A modification of Random Forest investment assets selection algorithm

Anna Zinenko*, and *Alena* Stupina

Siberian Federal University, 79, Svobodny Prospect, Krasnoyarsk, 660049, Russian Federation

Abstract. The paper presents selecting stocks for an investment portfolio algorithm, alternative to standard mathematical programming optimal portfolio methods. To implement the algorithm, a modification of the Random Forest machine learning model is proposed. At the first step, the algorithm builds a decision tree based on forecasts using the naive method and the ARIMA method, and then it forms a "forest" of trees from random subsamples. The algorithm was tested on different time intervals on the instruments of two exchange indices. Its implementation showed good results - at least 78% of the selected stocks increased in price over the forecast period.

1 Introduction

The forming an investment portfolio problem is one of the key areas of investment analysis. The goal of both private and institutional investors is to get a return on invested funds with minimal risk. The measure of risk is uncertainty, and mathematically the risk of investments can be estimated as the deviation of the return from the expected value. The expected value of return in portfolio theory is arithmetic mean of returns for past periods, and the risk as dispersion or standard deviation. Portfolio optimization can occur in two directions: either minimizing risk (dispersion) for a given return (arithmetic mean), or maximizing return for a given risk. The first calls direct, and the second – the inverse Markowitz problem. This is a standard problem of mathematical programming, where (in the case of a direct problem) the objective function is risk, and the constraint is return. The sum of the shares must be equal to one. The Markowitz model is represented by equation 1.

$$\begin{aligned} V_p = X^T V X \rightarrow \min, \\ M^T X = m_p, \\ I^T X = 1, \end{aligned} \tag{1}$$

where V_p – objective function (portfolio dispersion), V – covariation matrix, X – column matrix of asset shares, M – column matrix of expected returns on assets included in the portfolio, I – identity column matrix, m_p – given value of the portfolio return.

This problem can be solved by the standard mathematical programming methods. The final decision presents work [1] and it looks this way:

$$X^* = V^{-1} \frac{m_p(I J_{12} - M J_1) + M J_{12} - I J_2}{J_{12}^2 - J_1 J_2} \tag{2}$$

* Corresponding author: anna-z@mail.ru

$$J_{12} = I^T V^{-1} M, J_1 = I^T V^{-1} I, J_2 = M^T V^{-1} M$$

Despite its prevalence, the Markowitz model has some disadvantages. First of all, the standard optimization method selects only a small number of assets from the entire set. Thus, when analyzing an index consisting of hundreds of assets, the model can select only units, which is bad from the point of view of diversification. Another disadvantage of the model is its strong sensitivity to initial conditions [2]. With a small change in the variance of one of the assets, the portfolio structure can change significantly.

We propose selecting shares for a portfolio without optimization, in other words, a binary classification - we include a share in the portfolio or we do not. The developed by us method for forming an investment portfolio "Modified Random Forest" includes the following stages.

- Building forecasts using the ARIMA and Naive Forecast methods;
- Building a decision tree based on the obtained forecasts;
- Generating decision trees on random subsamples of time intervals and outputting the most frequently encountered stock trees in forecasts as the result;
- Checking the accuracy of the algorithm.

2 Methods

2.1 ARIMA Model

The Autoregressive Moving Average Integrated Model (ARIMA) is an autoregressive moving average model (ARMA(p,q)) that is applied not to the values of the series levels, but to the differences of order d [3,4]. The ARIMA (p, q) model is represented by the following equation

$$\Delta_t^d = \sum_{i=1}^{p} \alpha_i \Delta_{t-i}^d + \varepsilon_t + \sum_{i=1}^{p} \beta_i \varepsilon_{t-i} \quad (3)$$

where Δ_t^d – the difference of d-order, Δ_{t-i}^d – the difference of d-order, laggard by lag i, ε_t – moving average forecast error, ε_{t-i} – moving average forecast error laggard by lag i., α_i, β_i – model parameters.

While constructing ARIMA and ARMA models, the problem of determining the error ε_t (noise component) arises, We proposed the solution to this problem in [5]. For forecasting on stock markets, the ARIMA model of orders (1, 1, 1) is sufficient.

2.2 Decision Tree

The decision tree algorithm is machine learning method for classification and regression problems. The classification problem is to divide the final feature into two or more categories. Such problems include, for example, the decision to issue a loan, the selection of gifted students, etc. The problem of choosing shares for an investment portfolio that we formulated also relates to the classification problem. At the tree training stage, the final feature is already divided into categories. One of the factor features is selected as the root node of the tree, and then the tree branches out depending on the feature values. Then, other factor features are selected in the following nodes. The process is repeated until the tree reaches a given depth. The last level of the tree provides the final division into categories, and its nodes call leaves. Each node contains information about the number of sample elements in it, as well as an error metric. For classification, such metrics are entropy and the Gini coefficient [6], and for regression, the deviation indicators of predicted values from actual ones, such as the mean square error, average absolute error, etc. The tree depth must be chosen such that the error stops decreasing. For the classification task, both categorical and quantitative features can be located in the nodes of the decision tree.

After training, the decision tree is tested on a test sample. The actual values of the final feature are not givan to the algorithm, but are compared with the values predicted by the tree. At this stage, the error matrix is calculated for the classification tree, as well as the True Positive Rate, True Negative Rate, and Accuracy metrics. The calculation of these metrics is presented in equations 4-6.

$$\mathrm{TPR} = \frac{TP}{TP+FP}, \tag{4}$$

$$\mathrm{TNR} = \frac{TN}{TN+FN}, \tag{5}$$

$$Accuracy = \frac{TP+TN}{TP+FP+TN+FN}, \tag{6}$$

where TP – the number of sample elements that the algorithm classified as positive, and they are actually positive, FP – κ the number of sample elements that the algorithm classified as positive, and they are negative, TN – the number of sample elements that the algorithm classified as negative, and they are actually negative, FN – is the number of sample elements that the algorithm classified as negative, and they are positive.

An example of a decision tree constructed by the R programming language built-in functions is shown in Figure 1.

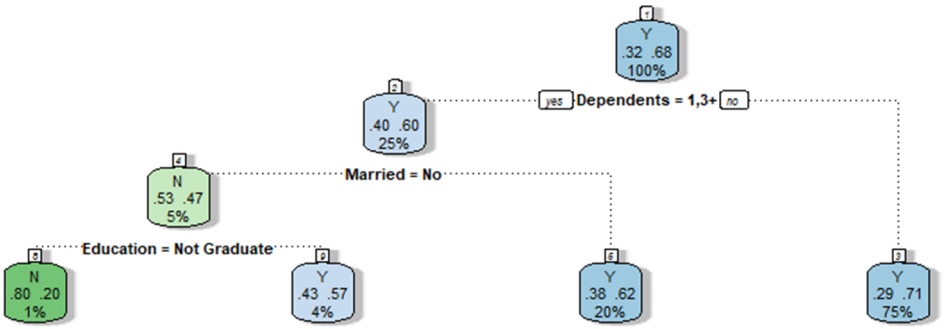

Fig. 1. An example of Decision Tree.

The tree nodes show the number of positive and negative answers, as well as the number of elements that match the conditions shown on the tree branches. The algorithm trains on actual data and then makes decisions on the test sample. The decision tree itself is a weak classification and forecasting algorithm. To improve the quality, ensemble methods [7] are used, based on the construction of many decision trees. Such algorithms include random forest and gradient boosting.

2.3 Random Forest

The main idea of the Random Forest algorithm is to build an ensemble of weak decision algorithms – decision trees and select, in the case of classification, those objects for which the largest number of trees vote, and in the case of regression – the average value for all trees. The algorithm is based on the ensemble bagging method [8] – building identical models, the results of which are averaged (in the case of classification, the results that received the most votes are selected).

Random subsamples from the original set of features and data are generated by the bootstrap method [8] – random generation from the original sample of a set of pseudo-samples, which necessarily contain values from the original sample, but they can be repeated. The first step of Random Forest algorithm is bootstrap: the algorithm generates pseudo-samples from the original sample, and build a decision tree for each of them. The splitting

features are also randomly selected in each tree. When implementing the algorithm, the number of trees, the number of features for training, the learning rate and the tree depth are specified. The criteria for assessing the accuracy of the algorithm are the same as for single decision trees.

The random forest method is still successfully used in classification and regression problems [9]. Random forest shows strong results in all machine learning problems except image recognition. Currently, an alternative Gradient Boosting algorithm is gaining popularity - sequential construction of decision trees, each subsequent one giving better results than the previous one. In our opinion, advantage of Random Forest over Gradient Boosting is the simplicity of independent development without using built into software packages libraries, which makes it possible to modify the algorithm for specific tasks.

2.4 Modified Random Forest

The algorithm was developed by the author in the Python programming language. To calculate the accuracy metrics (TPR and TNR), we divided original sample into training and testing. 80% of the original sample is training period, 20% is testing period. We made the forecast at the end of the testing period. First, we created a function that made a forecast using the ARIMA method, which was presented and tested in detail in our previous works [5]. Then, we constructed a function that made a forecast using the naive method (the arithmetic mean of the training sample). An example of a forecast using the naive method and the ARIMA method for Gazprom shares for the period from February 2023 to February 2024 is presented with Figures 2 and 3.

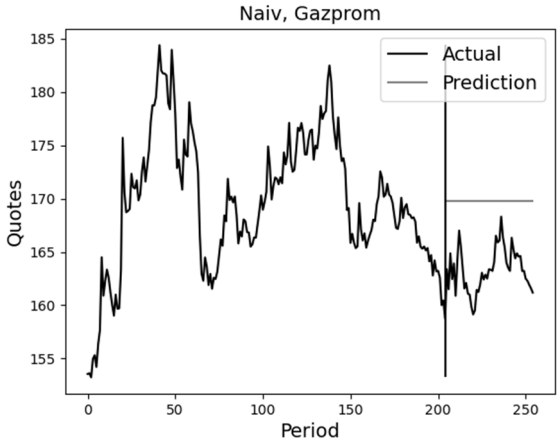

Fig. 2. An example of naïve forecast.

The ARIMA method shows the better results. Thus, according to the illustrated example of Gazprom shares, ARIMA showed an average error of 0.5%, while the naive forecast was 4%. However, to branch the tree, we need at least two methods. Forecasting methods that are more complex than the naive forecast, as well as the inclusion of a larger number of methods, will significantly increase the execution time of the algorithm, so we settled on the two methods described above.

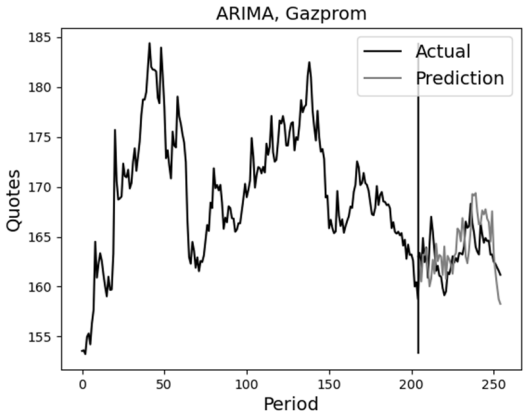

Fig. 3. An example of ARIMA forecast.

The decision tree looks as follows: at the root of the tree is a naïve forecast (arithmetic mean). If the forecast showed a fall in the stock price, we threw out the stock from the portfolio. Otherwise, the stock moved to the next level of the tree for making the ARIMA forecast. If the forecast showed growth, the stock remained in the portfolio, otherwise it was thrown out. The tree structure is shown in Figure 4. The tree function at the output returned two lists of stocks: accepted and not accepted into the portfolio.

For determining which of the two methods to place at the root of the tree, we used the Gini coefficient This coefficient shows the degree of uncertainty. We chose the method with less Gini coefficient value.

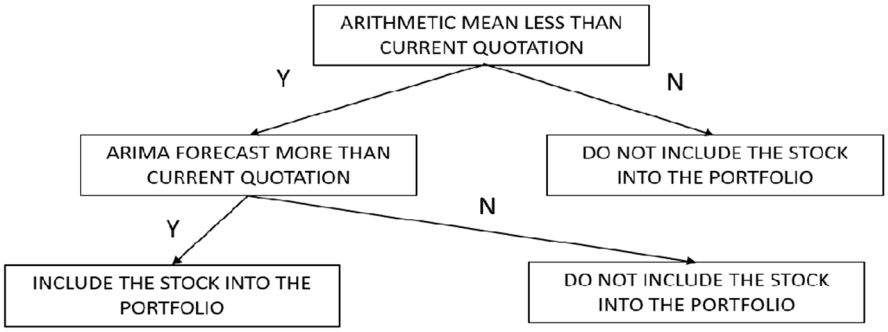

Fig. 4. Decision Tree for including stocks into portfolio.

Then we should construct a random forest from the trees. The forest must meet two conditions. First, since we had only two factors to build the tree on, we could not select factors randomly, and both factors had to be taken into account on each tree. Second, random subsamples must consist of successive elements of the initial sample, since we were making a forecast.

To generate random subsamples taking into account the specified conditions, we used two random numbers: the index of the initial sample k, starting from which we select a subsample, and the length of the subsample l. Obviously, the length of the subsample should not exceed the difference between the length of the initial sample and k. We also set the value of l to be no less than 50. 300 subsamples were generated, and the decision tree was built for each one. Then we count the "votes" of the trees for each stock included in the index. If the

number of "votes" was less than the median value, we threw the stock out of the portfolio, otherwise – accept into the portfolio. All the above actions were performed on the training sample. Next, in order to check the accuracy of the algorithm, it is necessary to compare the last price value of the selected shares on the test sample with the last value on the training sample. If the first exceeds the second, then we put the TRUE mark, otherwise - the FALSE mark. After putting the marks on all the index shares, the True Positive Rate and True Negative Rate metrics were calculated.

3 Results and discussion

We tested our algorithm on Moscow Exchange indices (47 instruments) and NASDAQ (231 instruments) stocks. We developed special functions to obtain quotes from exchanges and to interact with the Moscow Exchange API and Yahoo Finance. It was important to test the algorithm on different time periods and time intervals, because testing on one or two periods can be just getting into the market growth. For this reason, we took three-time intervals: 2021-2022, 2022-2023 and 2023-2024, with a six-month and annual interval for each. Table 1 presents the results of applying the Modified Random Forest algorithm in the form of True Positive Rate and True Negative Rate metrics, as well as the number of stocks selected for the portfolio.

Table 1. Results of applying the Modified Random Forest algorithm.

Indicator	Period					
	16.02.23 – 16.02.24	01.03.22 – 01.03.23	26.12.21 – 26.12.22	16.06.23 – 16.02.24	01.07.22 – 01.03.23	26.06.21 – 26.12.22
TPR, MOEX, %	78	95	82	89	87	70
TPR, NASDAQ, %	87	49	85	77	49	81
TNR, MOEX, %	17	12,5	11,5	25	29	41
TNR, NASDAQ, %	15	49	18	23	52	17
Quantity of accepted stocks, MOEX	23	23	22	19	21	23
Quantity of accepted stocks, NASDAQ, pcs.	114	113	113	113	114	111

As Table 1 shows, in most cases the algorithm showed good results - it selected at least 78% of profitable stocks for the portfolio. However, for the NASDAQ index, the results were weak over two time periods - only half of the selected stocks were profitable. For 2022 - 2023, this situation can be attributed to a market decline, since most of the stocks included in the index are American. Also, the NASDAQ index is mainly represented by the high-tech industry, which makes it poorly diversified. According to the True Negative Rate metric, the situation is much worse. In most cases, it does not reach 50%, which means a significant shortage of obviously profitable stocks. As we indicated above, this metric is not as important for selecting stocks for a portfolio as the True Positive Rate, but its values should be at least more than 50 - 60%.

As a final step of our work, we compare the Modified Random Forest method with the classic method of forming an investment portfolio – the Markowitz problem. We also developed algorithm in Python for this problem. The optimal solution to the problem was formed using formula 2.

We did not present the solution to the problem in the classical form – we need binary responses (to take / not to take the stock into the portfolio). Thus, we did not calculate the shares of the selected stocks, but only checked the stocks included in the Markowitz portfolio using the True Positive Rate and True Negative Rate metrics. Table 2 shows indicators similar to those presented in Table 1, obtained when applying the Markowitz problem.

Table 2. Results of applying the algorithm "Markowitz Problem".

Indicator	Period					
	16.02.23 – 16.02.24	01.03.22 – 01.03.23	26.12.21 – 26.12.22	16.06.23 – 16.02.24	01.07.22 – 01.03.23	26.06.21 – 26.12.22
TPR, MOEX, %	72	62,5	43	57	78	34,5
TPR, NASDAQ, %	70,4	80	55	71	43	20,5
TNR, MOEX, %	25	20	-	39	35,7	62,5
TNR, NASDAQ, %	21,2	13	46	34	59,6	81,5
Quantity of accepted stocks, MOEX	32	24	48	45	33	48
Quantity of accepted stocks, NASDAQ, pcs.	108	105	117	114	116	112

According to Table 2, the results of the classical model are not better, and in many cases worse, than the results of our model. The True Positive Rate metric is better only for the period from 03/01/2022 to 03/01/2023 for the NASDAQ index. The True Negative Rate metric showed good results for the period from 06/26/2021 to 12/26/2022 for both indices and for the period from 07/01/2022 to 03/01/2023 for the NASDAQ index. In general, the True Negative Rate metric using the Modified Random Forest method is slightly worse than using the classical Markowitz method. At the same time, the True Positive Rate metric is significantly better in all periods and intervals. For selecting stocks for a portfolio, the True Positive Rate metric is more important, since it is better to skip profitable stocks than to include unprofitable ones in a portfolio.

We can come to conclusion that different methods select different stocks for a portfolio, regardless of which direction the market is moving. Thus, during the most critical period presented for our country from 2021 to 2022, the Modified Random Forest method collected a profitable portfolio, and the Markowitz method showed the best percentage of discarded unprofitable stocks. It follows that even when the market falls, we can find working methods, such as the Modified Random Forest algorithm we presented. As noted above, the good results of this method during a decline is because the algorithm was tested on different time intervals randomly selected from the original, which can show both growth and decline. However, despite all the strengths of our algorithm, it shows a weak True Negative Rate, so in the future we plan to either refine this algorithm or create a new one based on it.

4 Conclusion

The main idea of the paper was an approach to forming a portfolio based on a modern machine learning method – Random Forest. As factors determining decision trees, we took the naive forecast by the arithmetic mean and the forecast by the ARIMA method. This algorithm, like the well-known knapsack method, unlike classical methods, does not determine the weight of shares in the portfolio, but determines whether to include a share in

the portfolio. We implemented the algorithm in Python and tested it on two indices –Moscow Exchange index and the NASDAQ index on three time intervals for periods of one year and six months. On most of the selected time intervals, the algorithm showed good results. At the same time, the less important metric True Negative Rate showed poor results. According to the classic Markowitz model, which we used for comparison on the same indices and time intervals, the True Negative Rate metric showed slightly better results, but the more important True Positive Rate metric turned out to be significantly worse. From this we can conclude that the developed algorithm needs some refinement, but at the same time it is at least no worse than the classic method, which requires large calculations and is also difficult to implement when automatically sorting through a large number of shares.

This work was supported by the Ministry of Science and Higher Education of the Russian Federation (Grant No.075-15-2022-1121).

References

1. Yu. F Kasimov, *Financial investments* (Ankil, 2008)
2. I. S. Ivanchenko, D. D. Osey. Financial Journal, **1**, 41 (2018)
3. J. Box, G. Jenkins, *Time Series Analysis: Forecast and Control* (Mir, 1974)
4. V. V. Vyugin, *Mathematical foundations of machine learning and forecasting* (MCMNO, 2018)
5. A. V. Zinenko, A. A. Stupina, ITM Web of Conferences **59**, 02005 (2024)
6. D. Carter, Business Information Review, **37(2),** 60-68. https://doi.org/10.1177/0266382120923962
7. H. Brink, D. Richards, M. Feverloff, *Machine Learning* (Piter, 2017)
8. A. L. Marchenko, *Python: a big book of examples* (Moscow University Publishing House. 2023)
9. J.-Sh. Chou, K. -E. Chen. Appl. Soft Comput., **158** (2024) https://doi.org/10.1016/j.asoc.2024.111567