Parallel Heterogeneous Multi-Classifier System for Decision Making in Algorithmic Trading

Yuri Zelenkov [0000-0002-2248-1023]

National Research University Higher School of Economics, Moscow, Russia
yuri.zelenkov@gmail.com

Abstract. The most important factors of successful trading strategy are the decisions to sell or buy. We propose multi-classifier system for decision making in algorithmic trading, whose training is carried out in three stages. At the first stage, features set is calculated based on historical data. These can be oscillators and moments that used in technical analysis, other characteristics of time series, market indexes, etc. At the second stage, base classifiers are trained using genetic algorithms, and optimal feature set for each of them is selected. At the third stage, a voting ensemble is designed, weights of base classifiers are selected also using genetic algorithms. However, the usage of genetic algorithms requires considerable time for computing, so the proposed system is implemented in a parallel environment. Testing on real data confirmed that the proposed approach allows to build a decision-making system, the results of which significantly exceed the trading strategies based on indicators of technical analysis and other techniques of machine learning.

Keywords: Algorithmic trading, Trading strategy; Multi-classifier system; Genetic algorithm.

1 Introduction

Algorithmic Trading (AT) refers to any form of trading using sophisticated algorithms and programmed systems to automate all or some part of the trade cycle [1, 2]. The trade cycle and components of AT system are described in [1,2]. The key stages in AT are the pre-trade analysis, signal generation, trade execution, post-trade analysis, risk management, and asset allocation.

The key factors of a successful trading strategy are the decisions to "buy" or "sell". These solutions are based on the alpha model, which is the mathematical model designed to predict the future behavior of the financial instruments that the algorithmic system is intended to trade [2]. A large number of studies related to the design of alpha models are known, including using machine learning methods. In this paper, we propose a method for designing the alpha model, based on multi-classifier system, whose training is carried out in three stages. At the first stage, features set is calculated based on historical data. At the second stage, base classifiers are trained using genetic algorithms, and optimal feature set for each of them is selected. At the third stage, a voting

ensemble is designed, weights of base classifiers are selected also using genetic algorithms.

To build an alpha model based on a multi-classifier system, the following actions should be performed: obtain and clean data that will drive AT; select base classifiers that mutually complementary; select architecture of their ensemble. Therefore, an organization of paper is following: after literature review in the second section, process of feature engineering is described. In next sections techniques of features wrapping and classifiers combination in ensemble are discussed. All proposed techniques are illustrated by practical examples. Obtained results are compared with other methods. In last section, parallel implementation of proposed algorithm is discussed.

2 Related Works

There are two financial instrument prediction methodologies:

- *Fundamental Analysis* is concerned more with the company and its macro-economic environment rather than the actual asset. The decisions are made based on the past performance of the company, the forecast of earning etc.
- *Technical Analysis* deals with the determination of the asset price based on the past patterns of the stock using time-series analysis.

When applying Machine Learning to stock data, Technical Analysis is the more applicable methodology, because it can learn the underlying patterns in the financial time series. The search of patterns is carried out in two main ways, the first is the identification of graphic figures that are formed by price charts, the second is the calculation of various indicators, the dynamics of which allows predicting asset price behaviour [3,4]. Developers of alpha models based on machine learning usually use technical indicators. For example, Zhang and Ren [5] presented a trading strategy model that utilizes different technical indicators such as Moving Average (MA), Moving Average Convergence Divergence (MACD), Relative Strength Index (RSI), Slow Stochastic etc.

At the same time, various models of machine learning are used. A major difficulty in dealing with financial time series representing asset prices is their non-stationary behavior (or concept drift), i.e. the fact that the underlying data generating mechanism keeps changing over time. Therefore, in real-time forecasting and trading applications one is often interested in on-line learning, a situation where the prediction function is updated following the arrival of each new sample [6]. Various approaches for incremental learning have been proposed in the literature, for both classification [7] and regression problems [6,8].

One of the most popular tools are artificial neural networks (ANNs) [9], which are often used together with evolutionary techniques, such as genetic algorithms (GA), because the combination of two or more techniques offers a better result [10,11]. Scabar and Cloette [12] developed a hybrid prediction model based on an ANN and GA, which gives evidence that financial time series are not entirely random, and that—contrary to the predictions of the efficient markets hypothesis—a trading strategy based solely on historical price data can be used to achieve returns better than those achieved using a

buy-and-hold strategy. Butler [13] developed an Evolutionary ANN (EANN) that makes future predictions based on macro-economic data. Tsai & Chiou [14] used technique that combines ANN with decision trees.

Peters [15,16] proved that time series of stock prices are produced by systems with memory, he also determined cycles for different industries and stock markets. Therefore, the predictive tools that can model the memory effect, for example, recurrent neural networks, are of considerable interest. For example, feedforward networks and recurrent networks (Elman network) that can build "memory" in the evolution of neurons are reviewed in [17] with application to finance.

As follows from this brief survey, the use of statistical characteristics of time series and indicators of technical analysis is a widespread practice in AT. Researchers choose different techniques of machine learning and their combinations. At the same time, comparatively little attention has been paid to researching the possibilities of combinations of simple classifiers, such as k Nearest Neighbors (kNN), Logistic Regression (LR), Naive Bayes (NB), Decision Tree (DT), Support Vector Machine (SVM). This paper intends to fill this gap.

3 Problems of Trading Strategy Design

As it was stated above, we suggest use an ensemble of heterogeneous classifiers. When creating an effective method for design the AT system, several problems must be solved.

The first problem concerns the feature engineering and selection. A feature is a piece of information that might be useful for prediction (wikipedia.org). It can be structured attribute, combination of attributes and any unstructured information that relevant to the context. Feature engineering is the process of using domain knowledge to create features that make model works. Feature selection problem deals with selection of an optimal and relevant set of features that are necessary for the recognition and prediction [18,19]. It helps reduce the dimensionality of the measurement space and facilitates the use of easily computable algorithms for efficient classification.

The second problem concerns hybrid multi classifier system (MCS) design [20]. It is: system topology (how to interconnect individual classifiers), ensemble design (how to drive the generation and selection of a pool of valuable classifiers) and fuser design (how to build a decision combination function).

We use parallel architecture because most MCS's reported in the literature are structured in a parallel topology [20,21]. In this architecture, each classifier is feed the same input data, so that the final decision of the MCS is based on the individual classifiers outputs obtained independently. We use voting ensemble with majority voting rule. In that case MCS output is formed as the weighed sum of individual classifiers responses.

The design of hybrid ensemble should support involving of mutually complementary individual classifiers that provide high diversity and accuracy [21], but it is impossible to predict what classifiers can be complementary. Therefore, we suggest using the combination of several approaches: random sampling, features selection for each individual

classifier on base of the GA; determination of each classifier weight in ensemble also through GA. Thus, the method proposed here includes three stages (Fig. 1). The first one generates a set of features that can be used for forecasting; on the second one, the relevant set of features is selected using genetic algorithms for each individual classifier; on the third one, the weights of the voting ensemble are determined also using genetic algorithms.

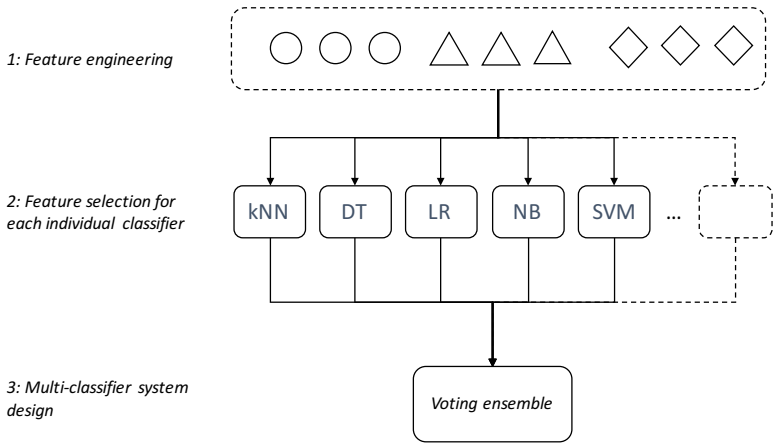


Fig. 1. Three stages of MCS training.

4 Features Engineering

4.1 Target Variable

A time series describing the dynamics of a financial instrument (for example, shares of a company) that can be downloaded from `http://finance.yahoo.com` includes the following variables: opening and closing prices, maximum and minimum prices, trading volume. The service `finance.yahoo.com` also provides a value of Adjusted Closing Price (ACP), which is a stock's closing price on any given day of trading that has been amended to include any distributions and corporate actions that occurred at any time prior to the next day's open. The ACP is often used when examining historical returns or performing a detailed analysis on historical returns.

Since we view trading strategy as a classification problem solved with the help of supervised learning, it is necessary to set the target variable. We define it as follows:

$$target(t) = \begin{cases} 1, & p(t)/p(t+1) < 1 \\ -1, & p(t)/p(t+1) \geq 1 \end{cases} \quad (1)$$

where $target(t)$ is target variable (trading signal) in the time t, $p(t)$ and $p(t+1)$ are the ACP's in current (t) and next $(t+1)$ trading intervals respectively.

This means that if the asset price in the next period increases, our strategy should generate a buy signal (1). If a financial instrument has already been acquired, it is necessary to retain it. If the price decreases in the next period, then the strategy should give a sell signal (-1) if the asset is already acquired. If the asset is not bought by this time, you should refrain from buying. In other words, our system has an ambitious goal - to predict based on historical data whether the price in the next period will increase or decrease.

We will assume that the initial capital of the investor is $x(0)$. If the 'buy' signal is received, he buys the maximum possible number of shares $d(t) = v(t)p(t)$, where $v(t)$ is the amount of acquired assets, which is restricted by condition $d(t) \leq x(t)$. If a 'sell' signal is received, he sells shares at the current price and his capital becomes $x(t) = x(t-1) + v(t)p(t)$. Thus, the goal of the strategy is to maximize the sum $x(t) + d(t)$, therefore, its effectiveness can be estimated as $e = [x(n) + d(n)]/x(0)$, where *n* is the number of trading periods.

Also, we use several assumptions that are typical for research of this kind:

- the volume of sales and purchases is quite small and does not affect the behavior of the market;
- Transaction costs for operations are zero.

4.2 Features Set

We suggest including in feature set most popular indicators of technical analysis: Moving Average (MA), Moving Average Convergence Divergence (MACD), Relative Strength Index (RSI), Stochastic and signals of trading strategies based on these indicators [3,4].

Trading signals based on MA are generated as follows:

$$S_{MA}(t) = \begin{cases} -1, & MA_{n1}(t-1) > MA_{n2}(t-1) \ \wedge MA_{n1}(t) < MA_{n2}(t) \\ 1, & MA_{n1}(t-1) < MA_{n2}(t-1) \ \wedge MA_{n1}(t) > MA_{n2}(t) \end{cases}, \quad (2)$$

where $MA_n(t)$ is mean average in time *t*, *n* is widows size. Usually, $n_1 = 9$ and $n_2 = 50$, these values were determined empirically.

Trading signals based on MA are calculated as follows:

$$S_{RSI}(t) = \begin{cases} -1, & RSI(t) > UB \\ 1, & RSI(t) < LB \end{cases}, \quad (3)$$

where $RSI(t) = 100[1 - \frac{1}{1+RS(t)}]$, $RS(t) = \max_n AG \,/\, \min_n AL$, AG is average gain, AL is average loss, *n* is number of periods, UB and LB upper and lower limits respectively. Usually, in technical analysis $n = 14$, $UB = 70$, $LB = 30$.

Trading signals based on MACD are calculated as follows:

$$S_{MACD}(t) = \begin{cases} -1, & F(t-1) > S(t-1) \ \wedge F(t) < S(t) \wedge F(t) > 0 \ \wedge S(t) > 0 \\ 1, & F(t-1) < S(t-1) \ \wedge F(t) > S(t) \wedge F(t) < 0 \ \wedge S(t) < 0 \end{cases}, \quad (4)$$

where $F(t) = EMA_{n1}(t) - EMA_{n2}(t)$ is fast MACD line, $S(t) = EMA_{n3}(F(t))$ is slow MACD line, EMA_n is exponential mean average with window size n. Typically, $n_1 = 13$, $n_2 = 26$, and $n_3 = 9$. Useful information also can be extracted from MACD histogram: $Hist(t) = S(t) - F(t)$.

Stochastic signals are calculated as follows:

$$S_{stoch}(t) = \begin{cases} -1, & \%K(t) > UB \ \wedge \%D(t) > UB \\ 1, & \%K(t) < LB \ \wedge \%D(t) < LB \end{cases}, \quad (5)$$

where $\%K(t) = 100[p(t) - \min_{n1} p(t)]/[\max_{n1} p(t) - \min_{n1} p(t)]$, and $\%D(t) = MA_{n2}(\%K(t))$. Usually, $n_1 = 14$, $n_2 = 3$, $UB = 70$, $LB = 30$. Sometimes also parameter 'slow D' is used $D(t) = MA_{n3}\big(\%D(t)\big)$ with $n_3 = 3$.

4.3 R/S Analysis

The first algorithm runs, performed on the historical data of Alphabet Inc. (ticker GOOG), showed that the above indicators are not enough to build an effective trading strategy. Therefore, to search for relevant characteristics, an R/S analysis of this time series was conducted using the methodology described by Peters [15,16].

R/S analysis helps to determine the nature of price series by measuring its speed of diffusion. The speed of diffusion can be characterized by the variance [22]:

$$\langle |z(t+\tau) - z(t)|^2 \rangle \sim \tau^{2H}, \quad (6)$$

where $z(t) = \log p(t)$ is the log prices (ACP) at the time t, τ is the arbitrary time lag, and $\langle \cdots \rangle$ is average over all τ's. The ~ means that this relationship turns into equality with some proportionality constant, H is the Hurst exponent. For a price series exhibiting geometric random walk, H=0.5, for a mean reverting series, H<0.5, and for a trending series H>0.5. In last case, a future data point is likely to be like a data point preceding it, i.e. logarithms $\log[p(t)/p(t-1)]$ and $\log[p(t+1)/p(t)]$ likely will have the same signs. So, value of Hurst exponent is very valuable domain knowledge that can help to design effective algorithm.

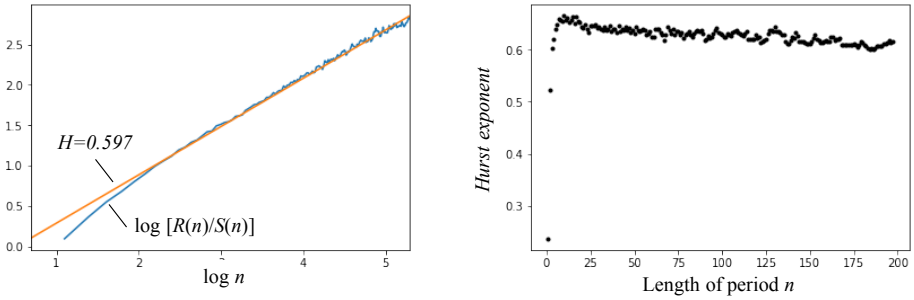
According to Peters [15,16] Hurst exponent, H, is calculated as asymptotic approximation of the rescaled range as a function of the time span of a time series as follows

$$E\left[\frac{R(n)}{S(n)}\right] = Cn^H \text{ as } n \to \infty, \quad (7)$$

where $R(n)$ is the rescaled range of first n values of $z(t)$, $S(n)$ is their standard deviation, $E[\cdot]$ is the expected value, n is the number of data points in time series, and C is a constant. Therefore, to find H, it is enough to find a regression $\log[\mathrm{R(n)/S(n)}] = \log \mathrm{C} + \mathrm{H} \log \mathrm{n}$.

The results of the Hurst exponent estimation for a series of daily closing prices (ACP) of Alphabet Inc. for 10 years (2007-2016) are shown in Fig. 2a, the dependence of its values on period length is shown in the Fig. 2b. As already mentioned, the value

$H = 0.597 > 0.5$ means that the logarithms of successive price changes are likely to have the same sign.



(a) (b)

Fig 2. Hurst exponent for the GOOG ACP time series.

Fig. 3a shows the dependence of $\log[p(t+1)/p(t)]$ on $\log[p(t)/p(t-1)]$, a significant part of the points is concentrated in areas II and IV. These values do not change sign. Fig. 3b shows the probability of changing the sign of the logarithm of the price increment, depending on the duration of the period:

$$P(\tau) = P\left[\log\frac{p(t+\tau)}{p(t)} \Big/ \log\frac{p(t)}{p(t-1)} < 0\right] \qquad (8)$$

It follows from the Fig. 3b that this probability does not never reach the value 0.5, corresponding to complete uncertainty. Thus, the investigated price range is not entirely random. Moreover, for a one day period we have $P(\log\frac{p(t+1)}{p(t)} / \log\frac{p(t)}{p(t-1)} < 0) = 0.25$, i.e. the probability that the sign of logarithm of the price ratio in the next day will not change is 0.75. It is very important domain knowledge, so we must include sign of $\log[p(t)/p(t-1)]$ in feature set.

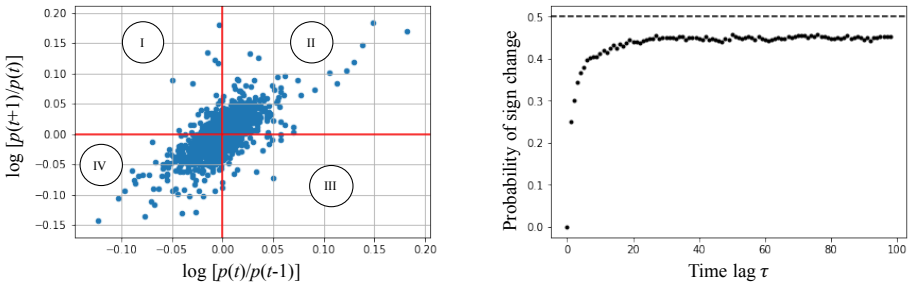


Fig 3. Probability of sign of logarithm of price ratio change.

Other useful information may be extracted form market indices, as well as logarithms of their changes. Therefore, we will also include them in the features set.

As result 41 features where selected for algorithm training including:

- Values of price series (open, close, min and max prices, volume and ACP);
- Technical indicators described in Section 4.2: $MA_9(t)$, $MA_{50}(t)$, $RSI(t)$, $F(t)$, $S(t)$, $Hist(t)$, $\%K(t)$, $\%D(t)$, $D(t)$;
- Trading signals described in Section 4.2: $S_{MA}(t)$, $S_{MACD}(t)$, $S_{RSI}(t)$, $S_{stoch}(t)$;
- Sign of $\log[p(t)/p(t-1)]$;
- Dow-Jones (ticker ^DJI), NASDAQ (^NDX) and S&P500 (^GSPC) indexes and signs of logarithms of their ratios.

To train algorithm we used two-year (2015 and 2016) daily prices of Alphabet Inc. (ticker GOOG). This training set contains 504 samples of 41 features. Test set contains 61 samples of daily data from January to March 2017.

5 Features Selection for Individual Classifiers

We use wrapping method based on genetic algorithm for features selection. The program code was developed on the Python language and based on machine learning library `scikit-learn` [23]. Therefore, only the classifiers available in this library were used as basic (kNN, NB, LR, DT and SVM).

Each classifier is coded by an array *G* with length *N*, which describes features set used for its training (*N* – quantity of features in the researched dataset). Array elements can take values 0 or 1. If the element is equal to 0, the corresponding feature is excluded from training set. Value 1 is assigned to all elements of a genotype *G* of individuals when initial population is generated. Thereby training of each classifier begins with full range of features.

The best individual in population is always copied in new population without any changes (the principle of elitism). Selection of other individuals is rank-based. Mutation operation is applied with probability p_m to randomly selected *G*-genotype element of selected individual, with its value replaced by opposite, i.e. 0 becomes 1, and 1 becomes 0. Crossover operation is applied with probability of p_c, it is implemented as exchange of randomly selected substring between two individuals.

Fitness is calculated as average classification accuracy value:

$$accuracy = \frac{TP + TN}{TP + FP + TN + FN} \tag{9}$$

where *TP* is true positives, *FP* is false positives (negatives classified as positives), *TN* is true negatives, and FN is false negatives (positives classified as negatives).

On this stage of training the researcher determines a set of classifiers types which will be used for ensemble design (denote the number of types by *M*). The classifier of each type is trained according to the algorithm described above. The set of the trained classifiers is transferred to the following stage.

Results of individual classifiers training are given in Table 1. For all classifiers, the following parameters were used: the population size 40, number of generations 20, $p_m = 0.5$, $p_c = 0.5$. Accuracy before training is calculated before start of a genetic algorithm (the features set includes all 41 features). Widely known models kNN, LR, NB, DT and SVM are used as the basic classifiers with the default parameters of scikit-learn library. The average accuracy and subsequent confidence interval after training are given in the column with caption "Accuracy after wrapping". Also, Table 1 lists the number of selected features and precision / recall values.

Table 1. Results of base classifiers training.

Classifier	**Accuracy**		**Number of selected features**	**Precision / recall**
	Before wrapping (N=41)	After wrapping		
KNN	0.526	0.558 ± 0.100	21	0.581 / 0.632
LR	0.536	0.585 ± 0.061	14	0.589 / 0.624
NB	0.542	0.577 ± 0.020	16	0.598 / 0.614
DT	0.530	0.530 ± 0.050	17	0.565 / 0.543
SVM	0.552	0.552 ± 0.059	18	0.595 / 0.612

As it follows from Table 1, features wrapping improves performance of classifiers, but their accuracy remains low, slightly bigger than 0.5.

6 Multi-Classifier System Training

At the third stage ensemble with majority voting rule is designed from the set of the classifiers trained at the previous stage. The GA is used again. The ensemble is coded by an array w of M real numbers, $w_i \geq 0$. They set value of weight coefficient to corresponding classifier. During creation of initial population, the elements w_i are initialized as random numbers with the normality condition $\sum w_i = 1$.

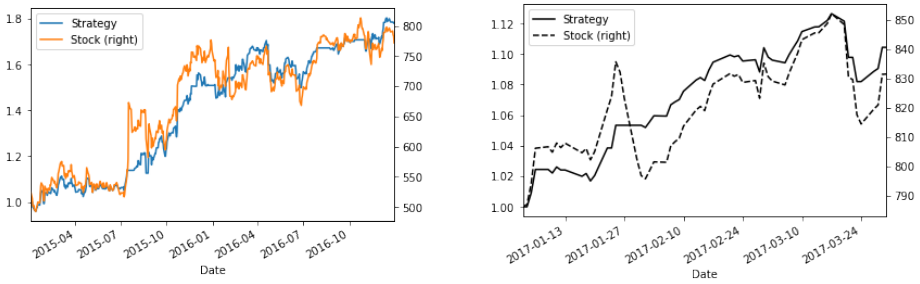
Selection rules are the same as at the previous stage: elitism and rank selection. Mutation operation is applied to all elements w_i of the selected individual, their values are randomly changed by the uniformly distributed number (-0.1; 0.1). If negative w_i is received as the result, it is replaced by 0. At the same time the normality condition isn't satisfied. These parameters were determined during experimental launches of the algorithm. Crossover operation is like the crossover at the first stage. Ensemble fitness is calculated as the accuracy. Object class C_E is calculated as the weighed sum of outcomes c_i of individual classifiers $C_E = \sum_{i \in M} w_i c_i$.

If the sign of C_E matches the object type ($C_E > 0$ means that next price will growth, $C_E < 0$ means that this price will go down), then an object is considered as recognized correctly. Absolute value $|C_E|$ corresponds with confidence of classification.

After training, ensemble with accuracy 0.744, precision 0.698, and recall 0.880 was received. This accuracy value notably outperforms the accuracy of individual classifiers.

7 Trading Strategy Results

The results of back testing of generated strategy on daily prices of Alphabet Inc. shares (ticker GOOG) in two-year period 2015 – 2016 is presented on Fig. 4a. Proposed algorithm gives the return $e = [x(n) + d(n)]/x(0) = 1.781$, this result outperforms market growth, which is 1.48. To check real possibility of proposed strategy to generate profit, another test was conducted on test set (Fig. 4b), return is $e = 1.104$.



(a) return on train set (back testing) (b) return on test set

Fig 4. Tests of generated trading strategy.

To check prediction performance of proposed method, few other well-known ensemble methods were tested on the same training and test set (Table 2). We tested five Bagging algorithms (on base DT, kNN, NB, LR, and SVC), three Adaptive Boosting algorithms (on base DT, SVC, and NB) and three other methods (Gradient Boosting, Random Forest, and ExtraTrees).

Table 2. Comparison of different ensemble techniques.

Method	**Training set**			**Test set**		
	accuracy	precision	recall	accuracy	precision	recall
BAGGING						
DT	0.988	0.985	0.992	0.508	0.643	0.474
kNN	0.687	0.676	0.744	0.377	0.500	0.316
NB	0.575	0.585	0.585	0.590	0.651	0.737
LR	0.563	0.565	0.643	0.574	0.667	0.632
SVC	0.575	0.585	0.589	0.492	0.667	0.368
ADABOOST						
DT	1.000	1.000	1.000	0.459	0.609	0.368
SVC	0.512	0.512	1.000	0.623	0.623	1.000
NB	0.569	0.575	0.609	0.541	0.625	0.658
OTHER TECHNIQUES						
Gradient Boosting	0.978	0.981	0.977	0.459	0.619	0.342
Random Forest	0.980	0.992	0.969	0.492	0.684	0.342
ExtraTrees	1.000	1.000	1.000	0.443	0.577	0.395
Proposed Method	0.744	0.698	0.880	0.590	0.633	0.816

As follows from Table 2, some techniques outperform proposed method on training set, but it shows better results on the test set. It means that proposed method more effectively avoids overfitting. Better results on test set are shown by Bagging on base NB and Adaboosting on base SVC. Comparison of trading strategies based on these two techniques and proposed algorithm is presented on Fig.5. Presented data show that proposed algorithm provides better results (return of Bagging + NB is 1.071, return of AdaBoost + SVC is 1.057, return of proposed method is 1.104).

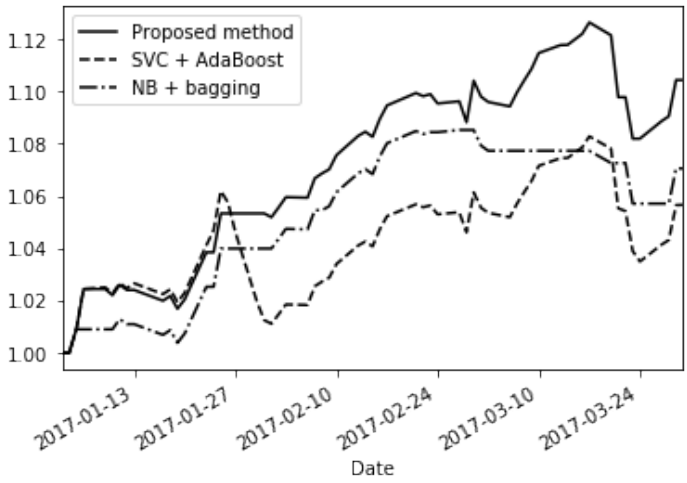


Fig 5. Comparison of three trading strategies.

To check capability of proposed algorithm to generate profit for assets of different companies and industries, test on securities of other companies was carried. We used two-years (2015-2016) daily data of 10 companies from 5 industries for training and three months' data (January-March 2017) for testing. Results are presented in Table 3, including accuracy of ensemble on train and test sets, stock prices changes $p(n)/p(0)$, where n is the length of price series, and return e as it define above.

The obtained results confirm that the presented algorithm ensures successful trade irrespective of the type of industry both in the growing and falling markets. This means that it can be used as the alpha model in the Portfolio Construction Model [1,2].

8 Parallel Implementation

The proposed algorithm provides good results on daily data, it also can be used on data of shorter periods. In its essence, the proposed algorithm identifies the trading orders through reverse engineering of observed quotes. It is so called market microstructure trading, and many authors suggest that typical holding period for such kind of strategies should not exceed 10 minutes [24]. Moreover, return of discussed AT strategy can be improved, first, by including additional basic classifiers (e.g. different models of ANN),

and second, by increasing the size of the population and the number of generations on training stages. But time of calculations with presented parameters on 2-core 1.5 GHz CPU is approximately 20 min, it should be extremely reduced to work on shorter trading intervals with extra types of base classifiers and larger populations.

Table 3. Performance of proposed method on different assets.

Company	Industry	Ensemble accuracy		Back testing (train set)		Real testing (test set)	
		Test set	Train set	Price change	Return	Price change	Return
Alphabet	ITC	0.744	0.590	1.502	1.781	1.055	1.104
Amgen	Pharma	0.785	0.519	0.969	2.491	1.096	1.109
Apple	ITC	0.709	0.607	1.133	11.173	1.242	1.246
Exxon Mobile	Oil	0.714	0.541	0.932	8.300	0.799	1.168
General Electric	Manuf.	0.750	0.507	1.371	2.194	0.948	1.003
Gilead Sciences	Pharma	0.881	0.516	0.766	11.436	0.923	1.001
HSBC	Finance	0.889	0.508	0.997	7.296	1.024	1.042
JPMorgan Chase	Finance	0.775	0.514	1.497	10.453	1.013	1.040
Shell	Oil	0.765	0.581	0.981	2.913	0.972	1.013
United Techn.	Manuf.	0.877	0.505	1.019	1.970	1.018	1.027

As it was noted above, algorithm was realized on Python programming language, because there are lot of tools of machine learning around it. It helps to reduce time for algorithm design and testing, but as Python code does not compiled to native CPU code, there are possible performance problems.

The most applicable approach without code rewriting is usage of parallel capabilities of `ipython` library [25] and multi-core system. Several tests with different number of CPU cores were executed, to determine if it is possible to reach the required performance within the `ipython` framework. Fig. 6 presents a test environment, which includes server with eight 3,5GHz CPU cores and client computer, both connected to trusted network.

On remote multi-core computer, several instances of IPyton engine were started, according with number of CPU cores used in test. The IPython engine is a regular Python interpreter that handles incoming and outgoing Python objects sent over a network connection. All program modules required for computation were located on local disks of server. IPython controller and client interface were ran on client computer.

As it follows from Fig.1 there are a few opportunities to parallelize program code. First, it is possible to parallelize individual classifiers training, because they are trained independently. Second, there is possibility to parallelize genetic algorithms where they used. Both features were used on base of IPython engine direct interface that provides the possibility directly manage computation on each engine (without automatic load balancing), because it required a small correction of the source code.

Results of tests performed on 8-core system are presented on Fig.7, which shows relative computational time (time of computation on one CPU core is 1). It is evident that the system with 4 cores provides the performance that satisfies the requirements of market microstructure trading (computational time is less than 10 minutes [24]).

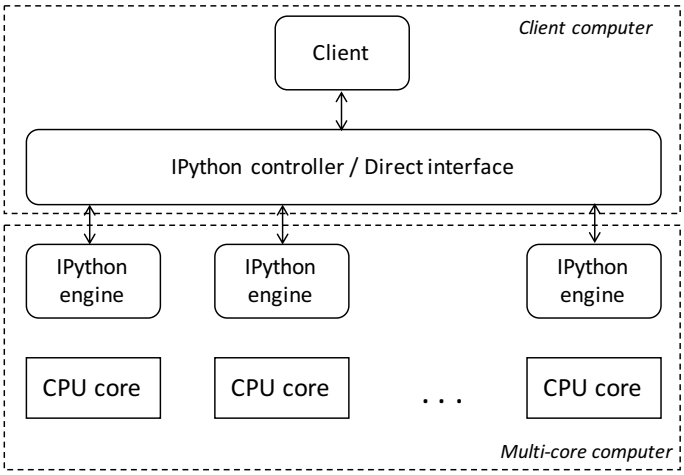


Fig 6. Test environment to evaluate algorithm performance

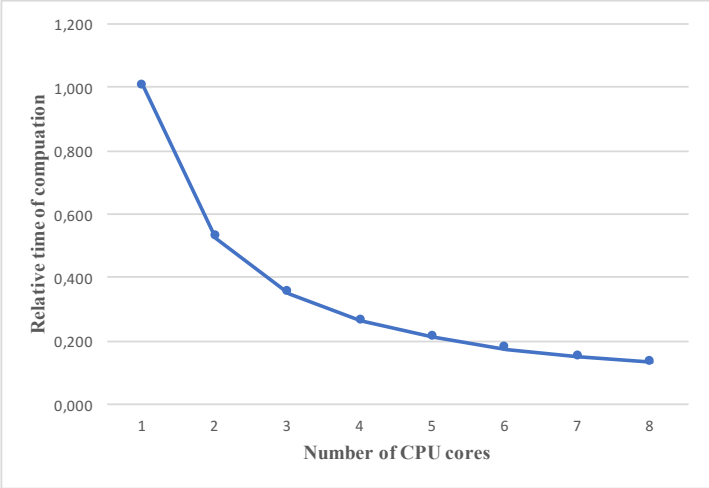


Fig 7. Reduction in computation time as a function of the number of CPU cores

However, from Fig. 7 it also follows that further possibilities for increasing performance with this approach are exhausted. To compute in shorter time intervals (1 minute and less), it is necessary to implement the algorithm in the programming language that allows more efficient use of computer resources.

9 Conclusion

The presented results show that the proposed algorithm allows to build a trading strategy that stably generates positive return regardless of the behavior of the stock market (growth or decline). This can be explained by the two reasons. The first is the domain knowledge, which was used for features engineering. The second is the use of the multi-classifier system, which combines enough simple classifiers, it helps notable improve the prediction of price behavior.

Note, that the ways to improve this algorithm are obvious. It is the inclusion of additional classifier models in the ensemble, as well as an extension of the search space when using the genetic algorithm.

Using the parallel capabilities of the ipython allows to reduce the computation time to 10 minutes or less. However, further performance improvement will require a transition to another programming language.

References

1. Nuti, G., Mirghaemi, M., Treleaven, P. and Yingsaeree, C.: Algorithmic trading. IEEE Computer 44(11), 61–69 (2011).
2. Treleaven P., Galas M., Lalchand V.: Algorithmic trading review. Communications of the ACM 56(11), 76-85 (2013).
3. Murphy, J.J.: Technical Analysis of the Future Markets. Prentice Hall, New York (1986).
4. Schwager, J.D.: Technical Analysis. Wiley, New York (1996).
5. Zhang, H., Ren, R.: High frequency foreign exchange trading strategies based on genetic algorithms. In: Proc. 2nd International Networks Security Wireless Communications and Trusted Computing (NSWCTC) Conference, vol. 2, pp. 426–429. (2010).
6. Montana G., Parrella F.: Learning to trade with incremental support vector regression experts. In: Corchado E., Abraham A., Pedrycz W. (eds) Hybrid Artificial Intelligence Systems. HAIS 2008. LNCS, vol. 5271. Springer, Heidelberg (2008).
7. Laskov, P., Gehl, C., Kruger, S.: Incremental support vector learning: analysis, implementation and applications. Journal of machine learning research 7, 1909–1936 (2006).
8. Wang, W.: An incremental learning strategy for support vector regression. Neural Processing Letters, 21:175–188 (2005).
9. de Oliveira, F. A., Nobre, C. N., Zarate, L. E.: Applying artificial neural networks to prediction of stock price and improvement of the directional prediction index–case study of PETR4, Petrobras, Brazil. Expert Systems with Applications, 40(18), 7596-7606 (2013).
10. Evans, C., Pappas, K., Xhafa, F.: Utilizing artificial neural networks and genetic algorithms to build an algo-trading model for intra-day foreign exchange speculation. Mathematical and Computer Modelling. 58(5), 1249-66 (2013).
11. Yu, L., Wang, S., Lai, K.: A novel nonlinear ensemble forecasting model incorporating GLAR & ANN for foreign exchange rates. Computers and Operations Research 32, 2523–2541 (2004).
12. Scabar, A., Cloete, I.: Neural networks, financial trading and the efficient market hypothesis. In: Oudshoorn, M. (ed.) XXV Australasian Computer Science Conference (ACSC2002), Melbourne, Australia., vol. 4, (2002).

13. Butler, M., Daniyal, A.: Multi-objective optimization with an evolutionary artificial neural network for financial forecasting. In: GECCO '09 Proceedings of the 11th Annual conference on Genetic and evolutionary computation, Montreal. Canada, pp. 1451-1458 (2009).
14. Tsai, C. F., Chiou, Y. J.: Earnings management prediction: a pilot study of combining neural networks and decision trees. Expert Systems with Applications, 36(3), 7183-7191 (2009).
15. Peters E.E.: Chaos and order in the capital markets. Wiley, New York (1996).
16. Peters E.E.: Fractal market analysis: applying chaos theory to investments and economics. Wiley, New York (2003).
17. McNeils, P.D.: Neural networks in finance: gaining predictive edge in the market. Elseiver, Amsterdam (2005).
18. Han, J., Kamber, M., Pei, J.: Data mining: concepts and techniques. Morgan Kaufmann, Waltham (2012).
19. Mukhopadhyay, A., Maulik, U., Bandyopadhyay, S. C.: A survey of multiobjective evolutionary algorithms for data mining: Part I. IEEE Transaction on Evolutionary Computation, 18(1), 4-19 (2014).
20. Wozniak, M., Grana M., Corchado, E.: A survey of multiple classifier systems as hybrid systems. Information Fusion, 16(1), 3-17 (2014).
21. Kuncheva, L.: Combining pattern classifiers: methods and algorithms. Wiley, New York (2004).
22. Chan, E.P.: Algorithmic Trading: Winning Strategies and their Rationale. Wiley, Hoboken (2013).
23. Pedregosa, F., et al.: Scikit-learn: machine learning in python. Journal of Machine Learning Researc 12, 2825-2830 (2011).
24. Aldridge, I. High-frequency trading: a practical guide to algorithmic strategies and trading systems. Wiley, Hoboken (2009).
25. Pérez, F., Granger, B. E.: IPython: a system for interactive scientific computing. Computing in Science & Engineering, 9(3), (2007).