Comparing Profitability of Different Machine Learning Algorithms in Financial Markets

Akshat, Anurag Gulati, Ayush Madan, Kabir Baghel
Indraprastha Institute of Information Technology, Delhi
{akshat20172, anurag20176, ayush20187, kabir20564}@iiitd.ac.in

Abstract

This project analyzes the trading signals generated by different machine learning algorithms and compares their potential profits to the original strategy given in the dataset and to each other. The learning and testing have been done specifically for the Indian stock market, but one can apply them to other asset classes and markets too. The sole performance metric for evaluating these algorithms is the profit generated in a given time frame.
GitHub Repository

1. Introduction

Until the late 1990s, trading decisions across all asset classes entirely depended on a person's analysis. Around the turn of the century, with the advent of faster machines, trading was automated with hard-coded strategies. Around the same time, machines learned to generate trading signals based on past data and strategies using various learning algorithms. Since the mid-2000s, researchers have been developing intelligent trading systems using countless learning algorithms.

Traders worldwide rely on fundamental and technical analyses while on the trading floor of an exchange. For instance, in stock markets, the former is based on understanding the financial records of a given company. In the forex market, it deals with the underlying economic and political policies of the two nations. In contrast, demand and supply analysis can be taken under the fundamental umbrella of the commodities market. On the other hand, technical analysis always deals with the patterns and statistics of price movement in any market. These statistical parameters are standard across asset classes, making technical analysis consistent.

Over the decades, researchers have employed machine learning algorithms to generate profitable trading signals. This project provides a comparative study on the profitability of some of these algorithms in the Indian stock market. In order to have enough data for learning, we must choose a stock that is old enough and holds a strong position in the market. For the same reasons, we have chosen the Tata Consultancy Services (TCS) data in the Bombay Stock Exchange.

1.1. Related work

Since automated trading has been a center of interest for many trading firms, the amount of dedicated research in this field is ever-growing. This calls for a literature review of the existing methods. Lin [1] proposed a modified version of the old filter rule [12] by introducing three decision variables that would be representative of fundamental analysis. The modified rule outperformed the filter rule in the Taiwan's stock market. Lin's method didn't incorporate the future information while clustering the trading points. Wu, Lin, and Lin [2] overcame this problem in their research, which was tested on both Taiwan's market and NASDAQ. Their method outperformed the filter rule as well as Lin's method. Teixeira and Oliveira [3] have used a nearest neighbors classifier built on conventional technical analysis. They exploited the k-NN classifier and tested its feasibility in a real market scenario.

The most recent developments in the field have been using deep learning for signal generation. Fernandez-Rodrıguez, Gonzalez-Martel, and Sosvilla-Rivero [4] investigate the profitability of artificial neural networks (ANNs) against the buy-and-hold strategy in various market scenarios. Their research suggests that, in the absence of trading costs, the technical trading rule is always superior to a buy-and-hold strategy for both bear market and stable market episodes. The strategy was tested on the General Index of the Madrid Stock Exchange. Chang, Liu, Fan, Lin, and Lai [5] proposed the use of ensemble learning based on neural networks. Their work demonstrates a hybrid model which, first, locates all the turning signals in the past data and uses them to train an ensemble neural network. The trained model is then deployed to predict the turning signals in the future and use those to automate the trading process.

More advanced studies focus on using genetic algorithms and genetic programming for trading. Incorporating news-based analysis using natural language processing is another approach that, when combined with one of the above methods, can give out a mix of fundamental and technical analysis. However, these advanced techniques are well beyond the scope of this project.

The rest of this report has been is divided as follows. Section 2 introduces the concept of technical indicators and goes on to explore the ones that are popularly used. Section 3 lays down a description of the dataset and summarizes the exploratory data analysis (EDA), that generated some interesting insights into the same. This section also selects the most relevant features based on the EDA, which are used for training and testing of the learning algorithms. Section 4 provides a brief about the different learning algorithms that have been explored in this work. Section 5 explains the experimental setup done to deploy the models and presents the results obtained from them. Finally, we provide the conclusions and future directions for the research.

2. Technical analysis

Murphy [6] defined *technical analysis* as *the study of market action, primarily through the use of charts, for the purpose of forecasting future price trends*. The whole concept of technical analysis is built around statistics. Using the past data of a given security, one can compute some heuristic quantities, known as *technical indicators*, to analyze the history of price action and forecast the future movement of the same. Throughout the 20th century, statisticians have developed a lot of technical indicators and categorized them on the basis of the underlying feature observed. For instance, the indicators that capture the momentum of price are termed as *momentum indicators*. Following are the popular categories of these indicators along with the particular indicators that we have used to construct the dataset for this project:

1. *Trend indicators*:
   (a) Commodities Channel Index (CCI)
   (b) Exponential Moving Average (EMA)
   (c) Moving Avg. Convergence Divergence (MACD)
   (d) Simple Moving Average (SMA)
   (e) Volume Weighted Moving Average (VWMA)
2. *Volume indicators*:
   (a) Accumulation/Distribution Index (ADI)
   (b) Ease of Movement (EMV)
   (c) Money Flow Index (MFI)
3. *Volatility indicators*:
   (a) Average True Range (ATR)
4. *Momentum indicators*:
   (a) Awesome Oscillator (AO)
   (b) Rate of Change (RoC)
   (c) Relative Strength Index (RSI)
   (d) Stochastic RSI %D
   (e) Stochastic RSI %K
   (f) William's %R

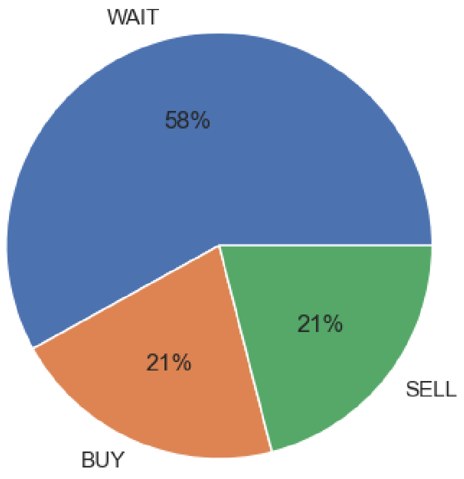


Figure 1. Chart depicting the relative distribution of different signals in the labels

Most of the above-mentioned indicators are calculated over an n-days time window. Based on popular use, we have considered $n = 12, 26$ for short-term computations and $n = 50$ and 200 for mid- and long-terms, respectively. Murphy [6] has provided a detailed mathematical view of all the above-mentioned indicators.

3. The Data

The raw dataset obtained from Kaggle [9] contains the price action data for TCS from January 14, 2002, to June 05, 2020. Using the raw data, we calculate the technical indicators listed in section 2 and create a new dataset for analysis and learning, which contains a total of 40 features. Based on the general rule of thumb, buy low and sell high, the dataset adds another column, Signal, that will serve as the label vector for all supervised learning algorithms.

3.1. EDA and feature selection

Before any pre-processing, we perform exploratory data analysis (EDA) and generate insights on the data. Overall, the data does not contain any missing values, which makes the pre-processing less tedious. As we can see in figure 1, most of the signals are of type 'wait' and the number of buy and sell signals are the same. In each feature, the standard deviation is quite high when compared to the mean of that

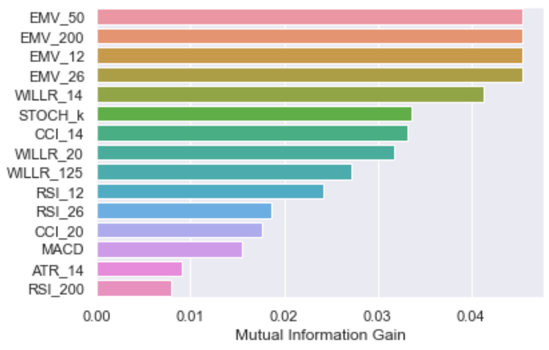
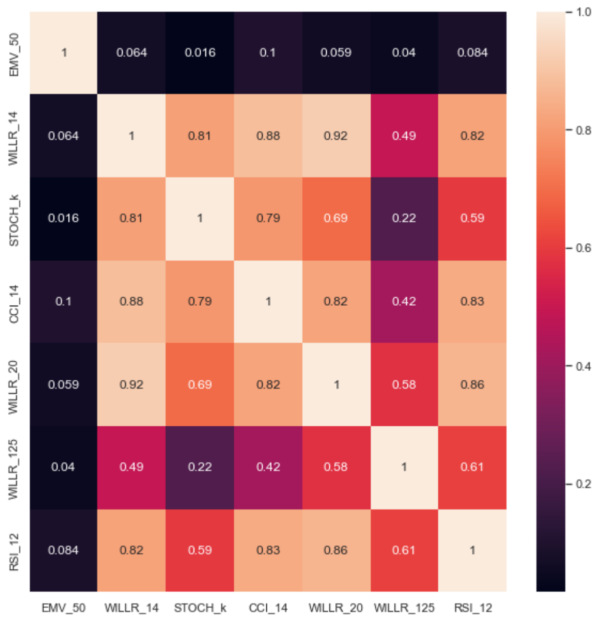


Figure 2. Chart depicting the top 15 features on the basis of mutual information gain



	EMV_50	WILLR_14	STOCH_k	CCI_14	WILLR_20	WILLR_125	RSI_12
EMV_50	1	0.064	0.016	0.1	0.059	0.04	0.084
WILLR_14	0.064	1	0.81	0.88	0.92	0.49	0.82
STOCH_k	0.016	0.81	1	0.79	0.69	0.22	0.59
CCI_14	0.1	0.88	0.79	1	0.82	0.42	0.83
WILLR_20	0.059	0.92	0.69	0.82	1	0.58	0.86
WILLR_125	0.04	0.49	0.22	0.42	0.58	1	0.61
RSI_12	0.084	0.82	0.59	0.83	0.86	0.61	1

Figure 3. Chart depicting the top 15 features on the basis of mutual information gain

feature. This indicates that the movement of indicators in time has a significant component of randomness.

Now, we consider selecting the appropriate features for better learning. Based on the mutual information gain of the features, we select the top 15 (out of 40) for further analysis. This is being depicted in figure 2. Out of these 15, we drop the features that are fully correlated, and only keep one of them intact. The result of this step is a correlation matrix shown in figure 3.

From this analysis and selection process, we finally narrow down to the following features (listed in decreasing order of information gain):

1. 50-days Ease of Movement (EMV_50)
2. 14-days William's %R (WILLR_14)
3. Stochastic RSI %K (STOCH_k)
4. 14-days Commodities Channel Index (CCI_14)
5. 20-days William's %R (WILLR_20)
6. 125-days William's %R (WILLR_125)
7. 12-days Relative Strength Index (RSI_12)

Using these features, we train the models for signal generation. The next section provides a brief about the learning models that we have utilized in this work.

4. Learning and Analysis

It is worth reiterating that our aim is to compare the profitability of different machine learning algorithms in financial markets. It is also noteworthy that the problem at hand is a classification task, i.e., we will be classifying a given set of indicator values as a potential buy, hold or sell. In the following subsections, we provide a brief about the learning algorithms whose results have been studied. Following are the algorithms under observation:

1. Logistic Regression
2. Decision Trees
3. Random Forests
4. AdaBoost
5. Naïve Bayes
6. Multilayer Perceptron
7. Support Vector Machines
8. K Nearest Neighbors

4.1. Logistic Regression

Logistic Regression is one of the simplest classification algorithms, inherently designed for binary classification problems. However, the concepts can be used in multi-class classification as well using the well known one-versus-all encoding. Let us say there are three classes in the labels, A, B, and C. In order to convert this problem into a binary classification task, let us keep A as it is and club the classes, B and C into a wrapper class, say D. Now the problem is reduced to classifying the data points into A or D, hence binary. If say the data point belongs to class D, we now break the abstraction layer and perform binary classification for B and C. In order to keep the mathematical explanation simple, we will only consider two classes, labeled 0 and 1. The decision boundary that logistic regression utilizes for this task is:

$$\sigma(\theta^T x) = 0.5 \tag{1}$$

where $\sigma(z)$ is the sigmoid function, defined as:

$$\sigma(z) = \frac{1}{1 + e^{-z}} \tag{2}$$

4.2. Decision Trees

A decision tree is one of the most explainable learning models that can be deployed for both regression and classification tasks. As the name suggests, it consists of several nodes, branches and leaves. At a given node, a trained decision tree will select the appropriate branch ahead based on the criterion defined by that node. The way that node defines its criterion is that it selects a feature (from the dataset) that provides the highest information gain at that level in the tree, and then recursively builds sub-trees in all of its branches. The said information gain can be computed either by using entropy or the Gini index. Kotsiantis [7] has provided a detailed mathematical and analytical view of decision trees.

4.3. Random Forests

An RF belongs to the class of ensemble learning algorithms, in which an ensemble of weak models is used to generate a collective output. In case of regression, the collective output might be calculated as the mean of the ensemble. On the other hand, majority voting is the popular selection criterion in case of classification tasks. In the same way, a random forest is built using a bunch of weak decision trees and then majority voting is carried out to get the final label.

4.4. AdaBoost

In machine learning, the AdaBoost (Adaptive Boosting) algorithm is a family of supervised learning methods that use error correction rather than error punishment. The idea behind AdaBoost is that each decision tree gets progressively stronger or more correct, on average.

The error correction approach is advantageous because it turns the original learning problem into an error estimation problem. In an AdaBoost model, the training data provide a set of instances that are possibly misclassified. For each instance, the AdaBoost algorithm outputs a weighted combination of decision rules trained on previous iterations. The combination is designed so that, on average, it corrects existing errors more than it makes new ones.

AdaBoost has many potential applications in machine learning because the simple base-case classifiers can be any type of machine learner (linear regression, nearest neighbour, decision tree, etc.). The AdaBoost algorithm starts by obtaining a set of decision rules (”leaf nodes”) from training data by ”fine-tuning” the parameters. These are the rule thresholds for each variable (the value over which a predictor variable should be used in making a prediction).

To make an initial prediction, an instance is given as input to all of the rules: using these values, each rule computes its distributional or ”boosted” error on that instance. This error is used to select one rule (the one with the smallest error) out of the remaining rules. The selected node is then added to the set of leaf nodes and used as input in new decisions based on the original classification data.

4.5. Naïve Bayes

This model is based on the popular Bayes theorem. Say there are n features, represented by $X = (\mathbf{x}_1, \mathbf{x}_2, ..., \mathbf{x}_n)$, and a label vector $\mathbf{y}$. According to Bayes theorem:

$$P(\mathbf{y}|X) = \frac{P(X|\mathbf{y})P(\mathbf{y})}{P(X)} \tag{3}$$

or,

$$P(\mathbf{y}|\mathbf{x}_1, \mathbf{x}_2, ..., \mathbf{x}_n) = \frac{P(\mathbf{x}_1, \mathbf{x}_2, ..., \mathbf{x}_n|\mathbf{y})P(\mathbf{y})}{P(\mathbf{x}_1, \mathbf{x}_2, ..., \mathbf{x}_n)} \tag{4}$$

The defining property of Naïve Bayes is that it makes a ’naïve’ assumption: all the features are independent of each other. This reduces the computational complexity as: $P(\mathbf{x}_1, \mathbf{x}_2, ..., \mathbf{x}_n|\mathbf{y})$ simplifies to $P(\mathbf{x}_1|\mathbf{y})P(\mathbf{x}_2|\mathbf{y})...P(\mathbf{x}_n|\mathbf{y})$ due to independence. So the effective relationship becomes:

$$P(\mathbf{y}|\mathbf{x}_1, \mathbf{x}_2, ..., \mathbf{x}_n) = \lambda P(\mathbf{x}_1|\mathbf{y})P(\mathbf{x}_2|\mathbf{y})...P(\mathbf{x}_n|\mathbf{y})P(\mathbf{y}) \tag{5}$$

Now, for a given set of inputs X, the predicted label vector $\mathbf{y}$ will be obtained as:

$$\mathbf{y} = \arg\max_{\mathbf{y}} P(\mathbf{y}) \prod_{i=1}^{n} P(\mathbf{x}_i|\mathbf{y}) \tag{6}$$

4.6. Multilayer Perceptron

Among the most primitive artificial neural networks (ANNs), an MLP is a fully connected network where each hidden node is connected to each node in its previous and next layers. Every node is a neuron that fires a weighted sum of its inputs through an activation function. The training phase of an MLP involves several epochs, each containing a sequence of forward pass and backpropagation. The backpropagation algorithm (BPA) is where the learning happens. MLPs can be used for both regression and classification tasks. In our case (classification), the output layer will consist of only 3 nodes. Each node will correspond to one output class. Let us say that the output nodes are labeled N_1, N_2, and N_3 corresponding to the buy, hold, and sell signals respectively. Now, if the MLP predicts a buy signal, then the output nodes will give out the values 1, 0, and 0, respectively. In essence, the number of output neurons for an MLP classifier will be the same as the number of output classes available.

4.7. Support Vector Machines

SVM establishes the optimum decision boundary, the line that can divide n-dimensional space into classes, so we can quickly classify new data points in the future. The hyperplane is the name given to this optimal decision boundary. In SVM, we choose vectors (data points) at the extreme ends of the data clusters, which helps in creating

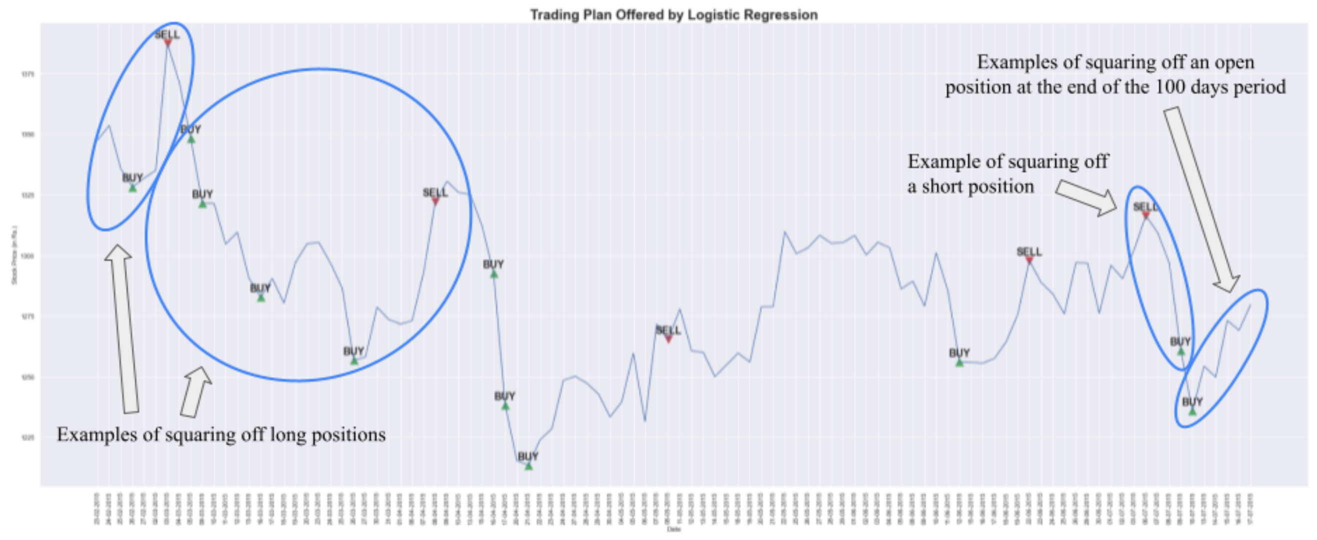


Figure 4. Visual depiction of the trading rules mentioned in section 5.2. Note that we are squaring off any open position(s) as soon as the opportunity arrives. Also, at the end of the 100 days trading window, we square off any outstanding position.

the hyperplane. There are two types of SVM: Linear and Non-Linear. When separating linearly-separable data, we use Linear SVM. It draws a straight line as the hyperplane through the aid of support vectors. In the case of non-linear data, non-linear SVM is implemented, in which another dimension is added to the dataset, making the data linearly separable.

4.8. K Nearest Neighbors

"The k-nearest neighbors algorithm (k-NN) is a non-parametric supervised learning method" *(Wikipedia)*. It can be used for both classification and regression. Input is given to this algorithm in the form of k closest training examples from the dataset. For classification, the output is a class membership, while in the case of regression, the output is the property value for the object.

With K-Nearest Neighbor (k-NN), all computation is postponed until after the function has been evaluated and the algorithm is only locally approximated. Since this technique relies on distance for classification, normalizing the training data can significantly increase accuracy if the features reflect several physical units or have distinct sizes. A weighted Nearest Neighbor algorithm can be implemented as follows:

$$\sum_{i=1}^{n} w_{ni} = 1 \tag{7}$$

5. Results and Profits

5.1. Experimental methodology

Before training any model, we first split the pre-processed dataset (the one with the technical indicators) and split it in a 70:30 ratio. Since the data is time-series, we can't do random selection of samples for the split. Simply taking the first 70% of the samples as training data and the remaining 30% for testing would be a good choice. For learning, we have used the scikit-learn library. Each model generates a set of trading signals, for which we compute their potential profits using the test dataset in a window of 100 trading days (February 23, 2015 to July 17, 2015). The chosen time window contains all three kinds of market trends: bullish, bearish and sideways. We must revisit the fact that our aim is to compare the profitability of different learning algorithms. Ergo, the accuracy of these models is not of concern to us. Now, we define our performance metric, the profit, and how we are computing the same.

5.2. Performance metric

As mentioned above, our sole performance metric is the amount of profit each model generates. The higher the profit, the better the model. But the way we are computing the profit must be stated and understood clearly. Our trading methodology follows the following rules strictly:

1. **Square-off:** The first priority of any trading action in our system is a square-off, i.e., it will always check for the possibility of squaring-off any previous position before executing the trade.

2. **No outstanding securities:** At the end of the 100 days period, any outstanding positions will be automatically resolved on the last trading day (July 17, 2015).

A visual explanation of the above-mentioned rules is provided in figure 4, which is the trading plan provided by logistic regression. A crucial assumption is the starting capital. We assume that the system starts with an ample amount of initial capital so as to be able to perform as many buy orders as needed.

5.3. Profit analysis

Now we come to the most crucial part of our study, computing and comparing profits using the trading scheme mentioned in section 5.2. Following metric values were obtained from the described setup:

Model deployed	**Profit**
Logistic Regression	22.13%
Decision Tree	27.44%
Random Forest	29.01%
AdaBoost	7.6%
Naïve Bayes	**120.39%**
Multilayer Perceptron	21.2%
Support Vector Machine	87.94%
K Nearest Neighbors	28.96%

6. Concluding Remarks

6.1. A review

We started our discussion with a brief history of algorithmic trading and how machine learning has taken over the domain by generating trade signals based on historic data. We then discussed some of the existing literature on this subject and how more and more complex algorithms are making their way through the industry. After that, we discussed some introductory concepts in technical analysis and listed the technical indicators used in our work. After a brief description, we explored the dataset and generated some insights out of it, after which we performed feature selection based on information gain and got the best features to train the models. Then, we provided a brief description of each learning algorithm used in the study. Finally, we defined our trading strategy and calculated the profits generated by different learning models.

6.2. Best model

From section 5.3, it is clear that Naïve Bayes generates the best profit (120%). However, the number of trades carried out by this model is also on the higher end when compared to other models.

6.3. Future work

There are a lot of opportunities in enhancing this study along various dimensions. More advanced models from deep learning and reinforcement learning can be tested for their profitability, whereas on the trading side, better strategies can also be explored. An inclusion of overhead charges and taxes is also important in order to compute the true profit.

References

[1] Lin, C. H. (2004). Profitability of a filter trading rule on the Taiwan stock exchange market. Master thesis, Department of Industrial Engineering and Management, National Chiao Tung University.

[2] Wu, M. C., Lin, S. Y., & Lin, C. H. (2006). An effective application of decision tree to stock trading. *Expert Systems with applications, 31*(2), 270-274.

[3] Teixeira, L. A., & De Oliveira, A. L. I. (2010). A method for automatic stock trading combining technical analysis and nearest neighbor classification. *Expert systems with applications, 37*(10), 6885-6890.

[4] Fernandez-Rodrıguez, F., Gonzalez-Martel, C., & Sosvilla- Rivero, S. (2000). On the profitability of technical trading rules based on artificial neural networks: Evidence from the Madrid stock market. *Economics letters, 69*(1), 89-94.

[5] Chang, P. C., Liu, C. H., Fan, C. Y., Lin, J. L., & Lai, C. M. (2009, September). An ensemble of neural networks for stock trading decision making. In *International conference on intelligent computing* (pp. 1-10). Springer, Berlin, Heidelberg.

[6] Murphy, J. J., *Technical Analysis of the Financial Markets: A Comprehensive Guide*

[7] Kotsiantis, S.B. Decision trees: a recent overview. *Artif Intell Rev* 39, 261–283 (2013).

[8] Investopedia: Technical Indicators `https://www.investopedia.com/terms/t/technicalindicator.asp`

[9] Kaggle: `https://www.kaggle.com/`

[10] Technical Analysis Library: `https://technical-analysis-library-in-python.readthedocs.io/en/latest/`

[11] AdaBoost: `https://www.analyticsvidhya.com/blog/2021/09/adaboost-algorithm-a-complete-guide-for-beginners/`

[12] Investopedia: Filter rule in trading `https://www.investopedia.com/terms/f/filterrule.asp`