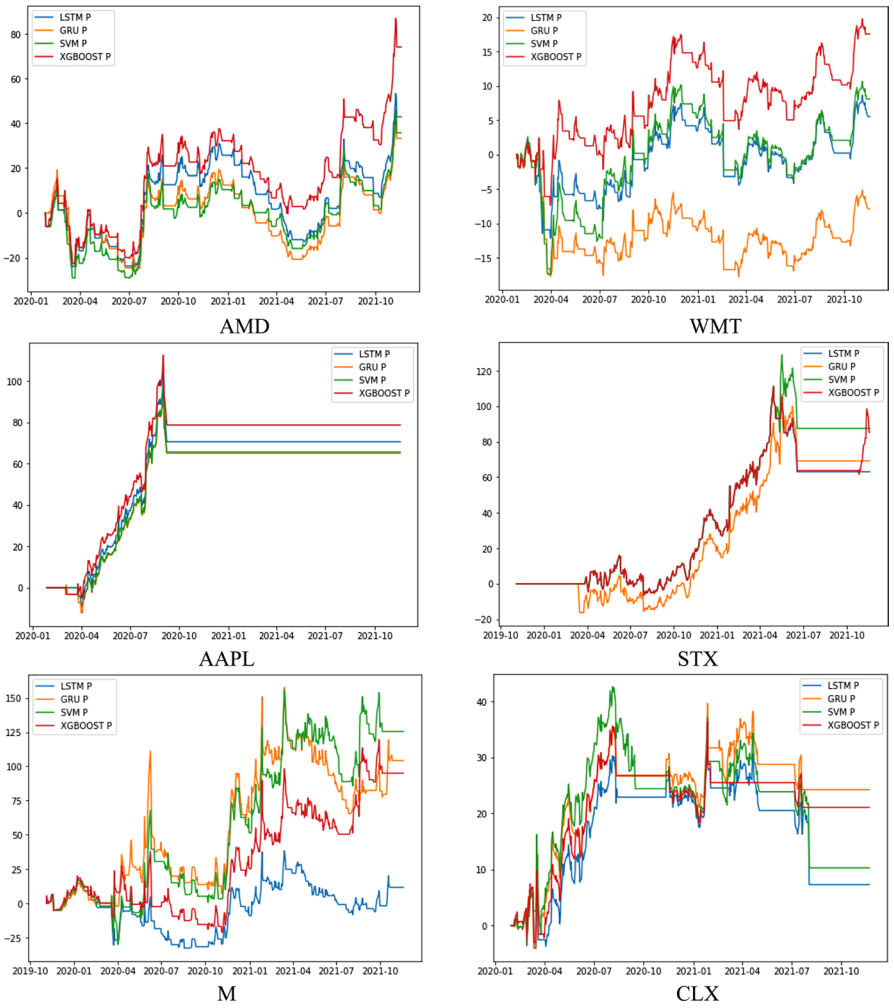

Fig. 10 RoR diagrams to show how much return is produced within each learning model. The horizontal axis represents date and the vertical one is the percentage of RoR. The charts show the RoR from January 2020 to November 2021

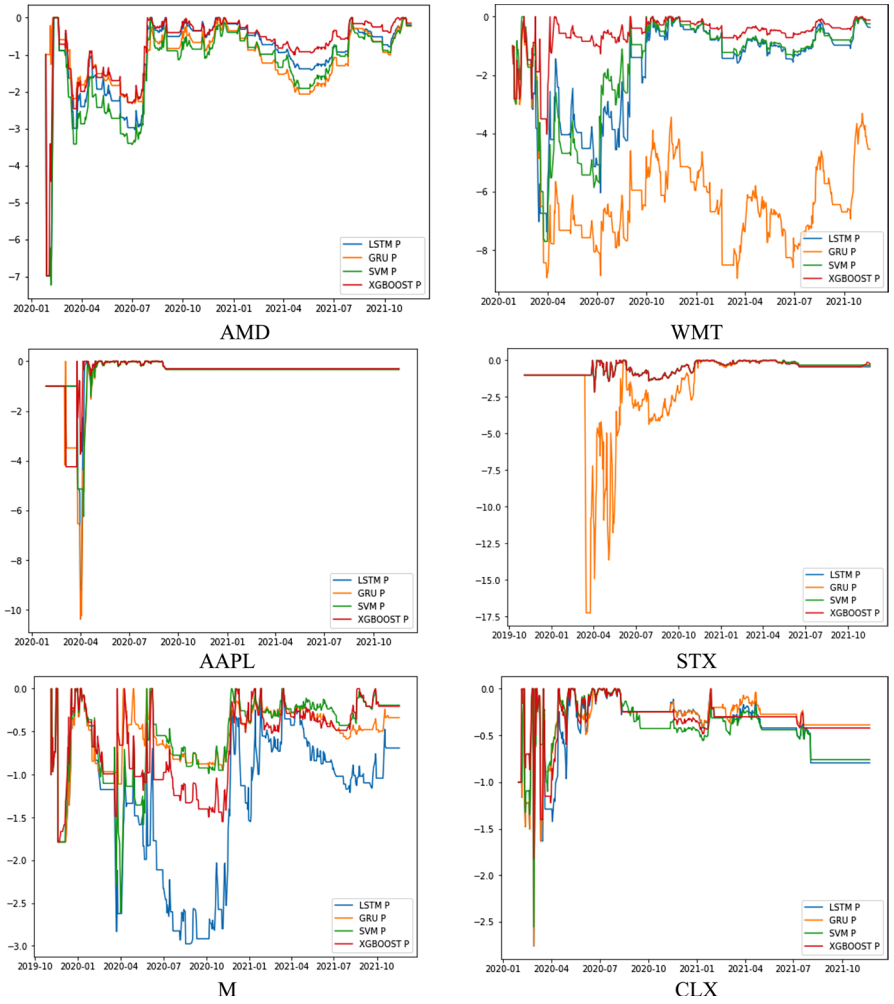

Fig. 11 Draw-Down comparison between classification algorithms used within the proposed framework. XGBoost has smaller DDs during the back-testing period

Data availability The authors confirm that the data analyzed in this study are openly available in yahoo finance at finance.yahoo.com and are included in supplementary information files. The authors also confirm that the data generated during the study are included in this published article and can be available upon request from the authors.

Declarations

Conflict of interest The authors declare that they have no financial and personal relationships with other people or organizations that can inappropriately influence the work reported in this paper.

References

1. Bai-Ling Z, Coggins R, Jabri MA, Dersch D, Flower B (2001) Multiresolution forecasting for futures trading using wavelet decompositions. IEEE Trans Neural Netw 12(4):765–775. https://doi.org/10.1109/72.935090
2. Bengio Y, Frasconi P, Simard P (1993) The problem of learning long-term dependencies in recurrent networks. In: IEEE International Conference on Neural Networks, pp 1183–1188. https://doi.org/10.1109/ICNN.1993.298725
3. Bergstra JS, Bardenet R, Bengio Y, Kégl B (n.d.) Algorithms for hyper-parameter optimization, p 9
4. Bustos O, Pomares-Quimbaya A (2020) Stock market movement forecast: a Systematic review. Expert Syst Appl 156:113464. https://doi.org/10.1016/j.eswa.2020.113464
5. Carta S, Ferreira A, Podda AS, ReforgiatoRecupero D, Sanna A (2021) Multi-DQN: an ensemble of Deep Q-learning agents for stock market forecasting. Expert Syst Appl 164:113820. https://doi.org/10.1016/j.eswa.2020.113820
6. Chen T, Guestrin C (2016) XGBoost: a scalable tree boosting system. In: Proceedings of the 22nd ACM SIGKDD International Conference on Knowledge Discovery and Data Mining, pp 785–794. https://doi.org/10.1145/2939672.2939785
7. Chen W, Zhang H, Mehlawat MK, Jia L (2021) Mean–variance portfolio optimization using machine learning-based stock price prediction. Appl Soft Comput 100:106943. https://doi.org/10.1016/j.asoc.2020.106943
8. Cho K, van Merrienboer B, Bahdanau D, Bengio Y (2014) On the properties of neural machine translation: encoder-decoder approaches. [Cs, Stat]. http://arxiv.org/abs/1409.1259
9. Cui Y, Chen Z, Wei S, Wang S, Liu T, Hu G (2017) Attention-over-attention neural networks for reading comprehension. In: Proceedings of the 55th annual meeting of the Association for Computational Linguistics (volume 1: long papers), pp 593–602. https://doi.org/10.18653/v1/P17-1055
10. Fernández-Macho J (2018) Time-localized wavelet multiple regression and correlation. Physica A 492:1226–1238. https://doi.org/10.1016/j.physa.2017.11.050
11. Gers FA, Schmidhuber E (2001) LSTM recurrent networks learn simple context-free and context-sensitive languages. IEEE Trans Neural Netw 12(6):1333–1340. https://doi.org/10.1109/72.963769
12 Guntu RK, Yeditha PK, Rathinasamy M, Perc M, Marwan N, Kurths J, Agarwal A (2020) Wavelet entropy-based evaluation of intrinsic predictability of time series. Chaos Interdiscip J Nonlinear Sci 30(3):033117. https://doi.org/10.1063/1.5145005
13. Gupta P, Majumdar A, Chouzenoux E, Chierchia G (2021) SuperDeConFuse: a supervised deep convolutional transform based fusion framework for financial trading systems. Expert Syst Appl 169:114206. https://doi.org/10.1016/j.eswa.2020.114206
14. Han KJ, Hahm S, Kim B-H, Kim J, Lane I (2017) Deep learning-based telephony speech recognition in the wild. Interspeech 2017:1323–1327. https://doi.org/10.21437/Interspeech.2017-1695
15. He K, Zhang X, Ren S, Sun J (2016) Deep residual learning for image recognition. In: 2016 IEEE Conference on Computer Vision and Pattern Recognition (CVPR), pp 770–778. https://doi.org/10.1109/CVPR.2016.90
16. Hochreiter S (2011) The vanishing gradient problem during learning recurrent neural nets and problem solutions. Int J Uncertain Fuzziness Knowl Based Syst. https://doi.org/10.1142/S0218488598000094
17. Hoseinzade E, Haratizadeh S (2019) CNNpred: CNN-based stock market prediction using a diverse set of variables. Expert Syst Appl 129:273–285. https://doi.org/10.1016/j.eswa.2019.03.029
18. Ioffe S, Szegedy C (2015) Batch normalization: accelerating deep network training by reducing internal covariate shift. [Cs]. http://arxiv.org/abs/1502.03167
19. Jung-Hua W, Jia-Yann L (1996) Stock market trend prediction using ARIMA-based neural networks. In: Proceedings of International Conference on Neural Networks (ICNN'96), vol 4, pp 2160–2165. https://doi.org/10.1109/ICNN.1996.549236
20. Kao L-J, Chiu C-C, Lu C-J, Chang C-H (2013) A hybrid approach by integrating wavelet-based feature extraction with MARS and SVR for stock index forecasting. Decis Support Syst 54(3):1228–1244. https://doi.org/10.1016/j.dss.2012.11.012
21. Kearns M, Laboratories TB, Hill M, Valiant L (1989) Cryptographic limitations on learning boolean formulae and finite automata. In: Proceedings of the Twenty-First Annual ACM Symposium on Theory of Computing, pp 433–444. https://doi.org/10.1145/73007.73049

22. Kim TW, Khushi M (2020) Portfolio optimization with 2D relative-attentional gated transformer. In: 2020 IEEE Asia-Pacific Conference on Computer Science and Data Engineering (CSDE), pp 1–6. https://doi.org/10.1109/CSDE50874.2020.9411635
23. Lee J, Koh H, Choe HJ (2021) Learning to trade in financial time series using high-frequency through wavelet transformation and deep reinforcement learning. Appl Intell 51(8):6202–6223. https://doi.org/10.1007/s10489-021-02218-4
24. Lee SI, Yoo SJ (2020) Threshold-based portfolio: the role of the threshold and its applications. J Supercomput 76(10):8040–8057. https://doi.org/10.1007/s11227-018-2577-1
25. Lee SI, Yoo SJ (2020) Multimodal deep learning for finance: integrating and forecasting international stock markets. J Supercomput 76(10):8294–8312. https://doi.org/10.1007/s11227-019-03101-3
26. Loughran T, Mcdonald B (2011) When is a liability not a liability? Textual analysis, dictionaries, and 10-Ks. J Finance 66(1):35–65. https://doi.org/10.1111/j.1540-6261.2010.01625.x
27. Lu J-Y, Lai H-C, Shih W-Y, Chen Y-F, Huang S-H, Chang H-H, Wang J-Z, Huang J-L, Dai T-S (2022) Structural break-aware pairs trading strategy using deep reinforcement learning. J Supercomput 78(3):3843–3882. https://doi.org/10.1007/s11227-021-04013-x
28. Marcos López de Prado (2018) Advances in financial machine learning. Wiley. https://www.wiley.com/en-us/Advances+in+Financial+Machine+Learning-p-9781119482086
29. Masset P (2008) Analysis of financial time-series using Fourier and wavelet methods. SSRN Electron J. https://doi.org/10.2139/ssrn.1289420
30. Pang X, Zhou Y, Wang P, Lin W, Chang V (2020) An innovative neural network approach for stock market prediction. J Supercomput 76(3):2098–2118. https://doi.org/10.1007/s11227-017-2228-y
31. Phan DHB, Sharma SS, Narayan PK (2015) Stock return forecasting: some new evidence. Int Rev Financ Anal 40:38–51. https://doi.org/10.1016/j.irfa.2015.05.002
32. Picasso A, Merello S, Ma Y, Oneto L, Cambria E (2019) Technical analysis and sentiment embeddings for market trend prediction. Expert Syst Appl 135:60–70. https://doi.org/10.1016/j.eswa.2019.06.014
33. Schapire RE (1990) The strength of weak learnability. Mach Learn 5:197–227. https://doi.org/10.1007/BF00116037
34. Shwartz-Ziv R, Armon A (2022) Tabular data: deep learning is not all you need. Inf Fusion 81:84–90. https://doi.org/10.1016/j.inffus.2021.11.011
35. Srivastava N, Hinton G, Krizhevsky A, Sutskever I, Salakhutdinov R (2014) Dropout: a simple way to prevent neural networks from overfitting. J Mach Learn Res 15(1):1929–1958
36. Tian L, Feng L, Yang L, Guo Y (2022) Stock price prediction based on LSTM and LightGBM hybrid model. J Supercomput. https://doi.org/10.1007/s11227-022-04326-5
37. Valencia F, Gómez-Espinosa A, Valdés-Aguirre B (2019) Price movement prediction of cryptocurrencies using sentiment analysis and machine learning. Entropy 21(6):589. https://doi.org/10.3390/e21060589
38. Wan C-X, Li B (2022) Financial causal sentence recognition based on BERT-CNN text classification. J Supercomput 78(5):6503–6527. https://doi.org/10.1007/s11227-021-04097-5
39. Wan X, Li H, Zhang L, Wu YJ (2022) Dimensionality reduction for multivariate time-series data mining. J Supercomput. https://doi.org/10.1007/s11227-021-04303-4
40. Wang X, Gan L, Liu S (2020) Research on intelligence analysis technology of financial industry data based on genetic algorithm. J Supercomput 76(5):3391–3401. https://doi.org/10.1007/s11227-018-2584-2
41. Wu D, Wang X, Su J, Tang B, Wu S (2020) A labeling method for financial time series prediction based on trends. Entropy 22(10):1162. https://doi.org/10.3390/e22101162
42. Yang J, Li J, Liu S (2020) A new algorithm of stock data mining in Internet of Multimedia Things. J Supercomput 76(4):2374–2389. https://doi.org/10.1007/s11227-017-2195-3
43. Yang L, Cai XJ, Zhang H, Hamori S (2016) Interdependence of foreign exchange markets: a wavelet coherence analysis. Econ Model 55:6–14. https://doi.org/10.1016/j.econmod.2016.01.022
44. Yazdani SF, Murad MAA, Sharef NM, Singh YP, Latiff ARA (2017) Sentiment classification of financial news using statistical features. Int J Pattern Recognit Artif Intell 31(03):1750006. https://doi.org/10.1142/S0218001417500069
45. Yousefi S, Weinreich I, Reinarz D (2005) Wavelet-based prediction of oil prices. Chaos Solitons Fractals 25(2):265–275. https://doi.org/10.1016/j.chaos.2004.11.015

46. Zhang Y, Yan B, Aasma M (2020) A novel deep learning framework: prediction and analysis of financial time series using CEEMD and LSTM. Expert Syst Appl 159:113609. https://doi.org/10.1016/j.eswa.2020.113609
47. Zhong X, Enke D (2017) Forecasting daily stock market return using dimensionality reduction. Expert Syst Appl 67:126–139. https://doi.org/10.1016/j.eswa.2016.09.027

Authors and Affiliations

Arsalan Dezhkam[1] · Mohammad Taghi Manzuri[1] · Ahmad Aghapour[1] · Afshin Karimi[1] · Ali Rabiee[1] · Shervin Manzuri Shalmani[2]

Arsalan Dezhkam
adejkam@ce.sharif.edu

Ahmad Aghapour
agapoorbonab@ce.sharif.edu

Afshin Karimi
fshnkarimi@ce.sharif.edu

Ali Rabiee
rabieee@ce.sharif.edu

Shervin Manzuri Shalmani
manzuris@mcmaster.ca

[1] Computer Engineering Department, Sharif University of Technology, Tehran, Iran

[2] Department of Computing and Software, McMaster University, Hamilton, ON, Canada