Available online at www.sciencedirect.com

ScienceDirect

Procedia Computer Science 131 (2018) 895–903

Procedia Computer Science

www.elsevier.com/locate/procedia

8th International Congress of Information and Communication Technology (ICICT-2018)

Deep Learning with Gated Recurrent Unit Networks for Financial Sequence Predictions

Guizhu Shen[a,*], Qingping Tan[a], Haoyu Zhang[a], Ping Zeng[a], Jianjun Xu[a]

[a] College of Computer, National University of Defense Technology, Changsha, China

Abstract

Gated recurrent unit (GRU) networks perform well in sequence learning tasks and overcome the problems of vanishing and explosion of gradients in traditional recurrent neural networks (RNNs) when learning long-term dependencies. Although they apply essentially to financial time series predictions, they are seldom used in the field. To fill this void, we propose GRU networks and its improved version for predicting trading signals for stock indexes of the Hang Seng Indexes (HSI), the Deutscher Aktienindex (DAX) and the S&P 500 Index from 1991 to 2017, and compare the GRU-based models with the traditional deep net and the benchmark classifier support vector machine (SVM). Experimental results show that the two GRU models proposed in this paper both obtain higher prediction accuracy on these data sets, and the improved version can effectively improve the learning ability of the model.

Keywords: Deep learning; GRU; SVM; financial time series; stock index

1. Introduction

Predicting financial time series is extremely difficult, mainly due to the essentially high-noise characteristic and the semi-strong form of market efficiency, approved by the general[1]. Yet, numerous renowned anomalies in capital market form a sharp contrast to the concept of market efficiency. For instance, some research on capital market anomalies have demonstrated that 100 or more such anomalies effectively beat the market depending on return predictive signals. They use these signals as the features (input) of specific financial models in order to get future returns as the targets (output). However, such financial models used cannot deal with complicated non-linear dependencies and are generally transparent in essence.

* Corresponding author. Tel.: +86-138-4053-8916.
E-mail address: 13840538916@163.com

10.1016/j.procs.2018.04.298

In recent years, preliminary studies have shown that machine learning (ML) technologies are able to effectively capture the non-linear structures in the complex financial market data. Specifically, Huck (2009)[3] deploys Elman neural networks combined with a multiple criteria decision-making approach named ELECTRE III on all S&P 100 constituents from 1992 to 2006. They reach directional accuracy of 54 percent at around 0.8 percent profits per week when k = 5 (k is the number of stocks bought or sold in the transaction process). Takeuchi and Lee (2013)[4] propose an intensive momentum strategy based on deep neural networks (DNNs). The empirical application on the out-of-sample trading period from the U.S. CRSP stock data between 1965 and 2009 produces returns of 45.93 percent per year. A similar model is run by Dixon et al. (2015)[5]. They set the frequency to a higher level with binning return data per 5 minutes, resulting in massive classification accuracies of more than 73 percent. Moritz and Zimmermann (2014)[6] develop a statistical arbitrage strategy based on "deep conditional portfolio sorts" using random forests (RFs). Further, the average monthly excess return which is risk-adjusted is 2 percent on U.S. CRSP stock universe between 1968 and 2012. All constituents on S&P 500 from 1992 to 2015 are predicted by Krauss et al. (2017)[7] using deep learning, RFs, gradient-boosted trees as well as a combination of them. They prove that a simple ensemble consisting of the three models above can get daily returns of more than 0.45 percent during the out-of-sample testing period for k = 10.

Given the available literature, the models applied are all memory-free, cannot take into account the impact of the previous information at present, and financial data exists time-series character. Therefore, the RNN network with memory function is apparently more suitable for this kind of task. In this paper, we primarily make three contributions.

- First, we focus on gated recurrent unit (GRU) networks, a kind of gated RNNs, which largely mitigate the problem of gradient vanishing of RNNs through gating mechanism and make the structure simpler while maintaining the effect of LSTM (another very popular gated RNNs).
- Second, we propose an improved vision on the basis of the traditional GRU networks. For classification tasks, like most of the “deep learning” models, GRU networks employ the softmax activation function for prediction and minimize cross-entropy loss. In this paper, we demonstrate a small but consistent advantage of replacing the softmax layer of the GRU network with a support vector machine. Learning minimizes a margin-based loss instead of the cross-entropy loss.
- Third, we select the representative stock indexes of the HIS, the DAX and the S&P 500 in three financial markets of Asia, Europe and the Americas to train the model and predict the operation signals. Particularly, in the task of predicting those financial time series, we provide an exhaustive description of how the raw data is preprocessed as well as how the GRU-based models are deployed and trained. In addition, we introduce the benchmark models (a DNN-based model and a standard SVM) chosen to be the comparisons with our findings. The former is used to show the superiority provided by the memory function of the GRU networks, the latter as a baseline. In statistics and economics, both GRU-based models make higher accuracies and more returns. And the GRU-SVM models perform slightly better.

The remainder of this paper is organized as follows. Section 2 briefly covers the architectures of the GRU models and the GRU-SVM models. Section 3 provides the experimental details. Section 4 presents the results and the discussion. Finally, section 5 draws conclusions.

2. Method

2.1. GRU-based models

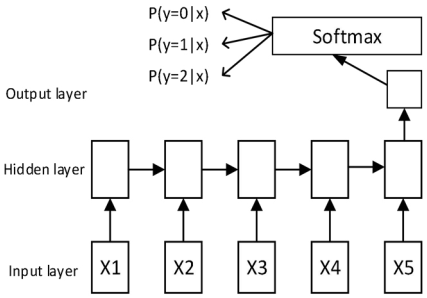

Fig. 1. Structure of GRU-based model.

GRU networks fall into the category of RNNs, i.e., neural networks whose underlying topology of inter-neuronal connections contains at least one cycle[8]. They were introduced in 1997 and further improved over the next few years. GRU is one kind of the gated RNNs which are used to solve the common problems of vanishing and exploding gradients in traditional RNNs when learning long-term dependencies.

As illustrated in Fig. 1, there is an input layer composed of multiple neurons, the number of neurons is determined by the size of the feature space. Similarly, the number of neurons in the output layer corresponds to the output space. The hidden layer(s) containing memory cells cover the main functions of the GRU networks. Changes and maintenance of cell status depend on two gates in the cell: a reset gate r_t and an update gate u_t. The structure of a memory cell is illustrated as a circuit diagram in Fig. 2.

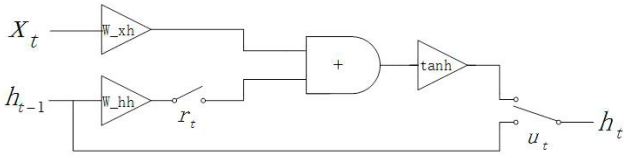

Fig. 2. Structure of GRU memory cell.

Both gates are related to two elements: x_t and h_{t-1}. The former comes from the input sequence, the latter is the output value of memory cells from the previous time point. Accordingly, each gate fulfills different tasks in order to achieve the purpose of filtering:

- The reset gate is used to control the influence of h_{t-1} (information of the unit at the previous timestep) on the current information x_t. If h_{t-1} is not important to x_t, then r_t can be opened, making h_{t-1} does not affect x_t.
- The update gate specifies whether to ignore the current information x_t. When u_t is turned on the under branch, we will ignore the current information x_t, while forming a "short-circuit connection" from h_{t-1} to h_t. This makes the gradient reversely propagate, effectively solving the gradient vanishing problem.

The equations below describe how memory cells at each hidden layer of the GRU networks are updated at each time step.

The equation of the reset gate:

$$u_t = s(W_u \cdot [h_{t-1}, x_t]) \tag{1}$$

The equation of the update gate:

$$r_t = s(W_r \cdot [h_{t-1}, x_t]) \tag{2}$$

The equations of the output:

$$\widetilde{h}_t = \tanh(W \cdot [r_t * h_{t-1}, x_t]) \tag{3}$$

$$h_t = (1 - u_t) * h_{t-1} + u_t * \widetilde{h}_t \tag{4}$$

The eigenvalues are required to enter in chronological order when GRU networks are dealing with an input sequence, the eigenvalues are required to enter in chronological order. Thus, the GRU networks process the corresponding inputs at each time step t according to the formula described above. The network will return the final output only if it finishes processing the last element in the sequence.

The adjustment methods of parameters (weights as well as bias) are similar to those of the traditional feed-forward neural networks. During training, the loss of the objective function from the training sets is minimized. Cross-entropy is chosen to be the objective function due to the classification task.

We make use of three common techniques in order to train the GRU networks effectively. To begin with, the optimizer is set as RMSprop. It derived from rprop, using mini-batch to optimize. This method is a typically choice for RNNs. Second, for the hidden layers, the dropout regularization is used in order to make it generalize better as well as avoid overfitting. Hereby, in the training process, for the neural network unit, according to a certain probability, it will be discarded from the network. Third, we apply early stopping to fulfill the similar propose as a further mechanism. Hereby, the training data is divided into training set and validation set according to a specific ratio. The former is used for training, and the test results are obtained on the verification set (for example, every 5-epoch make a test). As the epoch increasing, if the loss found on the validation set is rising, stop training. The weights after the stop are restored as the final parameters of the network. Since the best proportion of the post-sample data among the samples is verified as 20 percent, we split the training data into training and validation set following the ratio of 4:1. We set the patience of the training process to 10000 epochs and the maximum early stopping duration to 10. Hereby, we provide a detailed description of our GRU-based models as follows:

- Input layer with 240 timesteps containing one feature.
- GRU layer with 25 hidden neurons and a dropout rate of 0.9.
- Output layer with softmax activation function for ternary classification.

2.2. GRU-SVM-based models

Linear support vector machines (SVM) is originally formulated for binary classification. Given training data and its corresponding labels (x_n, y_n), SVMs learning consists of the following optimization:

$$\min_w \frac{1}{2} w^T w + C \sum_{n=1}^{N} \max(1 - w^T x_n t_n, 0) \tag{5}$$

The objective of equation (4) is known as the primal form problem of L1-SVM, with the standard hinge loss. Since L1-SVM is not differentiable, a popular variation is known as the L2-SVM which minimizes the squared hinge loss:

$$\min_w \frac{1}{2} w^T w + C \sum_{n=1}^{N} \max(1 - w^T x_n t_n, 0)^2 \tag{6}$$

L2-SVM is differentiable and imposes a bigger (quadratic vs. linear) loss for points which violate the margin. To predict the class label of a test data x:

$$\arg\max_t (w^T x) t \tag{7}$$

The simplest way to extend SVMs for multiclass problems is using the so-called one-vs-rest approach[9]. For K class problems, K linear SVMs will be trained independently, where the data from the other classes form the negative cases. There are also other alternative multiclass SVM approaches, but we leave those to future work.

Denoting the output of the k-th SVM as

$$a_k(x) = w^T x \tag{8}$$

The predicted class is

$$\arg\max a_k(x) \tag{9}$$

Note that prediction using SVMs is exactly the same as using a softmax. The only difference between softmax and multi-class SVMs is in their objectives parametrized by all of the weight matrices W. Softmax layer minimizes cross-entropy or maximizes the log-likelihood, while SVMs simply try to find the maximum margin between data points of different classes.

In the experiments section, we simply replace the softmax activation function in the output layer illustrated in Fig. 1 with the L2-SVM's objective to train our GRU networks.

3. Experiments

3.1. Data and software

In this paper, we choose the HSI, the DAX and the S&P 500 Index as the raw data to train and test our models. These three indexes have certain representativeness in financial markets in Asia, Europe and the Americas. Because the broader market is a concentrated manifestation of the overall financial market and excludes the volatility of individual stocks, the selection of the high-liquidity subsets of the stock market is a very effective test set for any trading strategy and has a calculated feasibility. The data are downloaded via Yahoo Finance. For each index, we take the closing price as the predictor. Specific experimental data are described in Table 1.

Data preprocessing and handling are conducted in Python 3.5, relying on the packages numpy and pandas. Deep learning GRU networks and deep neural networks are implemented with TensorFlow, an open source software library for numerical computation using data flow graphs.

Table 1. Experimental Data Set.

Index	Time period	Sample size
HIS	1991/8/27-2017/8/24	6423
DAX	1991/8/23-2017/8/23	6580
S&P 500	1991/8/23-2017/8/23	6551

3.2. Generation of training and trading sets

As per daily data between 1991 and 2017, a training-trading set is defined as a "study period", composed of a training period and a trading period. The former is set to about 750 days, nearly three years, for in-sample training. The latter used for out-of-sample trading is set to 250 days, equivalent to one year. With this setup, we offer plenty of training samples for the models proposed in section 2 to be estimated. Based on this, referring to the sliding-window strategy, the training-trading set is moved forward by a length of 250 days. Hereby, there will be 23 batches not overlapped with each other looping through the whole data set between 1991 and 2017 detailed in Table 1.

3.3. Feature and target generation

Since the values of the financial time series are usually large, which is not conducive to data calculation and model convergence, this paper normalizes the data as follows. Let $P = (P_t)_{t \in T}$ be defined as the close price at time t, with R_t^m the simple return over m periods, i.e.,

$$R_t^m = \frac{P_t}{P_{t-m}} - 1 \tag{10}$$

For the models proposed in this paper, daily returns (m = 1) R_t^1 is calculated for each day t. For standardization, the mean μ_{train}^m is subtracted from returns and the outcome is divided by deviation σ_{train}^m as shown in Equation 10. The mean as well as the deviation are calculated only from the training set.

During training, the input of our models must be in the form of a feature sequence, which means the features should be consecutively point by point chronologically. The single feature in our case is the normalized daily return $\widetilde{R}_t^1$. We choose the length of the input sequence to be 240, containing about one year of the information. Then, the overlapping sequences of length 240, containing the continuous standardization of daily returns $\widetilde{R}_t^1$, are generated as follows: Firstly, the features are ranked in ascending order according to date t. Secondly, for each date ($t \geqslant 240$) in the study period, we construct sequences in the form $\{\widetilde{R}_{t-239}^1, \widetilde{R}_{t-238}^1, ..., \widetilde{R}_t^1\}$. The sequence hence consists of the standardized one-day returns $\{\widetilde{R}_1^1, \widetilde{R}_2^1, ..., \widetilde{R}_{240}^1\}$. The second sequence consists of $\{\widetilde{R}_2^1, \widetilde{R}_3^1, ..., \widetilde{R}_{241}^1\}$ and so forth. Hereby, we can obtain about 760 of such sequences. Of these, about 510 are taken to be the training set and the remaining the out-of-sample testing set.

Table 2. Stock Indexes Trading Signal.

R_t^1	R_{t+1}^1	Y_t	Signal
-	+	1	long
+	-	2	short
-	-	0	flat
+	+	0	flat

For the sake of comparability, we define a multi-class classification task, i.e., for each date t, there are three possible values to be given by the response variable Y_t. We compare daily returns R_t^1 of day t and R_{t+1}^1 of day t+1 to state the three classes. If R_t^1 is negative and R_{t+1}^1 is positive, we take day t as an upward inflection point and go long it, and the response variable Y_t is equal to one (class 1). Similarly, if R_t^1 is positive and R_{t+1}^1 is negative, we take this point as a downward inflection point and go short it, and the response variable Y_t is equal to two (class 2). If the two are the same (i.e., both positive or negative), we think the point is not an inflection point, no trading operation, and the response variable Y_t is equal to zero (class 0). Table 2 gives the details. We construct a classification instead of a regression problem, as the literature suggests that the former performs better than the latter in predicting financial market data[10].

3.4. Benchmark models - deep neural network and SVM

For benchmarking the GRU and the GRU-SVM above, we opt for a common deep neural net to show the superiority of the GRU-based networks, and a SVM to act as baseline. Specifically, the input features (standardized

returns) and the output targets (the operation signals) are generalized in the same way as denoted in the previous subsection.

A standard DNN is configurated as a comparison with our GRU-based models in order to demonstrate the effectiveness and superiority of the method we proposed. Specifically, we opt for a feed forward neural network. There are 240 neurons in the input layer, and 3 in the output layer with softmax. As for the hidden layers, we set it to 3, the number of neurons from the first layer to the third layer is 1, 2 and 3 respectively. We make use of maxout as the activation function and set the rate of dropout to 0.5 as well as the shrinkage of the L1 regularization to 0.0001.

As baseline model, we also deploy SVM trained by LibSVM 3.21[11] . We choose the v-SVC as the SVM type and the RBF as the kernel function. The parameters are optimized using a gird search approach. To avoid overfitting and underfitting, we use k-fold cross validation with k=5. By comparing with SVM, we prove that the GRU-based models are able to obtain more remarkable results than the standard classifier through more complicated and intensive computations.

4. Results and Discussion

We deploy GRU, GRU-SVM, DNN and SVM for predicting the operation signals for every timestep during the whole time period of 3 stock indexes (HIS, DAX and S&P 500). For every index, we use the accuracy of the trading sets as the metric to estimate the performance of each of the 4 models above, it is an essential metric for deep learning and is obtained by calculating the proportion of correct classifications. In addition, by comparing with the benchmarks, we evaluate the profitability of the models proposed in this paper by daily mean returns and cumulative returns of the trading periods.

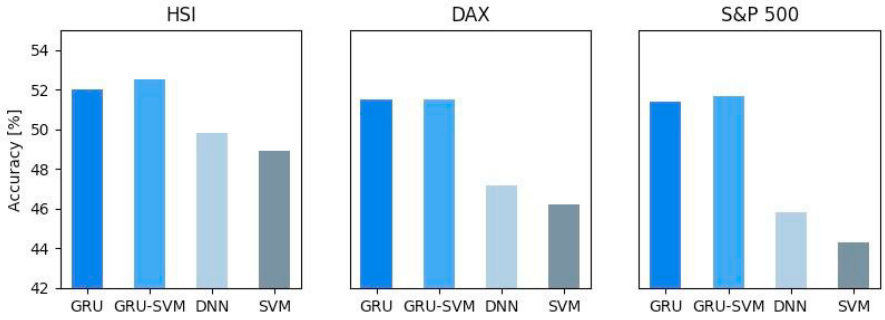

Fig. 3. Accuracies of 3 stock indexes (HIS, DAX and S&P 500) from 1991 until 2017.

We report the accuracies of different methods as shown in Fig. 3 and see the following trends. In the data sets of three stock indexes, the accuracy of the GRU networks and the GRU-SVM networks are always greater than 50 percent - an important benchmark for a dollar neutral strategy[7], this shows our GRU-based models are suitable for studying the prediction tasks on financial time series. Irrespective of the kind of the stock indexes, the GRU-SVM model performs better comparing with other methods. Specifically, the accuracy is at 51.7 percent, while 51.4 percent for the GRU, 45.8 percent for the DNN, and 44.3 for the SVM for S&P 500. The same situations happen in other two indexes, the GRU-SVM achieves the greatest performance, with the exception of the DAX, where its accuracy is tied with the GRU.

With respect to profitability illustrated by Fig. 4 and Fig. 5, both GRU-SVM and GRU exhibit much higher return than the DNN and the SVM - across all data sets, the former is on a similar level as the latter, with slightly higher values for the HIS and the S&P 500. The profitability of the same model on different data sets is different, which is related to the characteristics of the data set itself. However, both of the prediction models proposed in this paper show good profitability under the same conditions.

The above experimental results show that the two GRU based models are feasible and effective in the stock index predicting, of which the GRU-SVM presents the best effect, and the overall prediction effect of traditional GRU is also better than that of DNN and SVM.

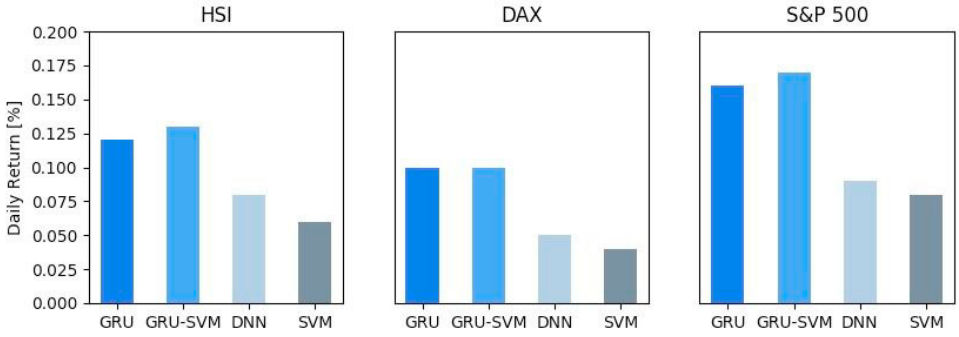

Fig. 4. Daily mean returns of 3 stock indexes (HIS, DAX and S&P 500) from 1991 until 2017.

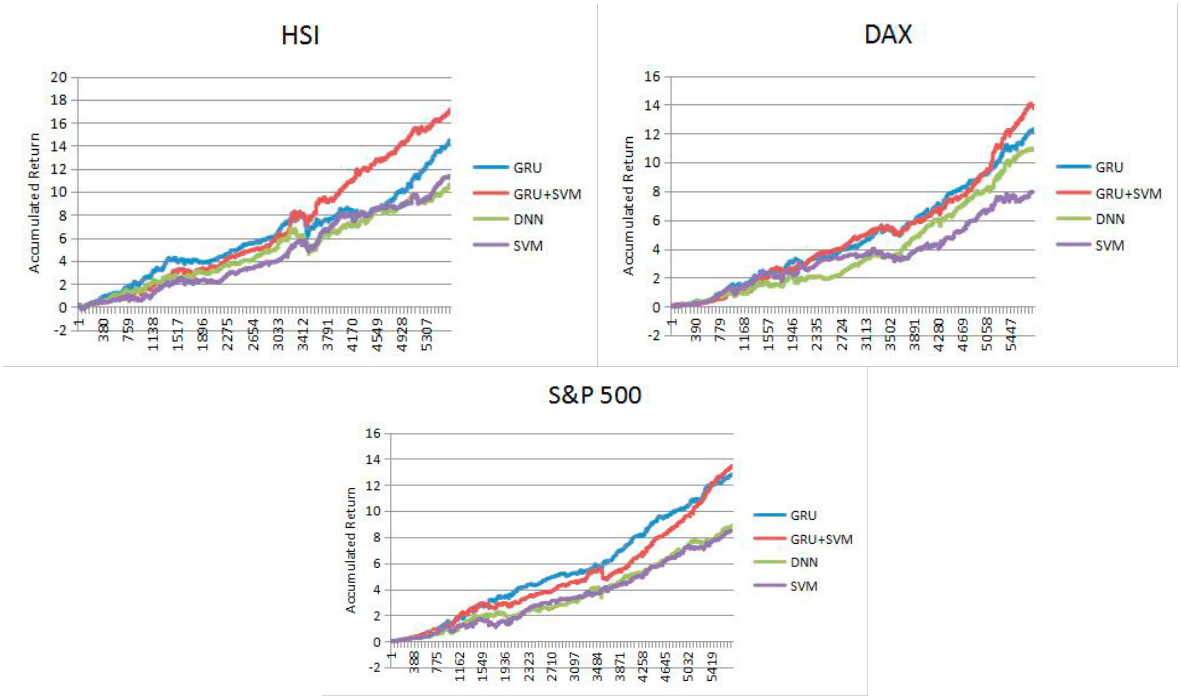

Fig. 5. Accumulated normalized return in the trading periods of the HIS, the DAX and the S&P 500.

5. Conclusions

In this paper, we apply gated recurrent unit network and replace its traditional output layer with SVM to predict the operation signal on the HIS, the DAX and the S&P 500, between December 1994 and October 2017. Our methods are proved to be effective by the experimental results above. The approach of using the GRU-based models to extract useful information from a vast array of financial time-series data has been shown to be effective, and the output layer using SVM outperforms that using softmax. Further research is required to explore other multiclass classifiers to enhance the effect of the GRU model.

Acknowledgements

This study is supported by the Scientific Research Fund of Hunan Provincial Education Department (No. 15A007).

References

1. Fama, E. F., 1970. Efficient capital markets: A review of theory and empirical work. The Journal of Finance 25 (2), 383-417.
2. Jacobs, H., 2015. What explains the dynamics of 100 anomalies? Journal of Banking & Finance 57, 65-85.
3. Huck, N., 2009. Pairs selection and outranking: An application to the S&P 100 index. European Journal of Operational Research 196 (2), 819-825.
4. Takeuchi, L., Lee, Y.-Y., 2013. Applying deep learning to enhance momentum trading strategies in stocks. Working paper, Stanford University.
5. Dixon, M., Klabjan, D., Bang, J. H., 2015. Implementing deep neural networks for financial market prediction on the Intel Xeon Phi. In: Proceedings of the 8th Workshop on High Performance Computational Finance. pp. 1-6.
6. Moritz, B., Zimmermann, T., 2014. Deep conditional portfolio sorts: The relation between past and future stock returns. Working paper, LMU Munich and Harvard University.
7. Krauss, C., Do, X. A., Huck, N., 2017. Deep neural networks, gradient-boosted trees, random forests: Statistical arbitrage on the S&P 500. European Journal of Operational Research 259 (2), 689-702.
8. Medsker, L., 2000. Recurrent Neural Networks: Design and Applications. International Series on Computational Intelligence. CRC-Press.
9. Vapnik, V. N. The nature of statistical learning theory. Springer, New York, 1995.
10. Enke, D. and Thawornwong, S. (2005). The use of data mining and neural networks for forecasting stock market returns. Expert Systems with Applications, 29(4):927-940.
11. Chang C C, Lin C J. LIBSVM: A library for support vector machines[J]. Acm Transactions on Intelligent Systems & Technology, 2007, 2(3, article 27):389-396.