IEEE Access
Multidisciplinary | Rapid Review | Open Access Journal

Received May 25, 2021, accepted June 8, 2021, date of publication June 21, 2021, date of current version June 28, 2021.

Digital Object Identifier 10.1109/ACCESS.2021.3090834

Stock Ranking Prediction Using List-Wise Approach and Node Embedding Technique

SUMAN SAHA, (Graduate Student Member, IEEE), JUNBIN GAO, AND RICHARD GERLACH
Discipline of Business Analytics, The University of Sydney Business School, The University of Sydney, Sydney, NSW 2006, Australia

Corresponding author: Suman Saha (s.saha@sydney.edu.au)

This work was supported by the Business School Research Scholarship of The University of Sydney.

ABSTRACT Traditional stock movement prediction tasks are formulated as either classification or regression task, and the relation between stocks are not considered as an input of prediction. The relative order or ranking of stocks is more important than the price or return of a single stock for making proper investment decisions. Stock ranking performance can be improved by incorporating the stock relation information in the prediction task. We employ a graph-based approach for stock ranking prediction and use the stock relation information as the input of the machine learning model. Investors might be interested in the prediction performance of top-k stocks as they would be more profitable than the others. Thus, the performance measure for stock ranking prediction should be top-weighted and bounded for any value of k. Existing evaluation measures lack these properties, and we propose a new measure named normalized rank biased overlap for top-k ($NRBO@k$) stocks for stock ranking prediction. $NRBO@k$-based investment strategy generates 0.281% to 4.928% higher relative investment gain than the topmost stock-based strategy. We show that the list-wise loss function can improve the stock ranking performance significantly in a graph-based approach. It generates better $NRBO@10$ than the combination of point-wise and pair-wise loss in three out of four cases. Node embedding techniques such as Node2Vec can reduce the training time of graph-based approaches for stock ranking prediction significantly. Additionally, we improve the prediction performance through hyperparameter tuning of Node2Vec when a sparse stock relation graph is applied.

INDEX TERMS Stock ranking prediction, Node2Vec, normalized rank biased overlap, list-wise loss.

I. INTRODUCTION

Predicting stock price movement is very challenging due to the volatile nature of the stock market. There are different schools of thought regarding the predictability of individual stocks or the stock market. According to Efficient Market Hypothesis (EMH), it is not economically viable to predict stock price as the current stock price reflects all the available information in an efficient market [1]. However, the stock market cannot be perfectly efficient all the time. Irregularities and patterns emerge in markets for intermittent periods [2], [3]. Thus, it can be worthwhile to predict stock movement and use that prediction for making investment decisions.

The stock price can be influenced by numerous factors such as macro-economic factors [4], the past value of technical and fundamental indicators [5], [6], or news and online search data [7]–[9]. It is possible to predict the movement of a single stock or the market as a whole [5]. Stock movement can be predicted from multiple perspectives such as one day ahead up-down direction [5], [7], one day ahead actual stock price, or top-k stocks on the basis of predicted return [10], [11].

Most existing studies consider the stock movement prediction as a regression or classification task. In a regression task, the actual value of the stock index or the stock price is predicted [12], [13]. In a classification task, one day ahead up-down direction is predicted [5]. In both regression and classification tasks, only the features of a single stock are considered. According to the Capital Asset Pricing Model (CAPM), the individual stock return has a relation with the market return [14]. Thus, the stock price is not only influenced by its factors but also by its relationship with other stocks. It is necessary to consider the prediction task from a perspective different than simple regression or classification if we want to take the relationship between different stocks into account. It is also necessary to incorporate information about stock relations in the prediction task.

In recent times, some studies consider the stock prediction as a ranking task [9]–[11], [15]. In this case, the stocks are ranked according to the predicted return. A major challenge of the ranking task is to determine an optimal performance

The associate editor coordinating the review of this manuscript and approving it for publication was Ikramullah Lali.

measure. Existing studies measure the ranking performance in terms of topmost stock prediction such as the mean reciprocal rank of the top stock (MRRT) [10], the accuracy of outperforming cross-sectional median return [11], the accuracy of outperforming market return [15], or the normalized discounted cumulative gain (NDCG) [9]. We argue that these evaluation measures are suboptimal for the stock ranking prediction, and each of them has its limitations. We propose a robust measure named normalized rank biased overlap (NRBO) based on the concept of rank biased overlap (RBO) [16] for stock ranking prediction. NRBO is suitable for measuring any finite stock ranking performance. We will discuss the limitations of existing evaluation measures and details of our proposed evaluation measure in the latter sections.

Stock relations can be represented by graphs. It is possible to construct a stock market graph based on the relationship between stocks, such as whether the stocks are in the same industry or not. Incorporation of graph information can improve the stock ranking performance [10] or prediction accuracy [17] significantly. While predicting stock ranking using stock market graph information, it is essential to choose the proper loss function for improved performance. Traditional evaluation measures such as mean squared error (MSE) focus on the performance of individual stocks. They fail to capture relative performance, such as the rank of stocks. While predicting stock ranking using the graph-based approach, it is common to use a combination of the point-wise and the pair-wise loss as the objective function [10]. We show that the combination of the point-wise and pair-wise loss function is not optimal for stock ranking prediction using the graph-based approach, and the list-wise loss function results in better predictive performance in such cases.

In general, the relations defined by a graph are represented by the adjacency matrix. The adjacency matrix of the stock market graph is used as input features of the predictive model in a graph-based approach. As there are many stocks in a market, the adjacency matrix can be large and sparse. Moreover, two stocks can have multiple types of relations between them, which can result in a multi-graph. This multi-dimensional and large adjacency matrix can make the training process of the graph-based approach very slow. One way to speed up the training process is to apply node embedding techniques on the raw adjacency matrix. We show that a proper embedding technique such as Node2Vec [18] can result in significant improvement in the training time and comparable ranking performance simultaneously. It is possible to uplift the ranking performance to the baseline level for a sparse stock market graph when the hyperparameters of Node2Vec are tuned properly. Overall, the major contributions of this study are as follows:

- This study proposes a new metric named normalized rank biased overlap (NRBO) for measuring the stock ranking prediction performance.
- This study demonstrates the effectiveness of the list-wise loss function in a graph-based approach of stock ranking prediction through improved performance.
- This study shows significant improvement in training time of stock ranking prediction by incorporating Node2Vec. It generates comparable ranking performance through hyperparameter tuning of Node2Vec.

The rest of the paper is organized as follows. Section II discusses the existing works related to stock movement prediction, loss function, performance measure, and node embedding techniques. Section III details several theoretical concepts related to the proposed model and evaluation measure. The rationale of the proposed performance evaluation measure is discussed in Section IV. Section V describes the methodology, and Section VI describes the experimental setting. Detailed empirical results are presented in Section VII. The paper is concluded in Section VIII.

II. LITERATURE REVIEW

Traditionally, researchers use econometric methods such as the auto-regressive integrated moving average (ARIMA) [19] or the auto-regressive fractionally integrated moving average (ARFIMA) [20] for explaining or predicting stock movement. Machine learning (ML) models have shown their effectiveness in various fields such as image processing, speech processing, and genetic engineering in recent times, and their application for stock movement prediction is increasing gradually [21].

Two dominant ML models for stock movement prediction are artificial neural network (ANN) and support vector machine (SVM) [21]. Technical indicators are used as input features of ANN for predicting the next day direction of stocks and indices [5]. ANN is also used with multiple feature selection techniques to predict the quarterly direction of stocks [4]. ANN also works well when combined with different optimization techniques such as genetic algorithms [22]. SVM is used with hybrid feature selection techniques for predicting the next day direction of the NASDAQ index [23]. SVM can predict the next day direction of the high price, and that prediction can be used to generate a reliable trading strategy, even if the prediction accuracy is low [24].

The application of deep learning (DL) based models for stock movement or financial time series prediction is on the rise in recent times. DL based models are used for predicting stock ranking [11] or one-day ahead close price [12]. They are also capable of producing superior performance while predicting momentum and reversal effects in the stock market [6] or forecasting the actual value of a stock index [13]. They tend to outperform ANN or SVM in the case of one step ahead stock movement prediction [25]. In general, ML methods or a combination of ML and statistical methods result in better prediction performance compared to traditional statistical methods [19], [26], [27].

Most studies consider stock movement prediction as a regression or classification task and focus on individual stock

or index. It is a common practice to predict the next day closing price of a stock or value of an index in a regression problem setting [12], [13], [25]. It is also possible to predict the stock return of the next period as the outcome of the regression task [28]. When stock movement prediction is considered as a classification task, the most common output is one day ahead up-down direction of stock price or stock index [4], [5], [7], [27]. Researchers calculate the stock movement direction by comparing the close price of two consecutive days [5] or other price values such as the open price of two consecutive days [7] or the high price [24].

Researchers are also focusing on stock movement prediction from other perspectives in addition to regression and classification. One such perspective is stock ranking prediction, where the considered stocks are ranked based on the predicted returns [9]–[11], [15]. This ranking task is not concerned about the actual performance, such as directional accuracy for classification or mean squared error for regression. The ranking task focuses on the relative order of the stocks, and that order is constructed based on the predicted returns. This predicted rank can be helpful while selecting stocks for portfolio construction. Investors can identify the top-ranked stocks for long-only portfolios and bottom-ranked stocks for short-only portfolios. Moreover, minimizing traditional evaluation measures such as mean squared error may not always result in optimum stock selection [10]. Thus, stock ranking prediction can be more beneficial from a pragmatic investment perspective.

The performance of any ML model is highly influenced by the choice of the input feature. Researchers use lagged values of different prices, volumes, and technical indicators as the input feature traditionally [5], [27], [29]. ML models such as ANN, SVM, and decision tree (DT) perform well when the input features come from disparate data sources such as market data, technical indicators, Wikipedia traffic, and Google news [7]. It is common to use different macroeconomic indicators as input features such as export amount, national monetary supply, the interest rate of bonds, and foreign exchange rate [4], [30]. Textual data such as online news [8] and Twitter feed [31] are used as the input features of ML models.

As stock ranking identifies the order of stocks, traditional input features such as price or technical indicators are not sufficient for this task. It is possible to use features based on investor sentiment to predict stock ranking [9]. Researchers use one-year close prices of stocks as the input features for stock ranking prediction [11]. Company financial status-related features and accounting ratios are used in addition to price, volume, and technical indicators for stock ranking prediction [32]–[34]. It is possible to derive stock intrinsic properties from mutual fund holdings and combine that with the individual stock properties, such as correlation with the market trend. This combination can be used as the input features for stock ranking prediction [35]. However, stock ranking performance improves by using stock relations as the input feature of the ML model [10], [36]. Stock relations are presented as a stock market graph, and the adjacency matrix of that graph is used as input features of subsequent ML models [10], [36].

While predicting the stock ranking using ML models, it is necessary to use a suitable loss function as a guide or performance measure. The loss function is optimized to improve the prediction performance of the ML model. Researchers use different loss functions for stock ranking prediction. Stock ranking based on predicted returns can be considered as a learning-to-rank task. The loss functions for learning-to-rank can be classified broadly into three categories: the point-wise loss, the pair-wise loss, and the list-wise loss [37]. In the point-wise approach, the loss function is defined based on an individual object, such as regression loss [37]. In the pair-wise approach, the loss function is defined using each pair of objects. One example is the pair-wise ranking-aware loss, where loss is zero if actual and predicted rank orders are the same and non-zero if they are different [10]. In the list-wise approach, ranked lists are considered as the instances in learning rather than individual elements or pairs [38].

References [9] and [11] use cross-entropy as the loss function. However, they differ in the approach of converting the scores into a probability distribution for cross-entropy calculation. Reference [11] uses the probability for each stock to outperform the cross-sectional median return. On the other hand, [9] uses two approaches to convert scores into probability measures. One approach uses the logistic function of scores according to Ranknet [39]. Another approach is based on the top k probability of Listnet [38]. Reference [10] combines both point-wise regression loss and pair-wise max-margin loss to formulate the loss function. The regression loss is minimized to reduce the difference between predicted return and actual return. The pair-wise max-margin loss is minimized to ensure that each stock pair has the same relative predicted order as the actual order [10].

Studies vary considerably in terms of the performance measure for stock ranking prediction. Reference [11] uses the accuracy of outperforming cross-sectional median return as the performance measure. Reference [10] uses MRRT as the performance measure, which reflects the performance of topmost stock prediction only. It is common to use NDCG or NDCG@k as a performance measure where k represents the top-k stocks. A similar measure for stock ranking prediction is the mean absolute precision (MAP) or MAP@k [35]. Researchers also use accuracy and precision of outperforming benchmark return by the annualized ranking model return while applying a trading strategy based on stock ranking prediction [15]. Studies simulate various trading strategies, and investment returns based on those trading strategies are used as an evaluation measure [9]–[11], [34], [36]. Table 1 represents a summary of the evaluation measures used by existing studies.

Node embedding can help to represent a graph from a high dimensional space of the adjacency matrix to a low dimensional space. Node embedding tries to learn a mapping

TABLE 1. Evaluation measures used by existing studies for stock ranking prediction.

Evaluation Measure	Reference
Accuracy	[11]
Mean squared error	[10], [36]
Mean reciprocal rank	[10], [35], [36]
Mean average precision	[35]
Normalized discounted cumulative gain	[9]
Investment return	[9]–[11], [15], [33], [34], [36]

from a high dimensional space to a low dimensional vector space so that the geometrical relationships in embedding space reflect the graph position and the structure of the local graph neighborhood of nodes in the original graph [40]. The learned low dimensional embedding can be used as the input feature for subsequent ML tasks such as stock ranking prediction. Node embedding techniques can be classified broadly into three categories: matrix factorization-based approaches, random walk-based approaches, and deep learning-based approaches [41].

Matrix factorization-based approaches try to factorize the similarity matrix between the nodes to obtain the low dimensional embedding. The similarity matrix can be constructed using different variants such as the adjacency matrix or the Laplacian matrix. Factorization-based approaches try to minimize a loss function which is formulated based on the assumption of the graph. For example, in local linear embedding (LLE), node embedding is assumed to be the linear combination of the embedding of neighboring nodes [42]. Thus, the loss between the embedding of a node and the linear combination using the embedding of its corresponding neighboring nodes is minimized. In graph factorization (GF), the distance between two nodes is calculated as the inner product in embedding space. The corresponding entry of the adjacency matrix is considered as the distance in the original space. Then embedding is constructed by minimizing the l_2 norm of loss between these distances along with a regularization term [43].

Two major random walk-based approaches are Node2Vec [18] and DeepWalk [44]. These approaches try to optimize node embedding so that the nodes with similar embedding tend to co-occur in the random walks over the graphs. These approaches try to minimize the cross-entropy, where the probability is calculated based on the samples of a random walk. Node2Vec and DeepWalk vary in terms of probability calculation methods as well as how they define the random walk strategy (e.g., biased or unbiased) [40].

III. PRELIMINARIES

This section discusses the relevant theoretical concepts which are employed in this study.

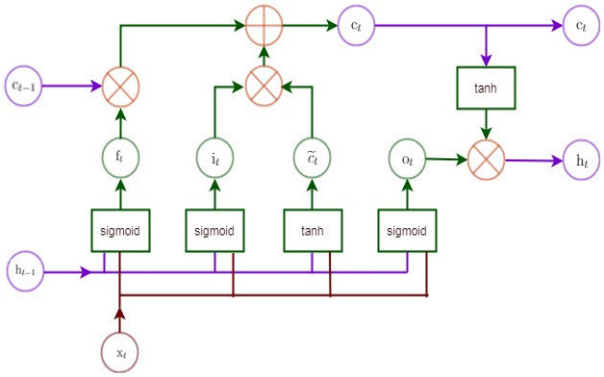


FIGURE 1. LSTM cell.

TABLE 2. Weight matrix of a LSTM cell.

Gate/State	Weight of x_t	Weight of h_{t-1}	Bias	Activation Function	Output
Forget	$W_{f,x}$	$W_{f,h}$	b_f	sigmoid	f_t
Input	$W_{i,x}$	$W_{i,h}$	b_i	sigmoid	i_t
Output	$W_{o,x}$	$W_{o,h}$	b_o	sigmoid	o_t
Candidate Cell State	$W_{c,x}$	$W_{c,h}$	b_c	tanh	$\widetilde{c}_t$

A. LSTM NETWORK

LSTM is proposed in [45] to capture temporal dependencies and to overcome the barrier of vanishing or exploding gradient. Fig. 1 shows the structure of a single LSTM unit indicating information flow and mathematical operations.

A brief description of the symbols of Fig. 1 is as follows:

- Input vector at time t: x_t
- Candidate cell state at time t: $\widetilde{c}_t$
- Cell state at time t: c_t
- Output at time t: h_t
- Output of the input gate at time t: i_t
- Output of the forget gate at time t: f_t
- Output of the output gate at time t: o_t

A single LSTM cell has three gates. They control the information flow inside the cell. A brief description of these three gates is as follows:

- Input gate: It determines the amount of information added to the cell state.
- Forget gate: It determines which information to remove from the cell state.
- Output gate: It determines which information of the cell state will be used for calculating the final output of the LSTM unit.

Each of the three gates and the calculation of candidate cell state uses x_t and h_{t-1} as inputs. These four calculations use weights, biases, and different activation functions, which are depicted in Table 2. The equations for calculating the output of three gates and candidate cell state are as follows:

$$f_t = \sigma(W_{f,x}x_t + W_{f,h}h_{t-1} + b_f), \quad (1)$$

$$i_t = \sigma(W_{i,x}x_t + W_{i,h}h_{t-1} + b_i), \quad (2)$$

$$o_t = \sigma(W_{o,x}x_t + W_{o,h}h_{t-1} + b_o), \quad (3)$$

$$\widetilde{c}_t = tanh(W_{c,x}x_t + W_{c,h}h_{t-1} + b_c). \quad (4)$$

Here, σ is the sigmoid function. After calculating the output of different gates and candidate cell state, cell state at time t, c_t is calculated as follows:

$$c_t = i_t \otimes \widetilde{c}_t + f_t \otimes c_{t-1}. \quad (5)$$

Here, $\otimes$ denotes element-wise product. i_t will determine what portion of candidate cell state $\widetilde{c}_t$ will be added to the current cell state, and f_t will determine what portion of previous cell state c_{t-1} will be added to the current cell state. The calculation of the final output at time t is as follows:

$$h_t = o_t \otimes \tanh(c_t). \quad (6)$$

Here, o_t will determine what portion of cell state c_t will be transferred to output h_t.

B. Node2Vec

Node2Vec is an embedding technique for learning continuous features representation of nodes in a network [18]. A low-dimensional mapping of nodes is learned by maximizing the likelihood of preserving the network neighborhoods of nodes. Let us consider a graph $G = (V, E)$, consisting of nodes V and edges E. The target of any node embedding technique is to find out a mapping function F, which can transform the nodes V to a lower-dimensional representation $\mathbb{R}^v$, given that $v \ll |V|$. Thus, F can be considered as a matrix of size $|V| \times v$ [18].

To estimate the function F, it is required to find out the network neighborhood $N_S(u)$ for every node $u \epsilon V$ in the network through a searching strategy S. This list of neighbors can be considered as a sentence where each node is a word. All the sampled sentences formulate a corpus. The next step is to define an objective function and optimize that in the process of estimating F from this corpus of neighborhood sentences. Node2Vec applies the idea of Word2Vec algorithm [46] to convert the sampled sentences to an embedding vector and uses log probability as the objective function.

Node2Vec deploys a random walk-based searching strategy to find out the network neighborhood $N_S(u)$. This random walk-based strategy allows sampling neighboring nodes with different structures by combining both breadth-first sampling (BFS) and depth-first sampling (DFS) strategies [18]. In pure BFS, N_S contains mostly the immediate neighbors such as first-order or second-order. Sampling using a pure BFS strategy leads to an embedding that preserves the structural equivalence [18]. In this case, nodes with a similar structural role, such as hubs, remain closer in the embedding space. In pure DFS, N_S contains neighbors with increasing distance from the source node. In this case, N_S portrays a macro view of the network, where the nodes from the same community remain closer in the embedding space [18].

In Node2Vec, the structure of sampled neighbors (N_S) is controlled by two parameters: return parameter (p) and in-out parameter (q). The likelihood of immediately revisiting a node in the random walk searching strategy is controlled by p [18]. The parameter q is used to distinguish between inward and outward nodes [18]. Lower values of p compared to q result in a sampling of neighbors from the local structure and thus, simulates the BFS strategy. Higher values of p encourage to travel away from an already visited node and suitable for DFS strategy. On the other hand, if $q > 1$, S would preserve the local view of the neighborhood and would emulate the BFS strategy. If $q < 1$, S would sample more nodes away from the source and would emulate the DFS strategy. Controlling the values of p and q would help to sample different neighbor combinations, and the optimal (p, q) can be chosen through validation results.

Once the neighboring nodes are sampled in N_S, the following log-probability is optimized to find out the mapping function F [18]:

$$\max_F \sum_{u \epsilon V} \log[\Pr(N_S(u)|F(u))]. \quad (7)$$

Node2Vec assumes that the likelihoods of observing the neighboring nodes are independent of each other [18], and thus, the objective function can be represented as follows:

$$\max_F \sum_{u \epsilon V} \log[\prod_{\eta_i \epsilon N_S(u)} \Pr(\eta_i|F(u))]. \quad (8)$$

The conditional likelihood of every source-neighborhood node pair is modeled as a softmax function and defined using the dot product of their corresponding features in embedding space.

$$\Pr(\eta_i|F(u)) = \frac{\exp(F(\eta_i).F(u))}{\sum_{\mu \epsilon V} \exp(F(\mu).F(u))}. \quad (9)$$

We can get the final objective function of the Node2Vec algorithm by combing the equation (8) and (9).

$$\max_F \sum_{u \epsilon V} [-\log(\sum_{\mu \epsilon V} \exp(F(\mu).F(u))) + \sum_{\eta_i \epsilon N_S(u)} \exp(F(\eta_i).F(u))]. \quad (10)$$

This objective function is optimized using a stochastic gradient ascent algorithm. Apart from p and q, few other hyperparameters can be varied to improve the performance of Node2Vec. The first one is the length of the random walk l, which controls the number of sampled neighbors in each random walk. It is also possible to vary the number of random walks (n) to explore different $N_S(u)$.

C. THE LIST-WISE LOSS

The list-wise loss is proposed in [38]. We will explain the list-wise loss from the perspective of stock ranking prediction. Let us assume that we want to rank N stocks based on their predicted return $\hat{r}$. The actual return or ground truth is r. We want to minimize the list-wise loss function L over the entire training period.

$$\sum_{\tau=1}^{T} L(r^\tau, \hat{r}^\tau). \quad (11)$$

Here, T is the length of the training period. In the list-wise loss, r and $\hat{r}$ are converted into probability distributions, and then metrics such as cross-entropy or KL divergence are used to generate a loss value. We can use the top k probability for this purpose [38]. If we have N stocks, we can formulate $N!$ different permutations from them. At the same time, we have the predicted return $\hat{r}_i$ for each stock i. Any permutation is possible from available $N!$ permutations for any predicted score $\hat{r}$. However, different permutations will have different probabilities. If we define π as a permutation of N stocks and $\phi()$ as a strictly positive and increasing function, then permutation probability can be defined as follows [38]:

$$\Pr(\pi) = \prod_{i=1}^{N} \frac{\phi(\hat{r}_{\pi(i)})}{\sum_{j=1}^{N} \phi(\hat{r}_{\pi(j)})}. \tag{12}$$

Here, $\hat{r}_{\pi(i)}$ denotes the predicted return of stock at the position i of permutation π. We can have $N!$ such permutation probabilities for N stocks. The top k probability of stocks $(\psi_1, \psi_2, \ldots.\psi_k)$ means the probability of their being ranked at the top k positions. We can define the top k subgroup of permutations as the set of permutations where the top k stocks are exactly $(\psi_1, \psi_2, \ldots.\psi_k)$. Let us denote this top k subgroup as $\Omega_k(\psi_1, \psi_2, \ldots.\psi_k)$. The top k probability of objects $(\psi_1, \psi_2, \ldots.\psi_k)$ is the sum of all permutation probabilities which are in $\Omega_k(\psi_1, \psi_2, \ldots.\psi_k)$ and can be defined as follows [38]:

$$\Pr(\Omega_k(\psi_1, \psi_2, \ldots.\psi_k)) = \sum_{\pi \epsilon \Omega_k} \Pr(\pi). \tag{13}$$

According to [38], top k probability can be calculated efficiently as follows:

$$\Pr(\Omega_k(\psi_1, \psi_2, \ldots.\psi_k)) = \prod_{i=1}^{k} \frac{\phi(\hat{r}_{\psi i})}{\sum_{j=1}^{N} \phi(\hat{r}_{\psi j})}. \tag{14}$$

For different Ω_k, we can calculate $\Pr(\Omega_k)$ and thus, can formulate a probability distribution. If we have r and $\hat{r}$, then we can calculate the list-wise loss as follows using cross-entropy:

$$L(r^{\tau}, \hat{r}^{\tau}) = -\sum_{\forall s \epsilon \Omega_k} \Pr_{r^{\tau}}(s) \log(\Pr_{\hat{r}^{\tau}}(s)). \tag{15}$$

D. RANK BIASED OVERLAP (RBO)

Rank biased overlap (RBO) is a similarity measure for comparing two infinite lists [16]. RBO gives more weight on a higher rank than a lower rank, and thus, suitable as a top-weighted measure. RBO ensures top-weightedness by using a decreasing and convergent weight series. It can also compare the list with different lengths and having different constituent members.

Let us consider the actual rank as a list B and the predicted rank as a list P. For stock ranking prediction, both B and P have the same length and constituent stocks if the list consists of all the stocks of the market. However, if we want to predict the top k stocks, then B and P have the same length but not the same constituent stocks unless the predicted ranks are 100% correct.

Let us consider $B_{:d}$ as the actual rank and $P_{:d}$ as the predicted rank up to depth d. The overlap up to depth d can be considered as the size of the intersection between $B_{:d}$ and $P_{:d}$. We can define the agreement at depth d, AG_d as the overlapped proportion of $B_{:d}$ and $P_{:d}$:

$$AG_d = \frac{|B_{:d} \cap P_{:d}|}{d}. \tag{16}$$

RBO considers B and P as infinite lists and is defined as an infinite weighted sum of agreement, AG [16]:

$$RBO(B, P, w) = \sum_{d=1}^{\infty} w_d . AG_d. \tag{17}$$

As stated earlier, the weight series is an infinite one, and for *RBO*, it is defined as a geometric progression using a probability parameter, ρ. The infinite sum of that geometric series is defined as follows:

$$\sum_{d=1}^{\infty} \rho^{d-1} = \frac{1}{1-\rho}. \tag{18}$$

To ensure that the sum of the weight series is 1, w_d is defined as $(1-\rho)\rho^{d-1}$. So, the final definition of *RBO* is as follows [16]:

$$RBO(B, P, \rho) = (1-\rho) \sum_{d=1}^{\infty} \rho^{d-1} AG_d. \tag{19}$$

The value of ρ determines the top-weightedness in the evaluation measures. Smaller values of ρ result in more top-weightedness in the evaluation measure. An extreme case is $\rho = 0$ when only the top-ranked stock is considered.

E. SUMMARY STATISTICS OF A GRAPH

Different summary statistics are used to analyze the characteristics of a graph. The first such statistic is the average node degree $(\bar{g})$, which is a measure of connectivity in the graph. If there are $|V|$ nodes in the network, then $\bar{g}$ can be defined as follows, where g is the degree of individual node:

$$\bar{g} = \frac{1}{|V|} \sum_{i=1}^{|V|} g_i. \tag{20}$$

Node degree g is the number of neighbors adjacent to a node. If node i is connected to 4 adjacent neighbors, then the degree g_i will be 4.

The second statistic is the average clustering coefficient (ζ). It can be defined as the portion of neighbors that are connected. For each node i in an undirected and unweighted graph, ζ_i can be defined as follows:

$$\zeta_i = \frac{2\lambda_i}{g_i(g_i - 1)}. \tag{21}$$

λ_i is the number of edges between the neighbors of node i, and g_i is the degree of node i. the average clustering

coefficient of a network or graph is defined as follows:

$$\zeta = \frac{1}{|V|}\sum_{i=1}^{|V|}\zeta_i. \tag{22}$$

We can use network density (D) as the third statistic. It measures the portion of possible connections in a network and defined as follows for an undirected graph:

$$D = \frac{2(\text{Total number of edges})}{|V|(|V|-1)}. \tag{23}$$

IV. THE RATIONALE OF A NEW PERFORMANCE EVALUATION MEASURE FOR STOCK RANKING

In Sections I and II, we have identified several existing evaluation measures for stock ranking performance prediction. However, each of those evaluation measures has its shortcomings. MRRT focuses on the topmost stock only [10]. MRRT will not change even if all the subsequent predictions from the second rank onward are entirely incorrect. It is not suitable for a practical investment strategy. There will be a significant change in the cumulative investment return if the top two stocks are swapped [10]. From an investment perspective, it is always necessary to invest in more than one stock for portfolio diversification [47], [48]. Most stock ranking studies focus on investment in more than one stock. Reference [9] takes a long position for the top 25% stocks and a short position for the bottom 25% stocks. Reference [11] uses top-k stocks for a long position and bottom-k stocks for a short position. If MRRT is used, then the investor might not get any idea about the performance of the second best or the third best stock. As a result, the investment performance will not be optimal if stocks are selected based on MRRT. Thus, it is worthwhile to propose an evaluation measure that focuses on the top-k stocks rather than the topmost stock only.

It is possible to use evaluation measures such as accuracy of outperforming cross-sectional median return or accuracy of outperforming market return [11]. However, none of these two evaluation measures has top-weightedness. Accuracy gives equal weight to all the stocks by definition. It does not differentiate between the first stock and the second stock, and so on. As a result, the investors cannot get any idea about the top-performing stocks using accuracy. We can consider a hypothetical scenario to demonstrate the limitation of accuracy as an evaluation measure for stock ranking prediction. Let us assume that we want to compare two methods for predicting the ranks of 100 stocks. The first method correctly predicts all the ranks except the 1^{st} and the 52^{nd} stock. Their positions are interchanged in the predicted rankings. The second method correctly predicts all the ranks except the 51^{st} and the 52^{nd} stock. The first method is worse as it fails to identify the topmost profitable stock for a long position investment strategy. However, the accuracy of both methods will be the same, and we will not be able to identify the best method.

The NDCG ensures top-weightedness by applying a discounting factor such as $\log_{base} k$ to the cumulative gain, where k is the rank of the stock [49]. We can divide the cumulative gain of each rank by the discounting factor to get the discounted cumulative gain (DCG). One shortcoming of this measure is that we cannot discount the ranks which are less than the *base* [49]. It will boost those ranks rather than discounting them. For example, we can never discount the first position as the $\log_{base} 1$ is 0 always. If we use $base = 10$, we cannot apply any discounting to the first ten ranks. Moreover, to normalize DCG, it is required to define the ideal gain value for each rank [9], [49]. However, the ideal gain values are not automatic. They are user-defined and can be the same for top k stocks. For example, the first k_1 positions can be considered highly relevant, and the user can give them an ideal gain value of 3. The ideal gain value can be 2 for the subsequent k_2 positions, 1 for the following k_3 positions, and 0 for the rest. The ideal gain vector will vary greatly depending on k_1, k_2, k_3, and the associated ideal gain values. This type of manual intervention makes NDCG less robust. On the other hand, *RBO* has only one hyperparameter, ρ. We can first decide the total weight of top-k stocks and select the corresponding value of ρ. For example, if we set $\rho = 0.9$, that will give 86% of the weight in the similarity comparison to the top ten stocks [16]. Thus, *RBO* is more flexible and requires less manual intervention than NDCG in a stock ranking task.

Average precision is defined as the mean of the precision for each position of a ranked list [35]. MAP is the mean of average precision across multiple ranks [35]. We can calculate average precision for all the trading days and then take the average to get MAP. MAP is not top-weighted by definition and thus, not an optimal choice for stock ranking prediction.

MSE calculates the mean of the squared difference between the ground truth and the predicted return. It does not use the predicted ranks in the calculation. Thus, it is not a direct measure of ranking performance. In a ranking task, we are concerned about the predicted ranks, not about the underlying scores which are used to create the ranks [9]. Thus, MSE can be used as a loss function but not as an evaluation measure for stock ranking prediction. Moreover, MSE is not top-weighted, and it does not differentiate between the error of the top stocks and the bottom stocks. It is also suboptimal as an evaluation measure from the top-weightedness perspective.

The above discussion identifies the limitations of existing evaluation measures for stock ranking prediction performance evaluation, and thus, it is required to define a more robust evaluation measure.

V. METHODOLOGY

A. TEMPORAL EMBEDDING LAYER

Historical price data and technical indicators of stocks are considered as influential input features for stock movement prediction [5], [11], [27]. Different technical indicators can signal about the future stock trend. Those indicators are used widely when stock movement prediction is considered as a classification or regression task [5], [27]. These technical

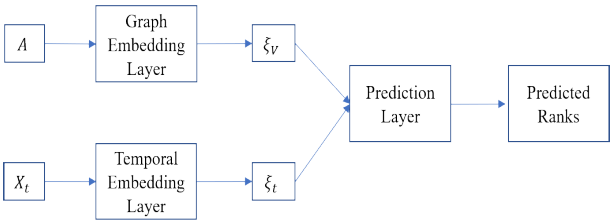


FIGURE 2. Overall network architecture.

indicators are hand-crafted features. It is also possible to automatically extract features or embedding using variants of recurrent neural network (RNN), such as LSTM [10], [11], [17], and use that extracted features or embedding in subsequent layers for stock movement prediction.

We use the LSTM network to extract the temporal embedding of the stocks rather than using hand-crafted technical indicators. LSTM network can capture long-term dependencies, and their incorporation results in improved stock movement prediction performance [11], [12]. The historical price time series data of stocks at time step t (X_t) is used as the input of the LSTM network. We apply min-max normalization to X_t before using it as the input of the LSTM network. The last hidden state of the LSTM network is taken as the temporal embedding of stocks at time step t. We use all the stocks together as the input of the LSTM network and minimize the combined point-wise and pair-wise loss to extract the temporal embedding according to [10]. We will denote this temporal embedding as ξ_t of shape $\mathbb{R}^{N\times z}$. We can consider z as the dimension of the temporal embedding for a single stock.

B. GRAPH EMBEDDING LAYER

In this layer, we use the stock market graph as the input and extract node embedding from that graph using the Node2Vec algorithm. Let us consider the stock market graph as a static multi-graph with different types of relations between two constituent stocks. If there are M types of relations between the stocks, the adjacency matrix (A) will be a tensor of shape $\mathbb{R}^{N\times N\times M}$. We first convert A from a three-dimensional tensor to a two-dimensional matrix. The two-dimensional A has value 1 in a position if two corresponding stocks have at least one relation between them. We will apply Node2Vec on the new A to extract node embedding (ξ_V) of shape $\mathbb{R}^{N\times v}$ and use that as features in the prediction task.

C. PREDICTION LAYER

After extracting features from the temporal embedding layer and graph embedding layer, we will use them as input to a fully connected layer. We will concatenate ξ_t and ξ_V to formulate the input for the prediction layer. The concatenated input will be of shape $\mathbb{R}^{N\times(v+z)}$. This fully connected layer will predict the return of the stocks, and that will be used to formulate the predicted ranking.

The overall network architecture is presented in Fig. 2.

D. EVALUATION MEASURES

1) NORMALIZED RANK BIASED OVERLAP (NRBO)

For stock ranking prediction, we need to find an evaluation measure that should satisfy at least the following conditions.

1) It should give more weight to the top-ranked stocks
2) The maximum value of the measure should be 1, when 100% predicted ranks are correct
3) The minimum value of the measure should be 0, when 0% predicted ranks are correct

The first condition is necessary due to the nature of investment in the stock market. An investor is more likely to invest in the top-ranked stocks. That is why, the evaluation measure should focus more on top-ranked stocks. The second condition determines the maximum value, and the third condition determines the minimum value of the evaluation measure. These two conditions are necessary to have a bounded measure. These bounds should be the same every time, irrespective of the number of ranked stocks.

RBO is an overlap based measure where a convergent series of weights is used to bias the proportional overlap [16]. The convergent weight series has higher weights for overlap for the top ranks and lower weights for the bottom ranks. Thus, it satisfies the first condition for the evaluation measure of stock ranking. In this study, we measure the performance of predicting top k stocks.

As stated in Section III-D, *RBO* is defined for an infinite list. However, for stock ranking prediction, the major focus is on top k or bottom k stocks. Investors can take a long position for top k stocks or a short position for bottom k stocks. We can calculate *RBO* for top k stocks as follows:

$$RBO@k = (1-\rho)\sum_{d=1}^{k}\rho^{d-1}AG_d \tag{24}$$

However, the definition of $RBO@k$ according to the equation (24) does not satisfy the second condition of evaluation measure for stock ranking. For example, if we set $\rho = 0.8$, then $RBO@5$ will be 0.67 even if all the predicted ranks are correct. Thus, we need to modify the definition of $RBO@k$ to ensure that it satisfies the second condition of an evaluation measure. We propose a new measure named normalized rank biased overlap (NRBO) to apply the concept of *RBO* for finite lists of stock ranking. This proposed measure will have a value between 0 to 1 for any arbitrary top k stocks. Our proposed definition for $NRBO@k$ is as follows:

$$NRBO@k = \sum_{d=1}^{k}\frac{\rho^{d-1}}{\sum_{j=1}^{k}\rho^{j-1}}AG_d \tag{25}$$

The definition of equation (25) ensures that $NRBO@k$ satisfies all three conditions of evaluation measure for stock ranking prediction. For this study, we are using $NRBO@10$ as the evaluation measure.

2) TRAINING TIME

The stock market graph can be quite large due to the presence of multiple relation types. If a fully connected neural

network such as temporal graph convolution (TGC) [10] is used, it might take a significantly longer time to complete the training process. If a node embedding technique such as Node2Vec is applied, this training time can be reduced significantly. Thus, we use the median training time per epoch as an evaluation measure.

3) MEAN RECIPROCAL RANK OF TOP STOCK (MRRT)

The mean reciprocal rank of top stock (*MRRT*) is calculated by taking the average reciprocal predicted rank of the topmost stock. If the topmost stock is predicted closer to the top rank, *MRRT* will be higher. It will be lower if the actual topmost stock is predicted at the bottom of the ranks. We are using *MRRT* as an evaluation measure to portray the prediction performance for the topmost stock as our baseline model in [10] uses this measure.

4) CUMULATIVE INVESTMENT RETURN RATIO (IRR)

We use the cumulative investment return ratio (IRR) as a performance measure following [10] and [11]. *NRBO@k* indicates whether the model can predict the top-ranked stocks properly or not. We use IRR to show that using *NRBO@k* as a performance measure can lead to better investment return compared to using *MRRT*.

Following [10], we simulate a daily buy-hold-sell trading strategy. The trading strategy is simulated as follows:

- Trading day t: The trader buys the top k stocks according to the predicted rank. The trader allocates an equal amount of funds for each stock.
- Trading day $t + 1$: The trader sells the stocks purchased at day t.

Reference [10] trades only a single stock. However, we trade top k stocks. It is always better to trade top k stocks than the topmost stock from an investment perspective. The main reason behind this is the topmost stock may not always be sufficiently liquid such that the buying order gets filled at the closing price of day t, and the selling price is the closing price of day $t + 1$. We also ignore transaction costs following [10].

VI. EXPERIMENTAL SETTING

For temporal embedding, we use optimal embedding according to [10]. In this case, the embedding dimension z for NASDAQ is 64 and for NYSE is 32. LSTM networks consisting of 64 LSTM cells for NASDAQ and 32 LSTM cells for NYSE are used to generate these embeddings. The sequence length or the look-back period of the input for the LSTM network is 16 for NASDAQ and 8 for NYSE.

To demonstrate the superiority of the list-wise loss function, we run the experiments twice. Firstly, with the list-wise loss and then with the combined point-wise and pair-wise loss, which is used in [10]. The combined point-wise and pair-wise loss is our baseline loss. We will refer this loss to point pair loss in several places for brevity. We also run experiments with and without Node2Vec. As a result, for each stock market and relation graph pair, we run four experiments. When Node2Vec is not used, we use a fully connected temporal graph convolution (TGC) layer to generate relational embedding according to [10]. Thus for each stock market and relation graph combination (e.g. [NYSE, Industry]), we have four embedding technique and loss function combinations (e.g. [Node2Vec, list-wise]). We consider the model with a fully connected TGC layer and the combined point-wise and pair-wise loss as the baseline model. The TGC layer has 43 neurons for NASDAQ with Wikidata graph and 97 neurons for NASDAQ with Industry graph. For NYSE, the number of neurons is 33 with Wikidata graph and 108 with Industry graph.

In the TGC method, temporal embedding extracted from return time-series data is combined again (through a matrix multiplication) with node embedding data to generate temporal relational embedding [10]. However, we are not using that technique in our proposed method when applying embedding from Node2Vec. The size of the fully connected prediction layer is $(v + z)$. This size is 128 for NASDAQ with TGC embedding and 64 for NYSE with TGC embedding.

We use Adam optimizer [50] with a learning rate of 0.001, a decay rate for the first momentum estimates of 0.9, and a decay rate for the second momentum estimates of 0.999. We use the leaky rectifier as the activation function for the TGC layer and the prediction layer, according to [10]. Glorot uniform initializer [51] is used to initialize the weights of the layers. We run the training process for three different epochs: 5000, 10000, and 15000. Generally, the model performance should improve with increased epochs.

To optimize the graph embedding performance, we vary different hyperparameters of the Node2Vec algorithm. Initially, we apply Node2Vec with a fixed hyperparameter setting. Then we apply hyperparameter tuning of Node2Vec if the result is worse than the baseline result. We use $p = 0.50$, $q = 2$, $l = 8$ and $n = 10$ as the default hyperparameters for Node2Vec. We use $v = 64$ for NASDAQ and $v = 32$ for NYSE as the default setting. The fully connected prediction layer has a size of 128 for NASDAQ with Node2Vec and 64 for NYSE with Node2Vec. This size changes according to v when we tune the hyperparameters of Node2Vec.

As we are using a train-validation-test split, it is also important to decide the model selection criterion. As mentioned in section V-D, we are using *NRBO@*10 and *MRRT* as the main evaluation measures. For each experimental setting, we select two models as the best models. One model is selected based on the best validation *NRBO@*10, and the other is selected based on the best validation *MRRT*. Then we are using these two models to calculate the test set *NRBO@*10 and *MRRT*.

VII. EMPIRICAL RESULTS

A. PRICE DATA

We have used stocks from New York Stock Exchange (NYSE) and NASDAQ stock exchange. The data has been collected from https://github.com/fulifeng/Temporal_Relational

_Stock_Ranking, which is prepared by [10]. The price data contains daily transactions from 01/02/2013 to 12/08/2017 for 1026 NASDAQ stocks and 1737 NYSE stocks. As the target is to calculate stock ranking based on predicted return, the one-day return is calculated as follows using the close price:

$$r^t = \frac{\text{Close}^t - \text{Close}^{t-1}}{\text{Close}^t}. \quad (26)$$

The price and return data is divided into three parts: training set (01/02/2013 − 12/31/2015, 756 trading days), validation set (01/04/2016−12/30/2016, 252 trading days), and testing set (01/03/2017 − 12/08/2017, 237 trading days).

B. STOCK MARKET RELATION GRAPH DATA

Two types of static stock market graphs are formulated according to [10]. The first type is based on whether a stock pair comes from the same industry or not. If they are from the same industry, then there will be a relation between them. For example, Google and Facebook belong to the Computer Software industry, and thus, there is a relation between them. All such industry relations are extracted from the official company list of NASDAQ,[1] and two stocks can have more than one industry relation [10]. As a result, the adjacency matrix will be a tensor based on the unique industry types. There are 108 unique industry relations between the stocks of NYSE and 97 unique industry relations between the stocks of NASDAQ. We refer to the appendix of [10] for details of industry relations.

The second type of relation is extracted based on the first-order and the second-order relations in the statements of Wikidata [10]. There is a first-order relation between two stocks if one of them is the subject and the other is the object of a statement in Wikidata.[2] Two stocks have a second-order relation if both have a common object in two different statements [10]. After analysis and cleaning of data, 43 types of Wikidata relations are created for NASDAQ stocks, and 33 types of relations are created for NYSE stocks. We refer to the appendix of [10] for details of Wikidata relations.

We calculate different statistics to analyze the stock market graphs as described in Section III-E. These statistics are average node degree ($\bar{g}$), average clustering coefficient (ζ), and network density (D). As the adjacency matrices are of dimension $\mathbb{R}^{N\times N\times M}$, we first convert them into a shape of $\mathbb{R}^{N\times N}$ and then calculate the above statistics. If there exists at least one relation out of M between two stock pairs, we consider that there is an edge between them in the converted graph. Thus, the final relation graph is an unweighted and undirected one. The above summary statistics are then calculated on the converted graphs and presented in Table 3.

The graphs formed based on Wikidata are very sparse with lower network density, average node degree, and clustering coefficient. On the other hand, graphs formed based on industry relations have relatively higher density and average node degree. The average clustering coefficients are almost close to 1, reflecting a densely connected graph.

[1]https://www.nasdaq.com/market-activity/stocks/screener
[2]https://doc.wikimedia.org/Wikibase/master/js/

TABLE 3. Summary statistics of stock market graphs.

Market	Relation	Density (%)	Average Node Degree	Average Clustering Coefficient
NASDAQ	Wikidata	0.38	1.95	0.102
NASDAQ	Industry	5.20	26.63	0.946
NYSE	Wikidata	0.40	3.44	0.128
NYSE	Industry	9.48	82.83	0.974

TABLE 4. Best *NRBO*@10 and *MRRT* performance.

Market	Relation	Loss function	Embedding Technique	Best NRBO@10	Best MRRT
NASDAQ	Wikidata	Point Pair Loss	TGC Embedding	0.358	**0.430**
			Node2Vec	0.355	0.427
		List-wise Loss	TGC Embedding	0.368	0.415
			Node2Vec	**0.368**	0.403
	Industry	Point Pair Loss	TGC Embedding	0.374	0.440
			Node2Vec	0.352	0.433
		List-wise Loss	TGC Embedding	**0.399**	**0.460**
			Node2Vec	0.369	0.400
NYSE	Wikidata	Point Pair Loss	TGC Embedding	**0.645**	0.702
			Node2Vec	0.637	0.696
		List-wise Loss	TGC Embedding	0.644	**0.716**
			Node2Vec	0.637	0.704
	Industry	Point Pair Loss	TGC Embedding	0.675	**0.755**
			Node2Vec	0.635	0.701
		List-wise Loss	TGC Embedding	**0.685**	0.730
			Node2Vec	0.642	0.733

The best performance for each stock market and relation graph combination is boldfaced.

C. NRBO PERFORMANCE

We present the *NRBO*@10 performance for four different experimental settings based on two different loss functions and two different embedding techniques in Fig. 3 to Fig. 6 and in Table 4.

The *NRBO*@10 performance for NASDAQ with Wikidata graph is presented in Fig. 3, and with Industry graph is presented in Fig. 4. In both cases, Node2Vec is applied with default parameter settings as described in section VI.

For NASDAQ with the Wikidata graph, the maximum *NRBO*@10 is 0.368 when the list-wise loss is applied

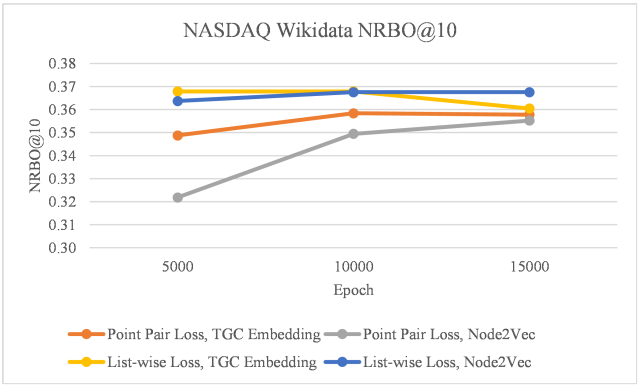


FIGURE 3. NRBO@10 of NASDAQ with Wikidata graph.

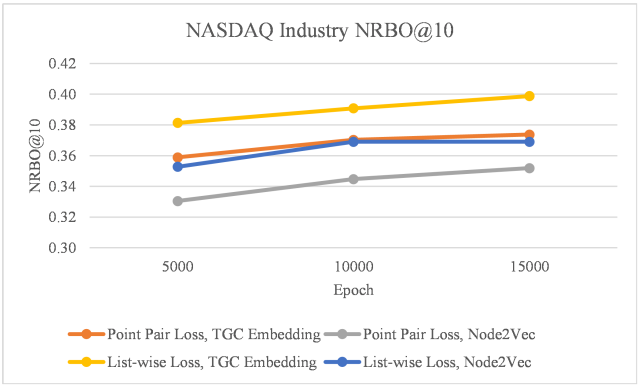


FIGURE 4. NRBO@10 of NASDAQ with Industry graph.

with Node2Vec, and the training is conducted up to 15000 epochs. It is a 2.65% relative gain compared to the baseline *NRBO*@10 of 0.358 obtained using TGC embedding and the combined point-wise and pair-wise loss. For NASDAQ with the Industry graph, maximum *NRBO*@10 is 0.399 when the list-wise loss is applied with TGC embedding and the training is conducted up to 15000 epochs. Here, the relative gain is 6.7% compared to the baseline performance of 0.374. Thus, for NASDAQ, incorporation of the list-wise loss results in better *NRBO*@10 performance. Node2Vec performance is also very close to the performance of the baseline model in the case of the Industry graph with NASDAQ. The highest *NRBO*@10 for Node2Vec is 0.369 when it is applied with the list-wise loss. It is only 1.25% lower compared to the baseline model.

For NYSE with the Wikidata graph, the best *NRBO*@10 is 0.645 with the baseline model. The list-wise loss with TGC embedding results in *NRBO*@10 of 0.644. Both models perform very close when the training is conducted up to 15000 epochs, as depicted in Fig. 5. However, we can observe significant improvement in *NRBO*@10 when the Industry graph is used, as observed in Fig. 6. The list-wise loss with TGC embedding results in a 1.47% relative gain (0.685) compared to the baseline *NRBO*@10 of 0.675. Node2Vec with default parameter setting does not perform well for NYSE compared to the baseline model.

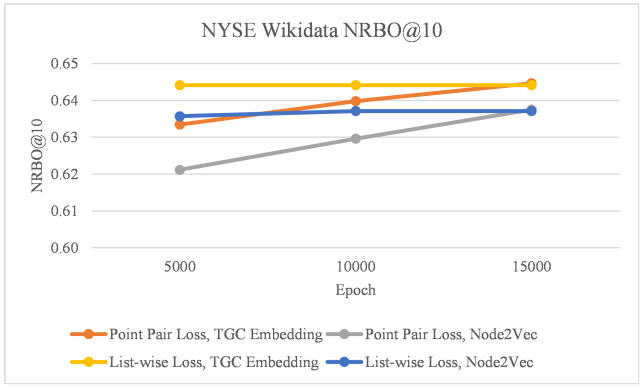


FIGURE 5. NRBO@10 of NYSE with Wikidata graph.

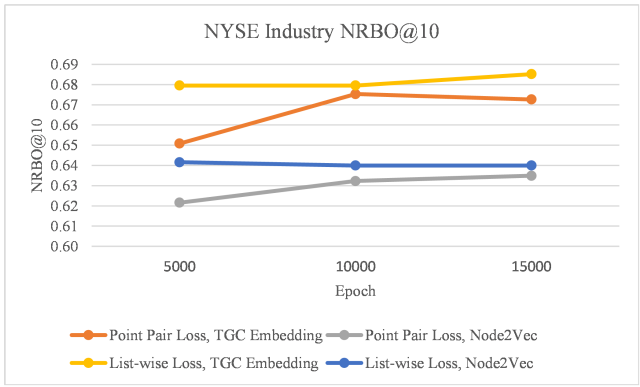


FIGURE 6. NRBO@10 of NYSE with Industry graph.

Overall, it can be concluded that the list-wise loss results in significantly better *NRBO*@10 compared to the combination of the point-wise and the pair-wise loss. Empirical results justify the use of the list-wise loss for stock ranking prediction. However, Node2Vec with default parameter setting does not perform better than the baseline for NYSE. We will analyze this issue further by tuning the hyperparameters of Node2Vec.

D. MRRT PERFORMANCE

We present the *MRRT* performance in Fig. 7 to Fig. 10 and Table 4. Fig. 7 and Fig. 8 show the *MRRT* performance for the NASDAQ stock exchange. When the Wikidata graph is used, the baseline model shows the best *MRRT* of 0.430. With the list-wise loss, the best *MRRT* is 0.415, which is lower compared to the baseline. However, we get a significantly improved *MRRT* of 0.460 when the Industry graph is used with the list-wise loss and TGC embedding. It again substantiates the usefulness of the list-wise loss even when predicting the topmost stock.

We represent the *MRRT* performance for NYSE in Fig. 9 and Fig. 10. When the Wikidata graph is used, the best *MRRT* of 0.716 is obtained using the list-wise loss and TGC embedding. This is a relative gain of 1.95% compared to the baseline of 0.702. However, when the Industry graph is applied, the baseline model performs the best with an *MRRT* value of 0.755. We get the second-best *MRRT* of 0.733 when we use the list-wise loss with Node2Vec embedding.

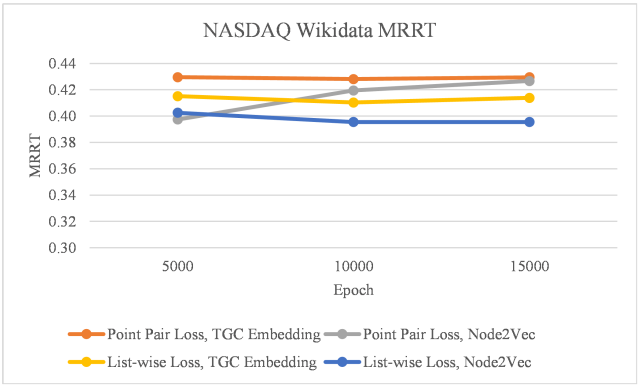


FIGURE 7. MRRT of NASDAQ with Wikidata graph.

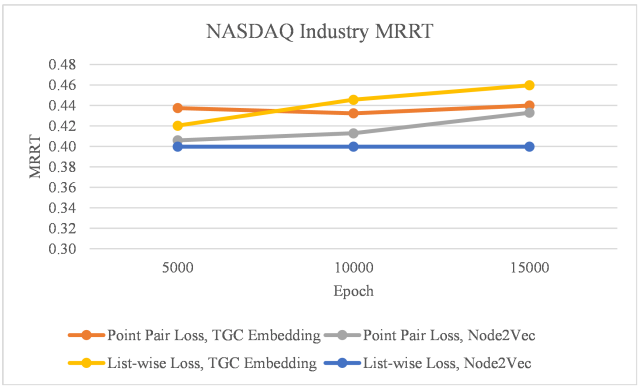


FIGURE 8. MRRT of NASDAQ with Industry graph.

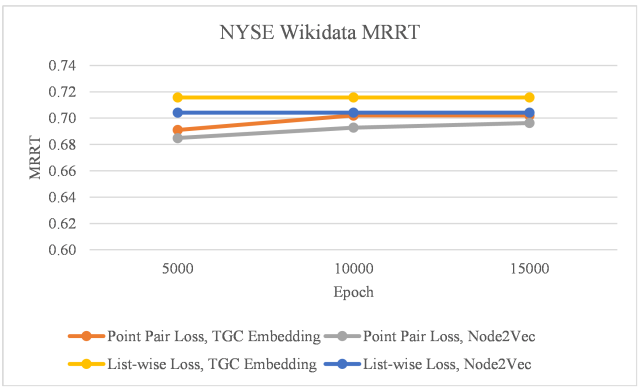


FIGURE 9. MRRT of NYSE with Wikidata graph.

So, the performance of the list-wise loss in terms of *MRRT* is not as evident as *NRBO*@10. The list-wise loss produces the best *MRRT* for NASDAQ, but not for NYSE. This is expected as the list-wise loss focuses on the entire list, whereas *MRRT* measures the performance of the topmost prediction only.

E. TRAINING TIME PERFORMANCE

Median training time per epoch varies significantly between different combinations of node embedding techniques and loss functions, as depicted in Table 5. The training time is significantly lower for the same embedding technique when the list-wise loss function is used compared to the combination of point-wise and the pair-wise loss function. The training

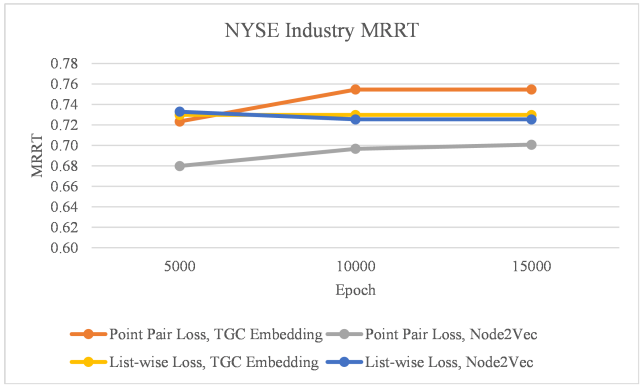


FIGURE 10. MRRT of NYSE with Industry graph.

TABLE 5. Median training time per epoch (seconds).

	NASDAQ		NYSE	
	Wikidata	Industry	Wikidata	Industry
Point Pair Loss, TGC Embedding	1.94	2.23	3.52	6.22
Point Pair Loss, Node2Vec	1.45	1.42	1.76	1.58
List-wise loss, TGC Embedding	1.73	2.25	3.34	6.00
List-wise loss, Node2Vec	1.40	1.49	1.61	1.64

time reduces significantly for a sparse graph such as Wikidata when the list-wise loss is applied. For NASDAQ and Wikidata combination, the training time is 10.57% lower (1.73 seconds compared to 1.94 seconds) when TGC embedding is applied with the list-wise loss compared to the benchmark loss function. It is 5.09% lower for NYSE and Wikidata combination (3.34 seconds compared to 3.52 seconds) with the list-wise loss compared to the benchmark loss function. This portrays a significant advantage of the list-wise loss for stock ranking prediction. The list-wise loss can result in improved performance in terms of *NRBO*@10 and *MRRT* and reduce the training time simultaneously for a sparse graph such as the Wikidata graph.

The difference in the training time is less obvious for a denser graph such as the Industry graph. For NASDAQ and Industry combination, the training time is almost identical (2.25 and 2.23 seconds) between both loss functions with TGC embedding. It is 3.63% lower for NYSE and Industry combination (6.00 seconds compared to 6.22 seconds). We can get improved performance for top k stock ranking prediction (as evident in *NRBO*@10) and reduce the training time to some extent by applying the list-wise loss for denser graphs.

When Node2Vec is applied, we observe a significant gain in the training time, especially for a denser graph like the Industry graph. For NASDAQ and Wikidata combination, the best training time is 1.40 seconds which is obtained using the list-wise loss and Node2Vec. This is a 27.6%

TABLE 6. IRR Performance for the best *MRRT* and the best *NRBO*@10 cases.

Market Name	Graph Type	Best MRRT	Best NRBO@10	IRR for the best test MRRT	IRR for the best test NRBO@10	Relative IRR Gain (%)
NASDAQ	Wikidata	0.430	0.368	12.069	12.664	4.928
NASDAQ	Industry	0.460	0.399	13.339	13.394	0.419
NYSE	Wikidata	0.716	0.645	15.920	15.964	0.281
NYSE	Industry	0.755	0.685	16.282	16.373	0.558

TABLE 7. *NRBO*@10 of NYSE Industry comparison.

Epoch	Point Pair Loss, TGC Embedding	Point Pair Loss, Node2Vec	List-wise Loss, TGC Embedding	List-wise Loss, Node2Vec (Default)	List-wise Loss, Node2Vec (Optimized)
15000	0.677	0.638	0.685	0.647	0.647

TABLE 8. *NRBO*@10 of NYSE Wikidata comparison.

Epoch	Point Pair Loss, TGC Embedding	Point Pair Loss, Node2Vec	List-wise Loss, TGC Embedding	List-wise Loss, Node2Vec (Default)	List-wise Loss, Node2Vec (Optimized)
15000	0.645	0.637	0.644	0.637	0.644

reduction in the training time compared to the baseline model. It comes with a 2.65% relative performance boost in terms of *NRBO*@10 compared to the baseline model.

For NASDAQ and Industry combination, the training time is reduced by 36.1% when Node2Vec is used along with the combination of the point-wise and pair-wise loss. It is reduced by 33.2% when Node2Vec is used along with the list-wise loss function. This improvement in the training time is obtained with comparable *NRBO*@10 performance (0.369 compared to 0.374).

The highest reduction in the training time is observed for NYSE and Industry graph combination. With the combination of the point-wise and the pair-wise loss, the median training time is 1.58 seconds (74.6% relative reduction compared to the baseline model). It is 1.64 seconds with the list-wise loss, which is a relative reduction of 73.7% compared to the baseline model.

F. INVESTMENT PERFORMANCE

We apply the daily buy-hold-sell trading strategy for the top 10 stocks. For each stock market and relation graph combination, we calculate the corresponding IRR for the best *MRRT* and the best *NRBO*@10 cases. The IRR performance is presented in Table 6. In all four cases, the IRR corresponding to the best *NRBO*@10 is higher than the IRR corresponding to the best *MRRT*.

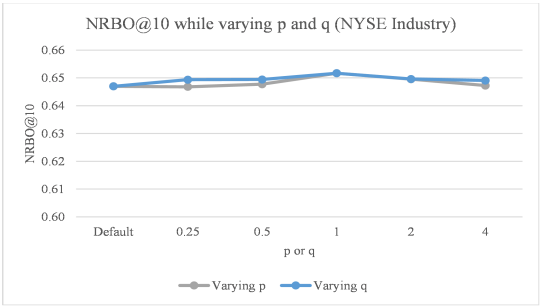
The highest relative gain is 4.928% for the NASDAQ and Wikidata combination. The *NRBO*@10-based trading strategy generates 12.664% IRR, whereas the *MRRT*-based strategy generates 12.069% IRR in this case. The second-best gain in IRR has been observed for the NYSE and Industry combination. The relative gain in IRR is 0.558% in this case. The *NRBO*@10-based trading strategy generates 16.373% IRR, whereas the *MRRT*-based strategy generates 16.282% IRR. The lowest relative gain is 0.281% for the NYSE and Wikidata combination (15.964% vs. 15.920%). This result validates the effectiveness of *NRBO*@k as an evaluation measure. If the investment decision is taken based on *NRBO*@k, that will generate a higher investment return than the decision based on *MRRT*.

G. HYPERPARAMETER TUNING OF Node2Vec

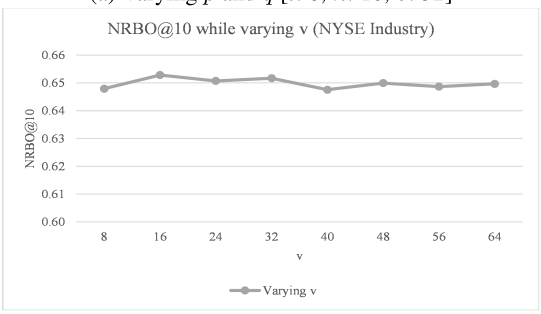
As mentioned in Section VI, we use some default hyperparameters of Node2Vec during training the models. We also try to improve the performance of Node2Vec by tuning those hyperparameters. The *NRBO*@10 performance for NYSE is degraded compared to the baseline model when Node2Vec is applied. So, we do hyperparameter tuning for NYSE with both Industry and Wikidata relation graph. We start with two hyperparameters p, and q, for the Industry graph. While changing one of them, we keep the other fixed at a value of 1. We use default values of all other Node2Vec hyperparameters while varying p and q. We explore five different values for each of p and q.

As evident from Fig. 11a, the maximum validation *NRBO*@10 is 0.652 for NYSE with the Industry graph, which is obtained when $p = 1$ and $q = 1$. We use these two values as the optimum value of p and q for NYSE with Industry graph. After specifying optimum values of p and q, we explore different values of embedding shape v while fixing p and q to optimum values and l and n to default values. The highest validation *NRBO*@10 remains the same as 0.652 while using v as 16. After fixing the values of p, q and v, we vary the value of the length of random walk, l, and the validation result is presented in Fig. 11c. We get a slight improvement of validation *NRBO*@10 (0.653) while using l as 8.

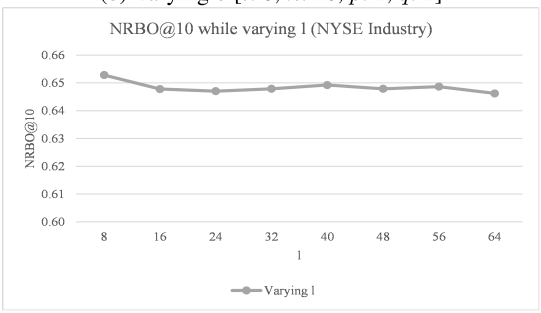
Finally, we vary the value of the number of random walks, n, and according to Fig. 11d, the best n is 10. Thus, for NYSE with the Industry graph, the optimal hyperparameter combination is $l : 8$, $n : 10$, $p : 1$, $q : 1$, and $v : 16$. The optimal test set *NRBO*@10 is presented in Table 7. We can see that the test *NRBO*@10 performance for NYSE Industry does not improve even after hyperparameter tuning of Node2Vec. Test *NRBO*@10 with optimized Node2Vec is 0.647, which is the same as the *NRBO*@10 with default Node2Vec parameters for NYSE and Industry graph combination.



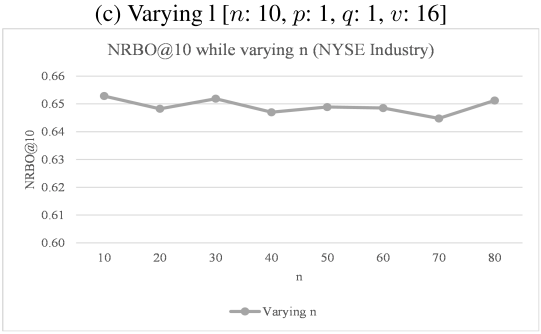
(a) Varying p and q [l: 8, n: 10, v: 32]



(b) Varying v [l: 8, n: 10, p: 1, q: 1]

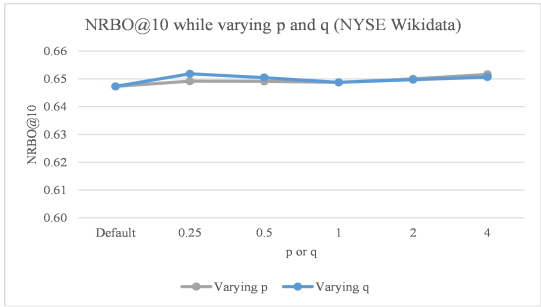


(c) Varying l [n: 10, p: 1, q: 1, v: 16]

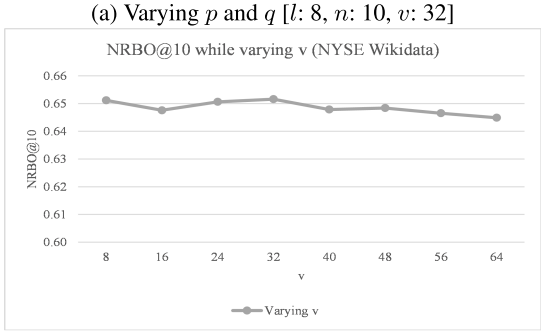


(d) Varying n [l: 8, p: 1, q: 1, v: 16]

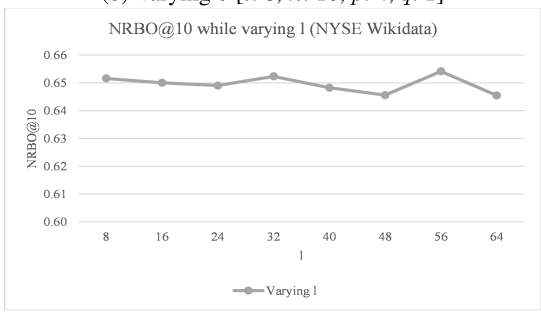
FIGURE 11. Varying hyperparameters of Node2Vec with list-wise loss for NYSE Industry.



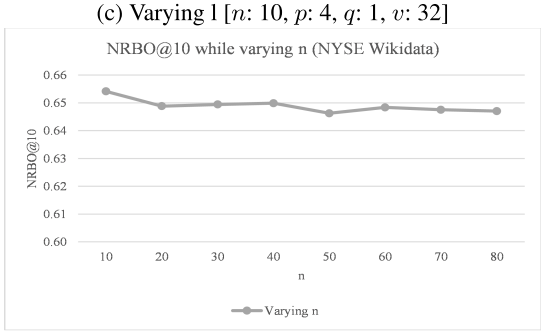
(a) Varying p and q [l: 8, n: 10, v: 32]



(b) Varying v [l: 8, n: 10, p: 4, q: 1]



(c) Varying l [n: 10, p: 4, q: 1, v: 32]



(d) Varying n [l: 56, p: 4, q: 1, v: 32]

FIGURE 12. Varying hyperparameters of Node2Vec with list-wise loss for NYSE Wikidata.

We follow the same procedure for tuning the hyperparameter for NYSE and Wikidata graph. As depicted in Fig. 12a, the best validation *NRBO*@10 (0.652) is obtained with $p = 4$ and $q = 1$ for NYSE and Wikidata graph. With these optimum values, when v is varied, the best validation *NRBO*@10 remains the same with $v = 32$, as depicted in Fig. 12b.

With optimum values of p, q, and v, we vary l, and the validation results are presented in Fig. 12c. The best validation *NRBO*@10 of 0.654 is obtained when l is 56. Finally, we vary the value of n, and the best validation *NRBO*@10 is obtained with $n = 10$, as depicted in Fig. 12d. Thus, for NYSE with Wikidata graph, optimal hyperparameter combination is $l : 56$, $n : 10$, $p : 4$, $q : 1$, and $v : 32$. The optimal test set *NRBO*@10 is presented in Table 8. We can see the *NRBO*@10 performance has improved significantly from 0.637 to 0.644 after hyperparameter tuning of Node2Vec. This performance is almost equal to the *NRBO*@10 (0.645) of the baseline model. It again shows that we can get close to optimal performance using Node2Vec for sparse stock relation graphs if the hyperparameters are tuned properly.

VIII. CONCLUSION

In this paper, we apply a graph-based technique, use an optimal loss function (e.g., the list-wise loss), propose a new performance evaluation measure (e.g., *NRBO@k*), and apply a node embedding technique (e.g., Node2Vec) for stock ranking prediction. Our study has three key findings. Firstly, the list-wise loss results in better *NRBO*@10 in three out of four cases with the same embedding technique. *NRBO*@10 improves by 2.65% when the list-wise loss is used compared to the combined point-wise and pair-wise loss for NASDAQ with Wikidata graph. This gain is 6.7% for NASDAQ with Industry graph and 1.47% for NYSE with Industry graph. This study has demonstrated that the list-wise loss function is the optimal choice over the combination of point-wise and pair-wise loss function.

Secondly, the existing stock ranking performance evaluation measures are suboptimal due to their limitations, such as lack of top-weightedness or requirement for manual intervention. *NRBO@k* overcomes these limitations, and it is also a bounded measure for any value of k, which is necessary for an objective measure of ranking prediction of finite lists. We get better investment returns when the top stocks are selected based on *NRBO*@10 compared to *MRRT*-based stock selection. We simulate a daily buy-hold-sell trading strategy for the top 10 stocks. We get a higher investment return prior to the transaction cost with the *NRBO*@10-based trading strategy compared to *MRRT*-based trading strategy in all cases. The relative gain in IRR ranges from 0.281% to 4.928%. It shows the effectiveness of our proposed evaluation measure from a pragmatic investment perspective. We have explored a simple trading strategy in this study. Further investment strategies can be explored using the predicted rankings. For example, the ranking task can be modified to explore short-selling strategies. It is also possible to explore different values of k and compare the performances.

Thirdly, Node2Vec results in a significant reduction in the training time, and it can achieve the same or better prediction performance for sparse graphs with proper hyperparameter tuning. We obtain the best *NRBO*@10 by applying the list-wise loss with Node2Vec for NASDAQ and Wikidata. It comes with an additional 27.6% reduction in the training time than the baseline model. Node2Vec with the list-wise loss shows the same *NRBO*@10 as the baseline model for NASDAQ with the Industry graph. The reduction in the training time is 33.2% in this case. It also improves the performance for sparse graphs when the hyperparameters are tuned properly. We can get the same *NRBO*@10 for NYSE with Wikidata graph when the hyperparameters of Node2Vec are tuned. It also reduces the training time by 54.4%. However, the performance of Node2Vec is slightly degraded than the baseline model when the stock relation graph is dense such as NYSE with Industry graph. We would like to consider this as future research which can focus on the performance improvement of Node2Vec with a dense stock market graph. It will be an interesting idea to compare the stock ranking performance of Node2Vec with other node embedding techniques such as DeepWalk or GF.

Overall, we have demonstrated that the list-wise loss is a better choice when a graph-based approach is used for stock ranking prediction. Node2Vec can improve or achieve comparable prediction performance for sparse graphs. Further research is required to improve its performance for denser graphs. However, it can reduce the training time significantly in all cases. *NRBO@k* is a better evaluation measure for stock ranking prediction as it overcomes the limitations of existing evaluation measures. *NRBO@k* based trading strategy can generate higher investment return which validates its effectiveness for stock ranking prediction.

REFERENCES

[1] E. F. Fama, "Efficient capital markets: A review of theory and empirical work," *J. Finance*, vol. 25, no. 2, pp. 383–417, May 1970.

[2] H. Jacobs, "What explains the dynamics of 100 anomalies?" *J. Banking Finance*, vol. 57, pp. 65–85, Aug. 2015.

[3] B. G. Malkiel, "The efficient market hypothesis and its critics," *J. Econ. Perspect.*, vol. 17, no. 1, pp. 59–82, Feb. 2003.

[4] C.-F. Tsai and Y.-C. Hsiao, "Combining multiple feature selection methods for stock prediction: Union, intersection, and multi-intersection approaches," *Decis. Support Syst.*, vol. 50, no. 1, pp. 258–269, Dec. 2010.

[5] J. Patel, S. Shah, P. Thakkar, and K. Kotecha, "Predicting stock and stock price index movement using trend deterministic data preparation and machine learning techniques," *Expert Syst. Appl.*, vol. 42, no. 1, pp. 259–268, Jan. 2015.

[6] Z. Li and V. Tam, "A machine learning view on momentum and reversal trading," *Algorithms*, vol. 11, no. 11, p. 170, Oct. 2018.

[7] B. Weng, M. A. Ahmed, and F. M. Megahed, "Stock market one-day ahead movement prediction using disparate data sources," *Expert Syst. Appl.*, vol. 79, pp. 153–163, Aug. 2017.

[8] S. Feuerriegel and H. Prendinger, "News-based trading strategies," *Decis. Support Syst.*, vol. 90, pp. 65–74, Oct. 2016.

[9] Q. Song, A. Liu, and S. Y. Yang, "Stock portfolio selection using learning-to-rank algorithms with news sentiment," *Neurocomputing*, vol. 264, pp. 20–28, Nov. 2017.

[10] F. Feng, X. He, X. Wang, C. Luo, Y. Liu, and T.-S. Chua, "Temporal relational ranking for stock prediction," *ACM Trans. Inf. Syst.*, vol. 37, no. 2, pp. 1–30, Mar. 2019.

[11] T. Fischer and C. Krauss, "Deep learning with long short-term memory networks for financial market predictions," *Eur. J. Oper. Res.*, vol. 270, no. 2, pp. 654–669, Oct. 2018.

[12] W. Bao, J. Yue, and Y. Rao, "A deep learning framework for financial time series using stacked autoencoders and long-short term memory," *PLoS ONE*, vol. 12, no. 7, Jul. 2017, Art. no. e0180944.

[13] Y. Baek and H. Y. Kim, "ModAugNet: A new forecasting framework for stock market index value with an overfitting prevention LSTM module and a prediction LSTM module," *Expert Syst. Appl.*, vol. 113, pp. 457–480, Dec. 2018.

[14] E. F. Fama and K. R. French, "The capital asset pricing model: Theory and evidence," *J. Econ. Perspect.*, vol. 18, no. 3, pp. 25–46, 2004.

[15] C.-F. Huang, "A hybrid stock selection model using genetic algorithms and support vector regression," *Appl. Soft Comput.*, vol. 12, no. 2, pp. 807–818, Feb. 2012.

[16] W. Webber, A. Moffat, and J. Zobel, "A similarity measure for indefinite rankings," *ACM Trans. Inf. Syst.*, vol. 28, no. 4, pp. 1–38, Nov. 2010.

[17] Y. Chen, Z. Wei, and X. Huang, "Incorporating corporation relationship via graph convolutional neural networks for stock price prediction," in *Proc. 27th ACM Int. Conf. Inf. Knowl. Manage.*, Oct. 2018, pp. 1655–1658.

[18] A. Grover and J. Leskovec, "Node2vec: Scalable feature learning for networks," in *Proc. 22nd ACM SIGKDD Int. Conf. Knowl. Discovery Data Mining*, Aug. 2016, pp. 855–864.

[19] A. F. Darrat and M. Zhong, "On testing the random-walk hypothesis: A model-comparison approach," *Financial Rev.*, vol. 35, no. 3, pp. 105–124, Aug. 2000.

[20] G. Bhardwaj and N. R. Swanson, "An empirical investigation of the usefulness of ARFIMA models for predicting macroeconomic and financial time series," *J. Econometrics*, vol. 131, nos. 1–2, pp. 539–578, Mar. 2006.
[21] B. M. Henrique, V. A. Sobreiro, and H. Kimura, "Literature review: Machine learning techniques applied to financial market prediction," *Expert Syst. Appl.*, vol. 124, pp. 226–251, Jun. 2019.
[22] M. Qiu and Y. Song, "Predicting the direction of stock market index movement using an optimized artificial neural network model," *PLoS ONE*, vol. 11, no. 5, May 2016, Art. no. e0155133.
[23] M.-C. Lee, "Using support vector machine with a hybrid feature selection method to the stock trend prediction," *Expert Syst. Appl.*, vol. 36, no. 8, pp. 10896–10904, Oct. 2009.
[24] M. G. Novak and D. Velušček, "Prediction of stock price movement based on daily high prices," *Quant. Finance*, vol. 16, no. 5, pp. 793–826, May 2016.
[25] J. Cao, Z. Li, and J. Li, "Financial time series forecasting model based on CEEMDAN and LSTM," *Phys. A, Stat. Mech. Appl.*, vol. 519, pp. 127–139, Apr. 2019.
[26] L.-J. Kao, C.-C. Chiu, C.-J. Lu, and C.-H. Chang, "A hybrid approach by integrating wavelet-based feature extraction with MARS and SVR for stock index forecasting," *Decis. Support Syst.*, vol. 54, no. 3, pp. 1228–1244, Feb. 2013.
[27] Y. Kara, M. A. Boyacioglu, and Ö. K. Baykan, "Predicting direction of stock price index movement using artificial neural networks and support vector machines: The sample of the istanbul stock exchange," *Expert Syst. Appl.*, vol. 38, no. 5, pp. 5311–5319, May 2011.
[28] E. Chong, C. Han, and F. C. Park, "Deep learning networks for stock market analysis and prediction: Methodology, data representations, and case studies," *Expert Syst. Appl.*, vol. 83, pp. 187–205, Oct. 2017.
[29] K. Chen, Y. Zhou, and F. Dai, "A LSTM-based method for stock returns prediction: A case study of China stock market," in *Proc. IEEE Int. Conf. Big Data (Big Data)*, Oct. 2015, pp. 2823–2824.
[30] H. Y. Kim and C. H. Won, "Forecasting the volatility of stock price index: A hybrid model integrating LSTM with multiple GARCH-type models," *Expert Syst. Appl.*, vol. 103, pp. 25–37, Aug. 2018.
[31] A. Mittal and A. Goel, "Stock prediction using twitter sentiment analysis," Standford Univ., Stanford, CA, USA, Tech. Rep. CS229, 2012.
[32] Y. Gu, T. Shibukawa, Y. Kondo, S. Nagao, and S. Kamijo, "Prediction of stock performance using deep neural networks," *Appl. Sci.*, vol. 10, no. 22, p. 8142, Nov. 2020.
[33] N. K. Avkiran and H. Morita, "Predicting japanese bank stock performance with a composite relative efficiency metric: A new investment tool," *Pacific-Basin Finance J.*, vol. 18, no. 3, pp. 254–271, Jun. 2010.
[34] X. Yang, W. Liu, L. Wang, C. Qu, and J. Bian, "A divide-and-conquer framework for attention-based combination of multiple investment strategies," in *Proc. IEEE Global Conf. Signal Inf. Process. (GlobalSIP)*, Nov. 2019, pp. 1–5.
[35] C. Chen, L. Zhao, J. Bian, C. Xing, and T.-Y. Liu, "Investment behaviors can tell what inside: Exploring stock intrinsic properties for stock trend prediction," in *Proc. 25th ACM SIGKDD Int. Conf. Knowl. Discovery Data Mining*, Jul. 2019, pp. 2376–2384.
[36] X. Ying, C. Xu, J. Gao, J. Wang, and Z. Li, "Time-aware graph relational attention network for stock recommendation," in *Proc. 29th ACM Int. Conf. Inf. Knowl. Manage.*, Oct. 2020, pp. 2281–2284.
[37] W. Chen, T.-Y. Liu, Y. Lan, Z.-M. Ma, and H. Li, "Ranking measures and loss functions in learning to rank," in *Proc. Adv. Neural Inf. Process. Syst.*, 2009, pp. 315–323.
[38] Z. Cao, T. Qin, T.-Y. Liu, M.-F. Tsai, and H. Li, "Learning to rank: From pairwise approach to listwise approach," in *Proc. 24th Int. Conf. Mach. Learn. (ICML)*, 2007, pp. 129–136.
[39] C. Burges, T. Shaked, E. Renshaw, A. Lazier, M. Deeds, N. Hamilton, and G. Hullender, "Learning to rank using gradient descent," in *Proc. 22nd Int. Conf. Mach. Learn. (ICML)*, 2005, pp. 89–96.
[40] W. L. Hamilton, R. Ying, and J. Leskovec, "Representation learning on graphs: Methods and applications," 2017, *arXiv:1709.05584*. [Online]. Available: http://arxiv.org/abs/1709.05584
[41] P. Goyal and E. Ferrara, "Graph embedding techniques, applications, and performance: A survey," *Knowl.-Based Syst.*, vol. 151, pp. 78–94, Jul. 2018.
[42] S. T. Roweis, "Nonlinear dimensionality reduction by locally linear embedding," *Science*, vol. 290, no. 5500, pp. 2323–2326, Dec. 2000.
[43] A. Ahmed, N. Shervashidze, S. Narayanamurthy, V. Josifovski, and A. J. Smola, "Distributed large-scale natural graph factorization," in *Proc. 22nd Int. Conf. World Wide Web (WWW)*, 2013, pp. 37–48.
[44] B. Perozzi, R. Al-Rfou, and S. Skiena, "DeepWalk: Online learning of social representations," in *Proc. 20th ACM SIGKDD Int. Conf. Knowl. Discovery Data Mining*, Aug. 2014, pp. 701–710.
[45] S. Hochreiter and J. Schmidhuber, "Long short-term memory," *Neural Comput.*, vol. 9, no. 8, pp. 1735–1780, 1997.
[46] T. Mikolov, I. Sutskever, K. Chen, G. S. Corrado, and J. Dean, "Distributed representations of words and phrases and their compositionality," in *Proc. Adv. Neural Inf. Process. Syst.*, vol. 26, 2013, pp. 3111–3119.
[47] H. Markowitz, "Portfolio selection," *J. Finance*, vol. 7, no. 1, pp. 77–91, 1952.
[48] C. B. Kalayci, O. Ertenlice, and M. A. Akbay, "A comprehensive review of deterministic models and applications for mean-variance portfolio optimization," *Expert Syst. Appl.*, vol. 125, pp. 345–368, Jul. 2019.
[49] K. Järvelin and J. Kekäläinen, "Cumulated gain-based evaluation of IR techniques," *ACM Trans. Inf. Syst.*, vol. 20, no. 4, pp. 422–446, Oct. 2002.
[50] D. P. Kingma and J. Ba, "Adam: A method for stochastic optimization," in *Proc. 3rd Int. Conf. Learn. Represent., (ICLR)*, 2015, pp. 1–15.
[51] X. Glorot and Y. Bengio, "Understanding the difficulty of training deep feedforward neural networks," in *Proc. 19th Int. Conf. Artif. Intell. Statist.*, vol. 9, 2010, pp. 249–256.

SUMAN SAHA (Graduate Student Member, IEEE) received the B.Sc. degree in electrical and electronic engineering from the Bangladesh University of Engineering and Technology, Dhaka, Bangladesh, in 2009, and the M.B.A. degree in finance from the Institute of Business Administration, University of Dhaka, Dhaka, in 2015. He is currently pursuing the Ph.D. degree with the Discipline of Business Analytics, The University of Sydney Business School, NSW, Australia. From 2010 to 2018, he worked with leading telecommunication operators in Bangladesh and an expert of end to end mobile network performance management. His research interests concentrate on the application of machine learning and graph theory for analyzing financial market.

JUNBIN GAO received the B.Sc. degree in computational mathematics from the Huazhong University of Science and Technology (HUST), Wuhan, China, in 1982, and the Ph.D. degree from the Dalian University of Technology, Dalian, China, in 1991. From 1982 to 2001, he was an Associate Lecturer, a Lecturer, an Associate Professor, and a Professor with the Department of Mathematics, HUST. From 2001 to 2005, he was a Lecturer of computer science with the University of New England, Armidale, NSW, Australia, where he was a Senior Lecturer. He was a Professor of computer science with the School of Computing and Mathematics, Charles Sturt University, Australia. He is currently a Professor of big data analytics with The University of Sydney Business School, The University of Sydney, Sydney, NSW, Australia. His current research interests include machine learning, data analytics, Bayesian learning and inference, and image analysis.

RICHARD GERLACH received the Ph.D. degree in statistics from the University of New South Wales, Sydney, NSW, Australia, in 2001. He was a Lecturer in statistics with the University of Newcastle, NSW, Australia, from 2001 to 2005. From 2006 to 2009, he worked as a Senior Lecturer in econometrics and business statistics with The University of Sydney, Sydney, where he was as an Associate Professor in econometrics and business statistics, from 2010 to 2013. He is currently a Professor of business analytics with The University of Sydney Business School, The University of Sydney. His research interests lie mainly in financial econometrics and time series. His work has concerned developing time series models for measuring, forecasting, and managing risk in financial markets and computationally intensive Bayesian methods for inference, diagnosis, forecasting, and model comparison for these models.