VietFinanceLab - Analyzing Market Trends and Forecasting Future Price

My Anh Tran Nguyen
20235474
Anh.TNM235474

Huyen Nguyen Thu
20235507
Huyen.NT235507

Ha Dan Tran Le
20235483
Dan.TLH235483

Ly Nguyen Khanh
20235600
Ly.NK235600

Abstract

Stock Price Prediction is the task of forecasting future stock prices based on historical data and various market indicators. It involves using statistical models and machine learning algorithms to analyze financial data and make predictions on the future performance of a stock to help investors make informed investment decisions. This report is a part of our work in the class 157361. It introduces predicting stock prices of the Vietnamese market, divided into sectors, by some simple machine learning techniques with four algorithms: ARIMA, XGBoost, Support Vector Regression, and Random Forest Regressor.

Keywords: Statistics, Machine Learning, Stock, Prediction, ARIMA, XGBoost, Support vector regression, Random forest regressor.

1 Introduction

THE VIETNAMESE STOCK MARKET has emerged as a vital component of the country's financial landscape, offering investors significant growth potential and opportunities for capital appreciation. With its dynamic economy and increasing participation from both domestic and international investors, accurately predicting stock prices has become a crucial task for market participants, financial institutions, and policymakers.

The ability to predict stock prices can provide valuable insights for investors, enabling them to make informed decisions regarding buying, selling, or holding stocks. Furthermore, accurate predictions can help market regulators identify potential risks and implement appropriate measures to safeguard market stability.

Despite the importance of stock price prediction, the task remains highly challenging due to the complex nature of financial markets, the interplay of various factors, and the presence of inherent uncertainties. However, advancements in computational techniques, machine learning algorithms, and access to vast amounts of historical financial data have opened up new avenues for developing robust prediction models.

This project report aims to explore the field of stock price prediction in the context of the Vietnamese stock market. By employing data analysis, statistical modeling, and machine learning techniques, we seek to forecast the future performance of selected Vietnamese stocks and examine the factors influencing their price movements.

The primary objectives of this project are as follows:

- To analyze historical stock market data from the Vietnamese market, including stock prices, and other relevant financial indicators.
- To identify key market trends, patterns, and dependencies that can aid in understanding the underlying dynamics of stock price movements.
- To develop predictive models using machine learning algorithms and statistical techniques to forecast future stock prices accurately.
- To evaluate the performance of the prediction models based on established evaluation metrics and compare them against traditional forecasting approaches.

By the conclusion of this project, we expect to achieve the following outcomes:

- A comprehensive analysis of historical stock market data, identifying trends and patterns specific to the Vietnamese stock market.
- Development of accurate prediction models capable of forecasting future stock prices for selected Vietnamese stocks.

- Comparative analysis between the performance of the prediction models and traditional forecasting approaches.
- Insights and recommendations for investors, financial institutions, and policymakers to make informed decisions in the Vietnamese stock market.

Through this project, we aim to contribute to the growing body of research in stock price prediction while providing practical implications for stakeholders in the Vietnamese stock market.

2 Data Analysis and Feature Selection

2.1 Data crawling

When searching for publicly available stock data using Google Dataset Search, we found that most of the datasets primarily cover international stock exchanges such as the NYSE and NASDAQ. However, since stock prices are highly influenced by local economic conditions and government policies, our focus was on collecting data specific to the Vietnamese market, including major exchanges such as HOSE, HNX, and UPCOM.

To obtain these data, we explored several Vietnamese financial dashboards and chose the *vnstock* Python library, which provides programmatic access to data from local sources such as *VCI* (*VietCapital Securities*). This library allows us to query stock quotes using a simple interface by specifying the stock symbol, source, time range, and interval.

In our implementation, we configured the function to retrieve daily historical data using the `quote.history()` method. The returned dataset includes six primary attributes: Open, High, Low, Close, Volume, and Trading Date, to which we added a Stock Code column for identification.

To ensure consistency, we converted all price fields to Việt Nam đồng (VND) by multiplying by 1,000 and formatted the values to one decimal place. The data is sorted chronologically and exported to .csv format using UTF-8 encoding with tab separators for downstream analysis.

The final dataset spans from the first trading date of the stock up to May 10, 2025, and is stored locally for further processing and visualization.

2.2 Data analysis

For the stock codes, we selected those that would provide us insight into the stock market as a whole. The stock market is categorized into 11 sectors, according to the Global Industry Classification Standard (GICS): finance, technology, consumer (service, F&B, etc.), construction, and other industries. Research suggests that sector performance is influenced by macroeconomic variables (Nguyễn, T. N. Q. and Võ, T. H. L., 2019), and stock prices also indicate a nation's economic environment (Keswani et al., 2024). On this basis, we concluded that sector performance can reflect the state of the stock market. For the sake of time efficiency, easy analysis, and generality, we decided to obtain data from the four aforementioned industries through the following stock codes:

- Finance:
  - TCB: Vietnam Technological and Commercial Joint Stock Bank.
  - VIG: Vietnam Financial Investment Securities Corporation.
- Technology:
  - VGI: Viettel Global Investment JSC.
  - ITD: Tien Phong Technology JSC.
- Consumer:
  - LSS: Lam Son Sugar JSC.
  - PLX: Vietnam National Petroleum Group.
- Construction:
  - LCS: Licogi JSC.
  - PTC: ICapital Investment JSC.

Furthermore, we reassessed whether the chosen stock codes were representative of the Vietnamese stock market. Figure 1 illustrates the stock prices of four companies, each of a different sector, from 2007 to 2025.

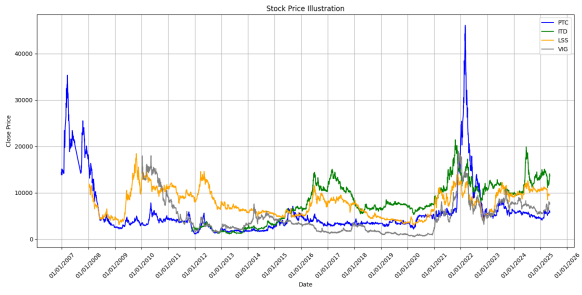

Figure 1: Stock price of PTC, ITD, LSS, and VIG from the first listed date up to now

Stock code	Number of data	Start Date	End Date
LCS	3537	2010-07-06	2025-05-09
PTC	4505	2006-12-25	2025-05-09
LSS	4317	2008-01-09	2025-05-09
PLX	2011	2017-04-21	2025-05-09
TCB	1748	2018-06-04	2025-05-09
VIG	3977	2010-01-04	2025-05-09
ITD	3500	2011-11-18	2025-05-09
VGI	1733	2018-09-25	2025-05-09

Table 1: Summary of dataset sizes and date ranges

As we can see, stock prices during 2008-2012 experienced a significant drop. This is due to the international economic crisis, which also affected Vietnam. During that period, the Vietnamese government implemented policies for the national economy. Consequently, prices increased slightly in 2010. However, they continued to fall later, as the policies were not sustainable in the long term (Nguyễn, T. T. V., 2010).

In addition, we observe a sharp increase in stock prices in 2021 and 2022. This is because the Vietnamese government introduced policies to promote economic growth after the COVID-19 pandemic (Chương, P. H., 2020).

Hence, the stock prices correspond to the trends in the Vietnamese stock market.

2.3 Feature selection

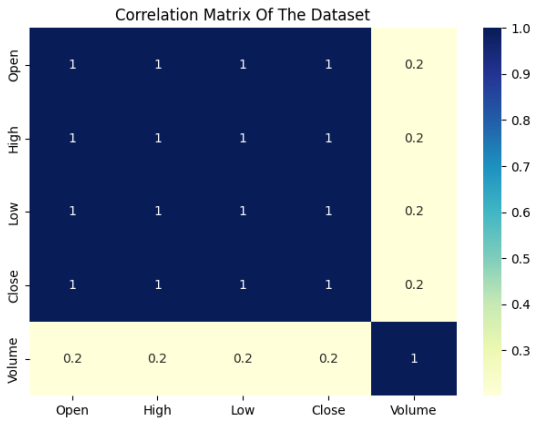

Figure 2: Correlation between features in stock price datasets

To select the appropriate feature favoring time efficiency and accuracy, we analyzed the dataset and plotted a correlation heatmap representing the correlation among various features. We utilized the *pandas*, *seaborn*, and *matplotlib* Python libraries to compute the correlation matrix, plot the correlation heatmap, and display it, respectively. The heatmap is shown in Figure 2.

The figure suggests that the relationships among the Open, High, Low, and Close features are strongly positive, while those between Volume and the other features are notably weaker. Moreover, the correlation coefficients between the Open, High, Low, and Close features are equal. All of these features represent the stock prices during a day, so we decided to use only one of them, the Close feature. However, we continued to utilize Volume in another part of our project to predict transacted volume by RSI score (relative strength index score) (Gumparthi, 2017).

2.4 RSI score and the relation to the volume

As mentioned above, we used the relative strength index to predict the volume. To start, we considered what RSI is and how it is related to the stock.

To calculate RSI, we define upward (U) and downward (D) indicators, which are:

$$U_t = \begin{cases} P_t - P_{t-1} & \text{if } P_t > P_{t-1}, \\ 0 & \text{otherwise.} \end{cases}$$

and

$$D_t = \begin{cases} -P_t + P_{t-1} & \text{if } P_t < P_{t-1}, \\ 0 & \text{otherwise.} \end{cases}$$

where t is the calculated day and P_t is the stock price for that day. Then up_t and down_t are average numbers of upward and downward moves of the closing price of the past n days:

$$\text{up}_t = \frac{s}{n+1} \times U_t + \left(1 - \frac{s}{n+1}\right) \times \text{up}_{t-1}$$

and

$$\text{down}_t = \frac{s}{n+1} \times U_t + \left(1 - \frac{s}{n+1}\right) \times \text{down}_{t-1}$$

where s is the smooth parameter. Normally, $s = 2$.

Finally, we get the RSI score by using the formula:

$$\mathrm{RSI}_t = 100 - \frac{100}{1 + \frac{\mathrm{up}_t}{\mathrm{down}_t}}$$

In calculations, we consider directly:

$$\mathrm{RSI}_t = 100 \times \frac{\mathrm{up}_t}{\mathrm{up}_t + \mathrm{down}_t}$$

The RSI tends to remain more static during uptrends than it does during downtrends. This makes sense because the RSI measures gains versus losses. In an uptrend, there are more gains, keeping the RSI at higher levels. In a downtrend, on the other hand, the RSI tends to stay at lower levels. During an uptrend, the RSI tends to stay above 30 and should frequently hit 70. During a downtrend, it is rare to see the RSI exceeds 70, and the indicator frequently hits 30 or below this threshold. These guidelines can help determine trends' strength and spot potential reversals. The reverse is true for a downtrend. This means that if the downtrend is unable to reach 30 or below, and then rallies above 70, that downtrend is said to weaken.

Hence, we can use the RSI for volume prediction to gain insight into how many stocks are exchanged over time and what the current trend of the stock code is. From that, investors can decide the volume of their transactions to maximize their benefits.

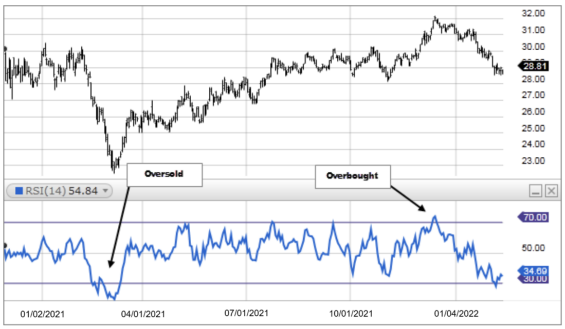

Figure 3: Example of RSI score corresponding to buy-sell trends

3 Algorithm Selection and Implementation

A. Random Forest Regressor

Random Forest Regressor is a machine learning algorithm used for regression tasks, namely, predicting continuous values such as stock prices. It is based on the ensemble learning technique called Random Forest, which combines the predictions of multiple decision trees to produce more accurate results.

Random Forest Regressor builds many decision trees. Each tree is built from a random subset of N samples drawn from the training set using bootstrap sampling (sampling with replacement). This tree-building process is repeated for a specified number of trees. For regression tasks, each tree in the forest makes an individual prediction of the target value, and the final prediction is the average of all tree predictions (Polamuri et al., 2019).

B. Extreme Gradient Boosting (XGBoost)

Ensemble Learning is one of the Machine Learning methods that combines some base models to give an optimal model. Instead of making a model and deploying this model expecting to bring the predictor accurately, ensemble methods take a lot of models into account and compute the average of those models to produce the final results.

Boosting is one of the three main methods in Ensemble Learning. Different from bagging, which trains many decision trees parallelly on different samples of the same dataset and produce predictions by averaging, the boosting method combines a set of weak learners into strong learners to minimize training error. In boosting, a random sample of data is selected, fitted with a model, and then trained sequentially, namely, each model tries to compensate for the weakness of its predecessor.

Extreme Gradient Boosting Machine (XGBM) is the latest version of boosting machines. In XGBM, trees are added one at a time. The machine learns from the errors of previous trees and improves them.

Figure 4 illustrates how gradient tree boosting works.

C. Support Vector Regression (SVR)

Support Vector Machine (SVM) is a machine learning algorithm for classification problems. After the invention of SVM, it was updated to support vector regression (SVR) by incorporating a form of loss function called the ε insensitive loss function, which penalizes data points as long as they exceed ε. SVR in a high-dimensional feature space becomes a non-linear kernel-based regression method that determines the optimal regression hyperplane with the minimum structural risk (Ghosh and Gor, 2022).

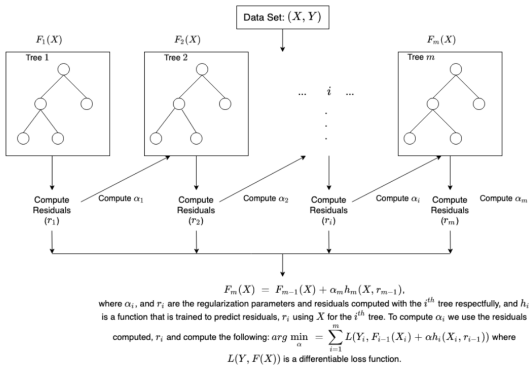

Figure 4: Illustration of XGBoost

The function of SVR method is:

$$y = f(x) = w^T \varphi(x) + b$$

subjects to:

$$\begin{cases} y_i - w^T \varphi(x_i) - b \leq \varepsilon, \\ b + w^T \varphi(x_i) - y_i \leq \varepsilon + \xi_i^* \end{cases}$$

Here, ξ_i and ξ_i^* are slack variables introduced to cope with training data possibly violating the condition $|f(x_i) - y_i| \leq \varepsilon$

$$min \frac{1}{2}||w||^2 + C \sum_{i=1}^{n} (\xi_i + \xi_i^*)$$

subject to:

$$\begin{cases} y_i - w^T \varphi(x_i) - b \leq \varepsilon, \\ b + w^T \varphi(x_i) - y_i \leq \varepsilon + \xi_i^* \\ \xi_i, \xi_i^* \geq 0, i = 1, ..., n \end{cases}$$

where C is a constant known as the penalty factor, ε is the insensitive loss parameter, and the slack variables ξ_i and ξ_i^* measure the amount of difference between the estimated value and the target value beyond ε.

D. Auto Regressive Integrated Moving Average (ARIMA)

ARIMA is a class of models that explains a given time series based on its own past values, that is, its own lags and the lagged forecast errors, so that the equation can be used to forecast future values (Prabhakaran, 2023).

Any 'non-seasonal' time series that exhibits patterns and is not a random white noise can be modeled with ARIMA models.

An ARIMA model is characterized by 3 terms: p, d, q, where,

- p is the order of the AR term
- q is the order of the MA term
- d is the number of differencing required to make the time series stationary.

The first step to building an ARIMA model is to make the time series stationary, because the term "Auto Regressive" in ARIMA means it is a linear regression model that uses its own lags as predictors. Linear regression models work best when the predictors are not correlated and are independent of each other. The most common approach is to difference them, that is, to subtract the previous value from the current value. Sometimes, depending on the complexity of the series, more than one differencing may be needed.

The value of d, therefore, is the minimum number of differencing needed to make the series stationary. If the time series is already stationary, then $d = 0$.

p is the order of the "Auto Regressive" (AR) term. It refers to the number of lags of Y to be used as predictors. And q is the order of the "Moving Average" (MA) term. It refers to the number of lagged forecast errors that should go into the ARIMA Model.

A pure Auto Regressive (AR only) model is one where Y_t depends only on its own lags. That is, Y_t is a function of the "lags of Y_t."

$$Y_t = \alpha + \beta_1 Y_{t-1} + \beta_2 Y_{t-2} + ... + \beta_p Y_{t-p} + \varepsilon_1$$

where Y_{t-1} is the lag1 of the series, beta1 is the coefficient of lag1 that the model estimates, and α is the intercept term, also estimated by the model.

A pure Moving Average (MA only) model is one where Y_t depends only on the lagged forecast errors.

$$Y_t = \alpha + \epsilon_t + \varphi_1 \epsilon_{t-1} + \varphi_2 \epsilon_{t-2} + ... + \varphi_q \varepsilon_{t-q}$$

where the error terms are the errors of the autoregressive models of the respective lags. The errors E_t and E_{t-1} are the errors from the following equations:

$$Y_t = \beta_1 Y_{t-1} + \beta_2 Y_{t-2} + ... + \beta_t Y_0 + \varepsilon_t$$

$$Y_{t-1} = \beta_1 Y_{t-2} + \beta_2 Y_{t-3} + ... + \beta_{t-1} Y_0 + \varepsilon_{t-1}$$

An ARIMA model is one where the time series is differenced at least once to make it stationary,

and the AR and the MA terms are combined. The equation then becomes:

$$Y_t = \alpha + \beta_1 Y_{t-1} + \beta_2 Y_{t-2} + ... + \beta_p Y_{t-p} + \varepsilon_t$$

$$+\varphi_1 \varepsilon_{t-1} + \varphi_2 \varepsilon_{t-2} + ... + \varphi_q \varepsilon_{t-q}$$

4 Model Tuning

A. Hyperparameter Tuning

Hyperparameter tuning is primarily finding the set of parameters' values such that they contribute to the performance of the model, maximize the efficiency, yet still ensure the complexity within a dataset. This process is used not only to improve the predictive power of the model, but also to improve the speed, making the model runs faster.

1) Choosing hyperparameters: It is probably the hardest problem in tuning hyperparameters. The reason is that it is difficult to choose the most important ones to tune, and the hyperparameters interact and affect each other within the process. Therefore, a one-time evaluation doesn't work, since when you change to another hyperparameter, the evaluation will be completely different. In this model, intuitively and experimentally, we select the hyperparameters that are likely has the most significant impact on the model for the tuning test.

Random Forest Regressor:

- **n_estimators**: This parameter determines the number of decision trees to be built by the RFR model. Increasing the number of estimators can improve the model's performance by reducing bias, but it also increases computation time.
- **min_samples_leaf**: This parameter guarantees a minimum number of samples in a leaf. A very small number will usually mean the tree will overfit, whereas a large number will prevent the tree from learning the data.
- **max_depth**: This parameter represents how deeply each tree in the boosting process can grow during training. A tree's depth refers to the number of levels it has from the root node to the leaf nodes. If max_depth is too large, the tree may grow too deep and likely overfit the training data. On the other hand, if this number is too small, the tree may not be able to learn the data's patterns, leading to underfitting.

We used GridSearchCV with 5-fold cross-validation to find the optimal combination of the hyperparameters n_estimators, min_samples_leaf, and max_depth.

Support Vector Regression:

- **C**: In SVR (Support Vector Regression), the regularization parameter, denoted as C, controls the bias-variance trade-off. It determines the penalty for data points that fall outside the margin or violate the regression tolerance. A higher value of C in SVR implies a smaller tolerance for errors, making the model try to fit the training data more closely. This can lead to overfitting, where the model becomes too specific to the training data and performs poorly on unseen data. On the other hand, a lower value of C allows for a wider margin and allows more errors, resulting in a more flexible model that can better generalize unseen data. In the provided code, the values of C are [0.01, 0.1, 1, 100], and the SVR model will be trained and tested using each of these values to find the optimal value of C for the given regression task.
- **kernel**: This parameter determines the type of kernel function used in SVR to transform the feature space. There are three commonly used kernels:
  - **linear**: Linear kernel, used when the data can be well separated by a hyperplane.
  - **rbf**: Gaussian kernel (RBF - Radial Basis Function), the most commonly used kernel, used when the data is not linearly separable in the original space.
  - **sigmoid**: Sigmoid kernel, used when the data is not linearly separable and has similar characteristics to the sigmoid function in logistic regression. The model will be trained and tested with each of these kernel types.
- γ: This parameter adjusts the influence of a training data point on other data points. A higher gamma value limits the influence to nearby points only, while a lower gamma value spreads the influence to both nearby and distant points. In the given code, the values of gamma are [0.01, 0.001, 1], and the model will be trained and tested with each of these values.

These parameters allow adjusting the SVR model to find optimal separating hyperplanes in the feature space for better performance.

XGBoost:

- **n_estimators**: This parameter represents the number of individual decision trees (weak learners) to be built in the XGBoost model. Increasing the number of estimators can improve the model's performance by reducing bias, but it also increases computation time. In the provided code, the values of n estimators are [100, 500, 1000], and the model will be trained and tested with each of these values.

- **max depth**: This parameter defines the maximum depth of each decision tree in the GBM model. A higher max depth allows the model to learn more complex relationships in the data, but it may lead to overfitting. On the other hand, a lower max depth restricts the tree's complexity, reducing overfitting, but potentially sacrificing some predictive power. In the provided code, the values of max depth are [3, 6, 10], and the model will be trained and tested with each of these values.

- **learning rate**: This parameter controls the contribution of each tree in the ensemble. A smaller learning rate makes the model more conservative by reducing the impact of each tree, leading to slower learning, but potentially better generalization. Conversely, a higher learning rate allows the model to learn faster, but may also result in overfitting. In the given code, the values of learning rate are [0.01, 0.05, 0.1], and the model will be trained and tested with each of these values.

B. Cross Validation For Time Series

When constructing a model, it is important to assess its performance. Cross-validation is a statistical technique that can assist in this process. One commonly used approach is K-fold Cross-Validation, where the dataset is divided into several subsets or folds. The model is trained on all but one fold, and then tested on the remaining fold. This process is repeated for each fold, allowing for a comprehensive evaluation of the model's performance.

However, we can't use this process in Time Series. Cross-validation becomes more challenging in the context of time series data. Randomly assigning samples to the test and training sets is not appropriate because it would involve using future values to predict past values, which is illogical. In other words, future-looking needs to be avoided while training the model.

Time Series Cross Validation: The solution is a unique way of Cross-validation for time series. Cross-validation on a rolling basis is a technique that can be applied to cross-validate the time series model. Start with a small subset of data for training purposes, make predictions for subsequent data points, and then assess the precision of the predictions. The following training dataset has the same predicted data points, and additional data points are forecasted after that. These splits must follow these rules:

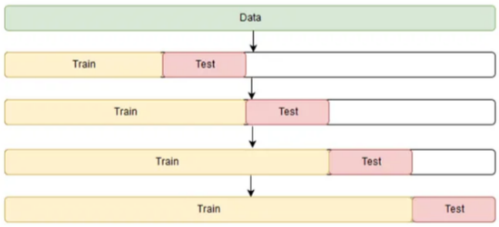

Figure 5: Cross Validation for Time Series

- Every test set contains unique observations
- Observations from the training set occur before their corresponding test set

In this project, we introduce two kinds of **Nested Cross Validation**. We will address the case of stock with data spanning numerous days:

- Predict Second Half
- Day Forward Training

*1) **Predict Second Half***: The first approach is called Predict Second Half, which serves as the fundamental nested cross-validation method. The "Predict Second Half" method involves a single train/test split, where the first half of the data is used for training, and the second half is used for testing.

In this approach, the first half of the data is allocated to the training set, and the latter half is designated as the test set, with the validation set's size

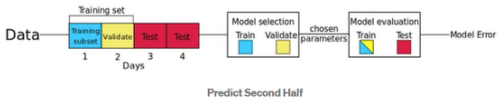

Figure 6: Predict Second Half

varying depending on the specific problem being addressed. It is crucial to maintain chronological order, ensuring that the validation set consists of data subsequent to the training subset. Figure 6 illustrates how Predict Second Half works.

The advantage of this approach is its simplicity in implementation. However, it has a limitation: The test set is chosen arbitrarily, which may introduce bias or lead to unreliable results. To address this, it is important to maintain the chronological order of the data, ensuring that the validation set, used for model selection or hyperparameter tuning, consists of data that comes after the training subset. This helps in evaluating the model's generalization performance on unseen future data.

2) ***Day Forward-Chaining***: The Predict Second Half nested cross-validation method above has a limitation in that the arbitrary selection of the hold-out test set can lead to biased estimates of prediction error when evaluating the model on an independent test set. To overcome this drawback and obtain a more reliable estimate of model prediction error, a common approach is to create multiple train/test splits and average the errors across all splits. One effective technique for achieving this is known as *Day Forward-Chaining*.

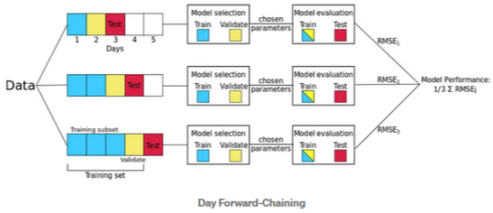

Figure 7: Predict Second Half

Day Forward-Chaining is based on the principles of forward-chaining and rolling origin-recalibration evaluation. This method involves systematically considering each day as the test set while assigning all previous data to the training set. For instance, if we have a dataset spanning five days, Day Forward-Chaining would generate three different training and test splits, ensuring that there is at least one day of training and validation data available. The figure 7 illustrates this process.

By employing this approach, we generate multiple train/test splits, allowing us to assess the model's performance across different periods of time. Each split represents a distinct evaluation scenario, where the model is trained on historical data and tested on subsequent days. This diversity of splits helps to capture the model's ability to generalize and make accurate predictions across different time periods.

To compute a robust estimate of the model error, the prediction error on each split is averaged. This averaging process smooths out any fluctuations or variations that may occur due to specific train/test splits and provides a more stable evaluation of the model's performance.

3) ***Comparison and Conclusion***: The Predict Second Half approach relies on a single train/test split, which may not capture the full variability of the data or provide a robust estimate of model error. The Predict Second Half approach may not effectively evaluate the model's ability to generalize to unseen future data, as the test set is chosen arbitrarily.

Whereas, Day-Forward Training generates multiple train/test splits by considering each day as a separate test set. By averaging the model's performance across these splits, a more robust estimate of the model's error can be obtained, accounting for variations and fluctuations across different time periods. These align with the objective of forecasting future values in time series analysis. By training the model on past data and evaluating it on subsequent days, the model is tested on unseen future data points, allowing for a better assessment of its generalization capabilities.

In summary, Day-Forward Training offers advantages over the Predict Second Half approach in terms of test set selection, accounting for temporal dynamics, evaluation robustness, and generalization to future unseen data. By systematically considering each day as a test set, Day-Forward Training provides a more representative evaluation of the model's performance and better captures the temporal dependencies present in time series data.

5 Implementation

All the source code and implementation are here: Repository.

The dataset is attached in the folder "Dataset", which includes some stocks in different fields (finance, technology, consumer, construction).

We analyze some indicators like RSI, buy or sell signal in the folder "Data Analysis". You can access this to see the implementation, some properties of the dataset, and our buy-or-sell prediction using RSI.

Regarding machine learning models, we use four algorithms, which are ARIMA, Support Vector Regression (SVR), Extreme Gradient Boosting (XGBoost), and Random Forest Regressor (RFR) to train, deploy, and evaluate models, then give predictions about the price of stocks. Before using algorithms to train the models, we preprocessed the data, which involved cleaning and visualizing for easier implementation.

In the ARIMA model, we apply some test statistics to check whether the data is stationary, and then choose the parameters to train and evaluate the model.

In the folders SVR and XGBoost, we try model selection and choose optimal hyperparameters by using Nested Cross Validation Time Series (Predict-Second-Half and Day-Forward-Chaining). After that, we use those hyperparameters to retrain, evaluate the models, and predict the price of stocks.

The Random Forest Regressor (RFR) model was implemented using the scikit-learn library in Python. The data was then split into features (X) and target (y), and further split into training and testing sets. The SimpleImputer class from the scikit-learn library was used to fill in any missing values in the training data. The best hyperparameters for the RFR model were found using GridSearchCV. The model was then fit to the training data using the best hyperparameters found and was used to predict the stock prices on the testing set. Finally, the model is evaluated on the test set by comparing predicted vs actual values.

6 Summaries and Implementation

A. Results and Summaries

After running the models several times, we obtain the accuracy of the chosen dataset, which is represented in the table below. The numbers in each cell are train mean squared error, train R^2 score, test mean squared error, and test R^2 score, respectively.

In conclusion, the XGBoost models performed well on the training set but showed inconsistent underperformance on the test set compared to the other models. These indicators suggest signs of overfitting. In contrast, the ARIMA, Support Vector Regression, and Random Forest Regressor models demonstrated acceptable performance. From this, we can gain clear insight into stock market trends, provided there are no sudden policy changes (i.e., national bankruptcy, stock takeover, etc.).

B. Proposal of Improvements

Further Model Tuning: To enhance the performance of the XGBoost model, we recommend tuning its hyperparameters. This adjustment will help the model avoid overfitting the test set and improve performance on unseen data.

Enhancing Data Integration: To improve the overall performance of the system, we recommend integrating additional relevant data sources. By incorporating financial news, social media sentiment analysis, and economic indicators, we can capture a broader range of factors that influence stock prices. This enriched dataset will allow more comprehensive analysis and prediction, resulting in enhanced decision-making capabilities for users.

User-Friendly Interface and Visualization: To ensure an intuitive and user-friendly experience, we propose developing a web-based interface that allows users to interact with the system effortlessly. The interface should provide clear visualizations of predicted stock prices, historical data, and relevant performance metrics. Additionally, incorporating features such as customizable alerts and notifications will encourage users to make timely and informed investment decisions.

7 Acknowledgments

This work is supported and supervised by Professor Sang V. DINH, Professor Thanh H. NGUYEN under the course of Machine Learning, and Professor Tuan Q. DAM under the course of Applied Statistics and Experimental Design.

8 Disclaimers

Our project is done just for knowledge learning and researching. It is not investigated in-depth enough to make it applicable. We are not responsible for any risks when you invest your money on what our system recommended.

Algorithm	LCS	PTC	LSS	PLX	TCB	VIG	ITD	VGI
ARIMA	0.0158	0.0111	0.0170	0.0224	0.0211	0.0179	0.0110	0.0263
	0.9864	0.9937	0.9915	0.9790	0.9932	0.9912	0.9976	0.9782
	0.0080	0.0327	0.0340	0.0270	0.0322	0.0380	0.0474	0.1577
	0.9687	0.9676	0.9613	0.9212	0.9747	0.9370	0.9065	0.9584
SVR	0.0052	0.0039	0.0054	0.0152	0.0044	0.0033	0.0028	0.0215
	0.9948	0.9961	0.9946	0.9848	0.9956	0.9967	0.9972	0.9785
	0.0032	0.0064	0.0169	0.0209	0.0489	0.0356	0.0275	0.0384
	0.9968	0.9936	0.9831	0.9791	0.9511	0.9644	0.9725	0.9616
SVR & DFC	0.0061	0.0037	0.0061	0.0173	0.0032	0.0030	0.0021	0.0019
	0.9938	0.9960	0.9941	0.9771	0.9938	0.9968	0.9974	0.9766
	0.0012	0.0050	0.0060	0.0114	0.0064	0.0044	0.0053	0.0078
	0.9794	0.9779	0.9882	0.9633	0.9668	0.9774	0.9804	0.9542
SVR & PSH	0.0064	0.0036	0.0057	0.0259	0.0036	0.0028	0.0018	0.0207
	0.9936	0.9964	0.9943	0.9741	0.9964	0.9972	0.9982	0.9793
	0.0105	0.0054	0.0070	0.0133	0.0069	0.0051	0.0113	0.0061
	0.9895	0.9946	0.9930	0.9867	0.9931	0.9949	0.9887	0.9939
XGB	0.0002	0.0002	0.0004	0.0003	0.0002	0.0002	0.0002	0.0004
	0.9997	0.9997	0.9995	0.9996	0.9998	0.9998	0.9998	0.9996
	0.1385	0.0447	0.0208	0.0371	0.1868	0.0443	0.0677	1.0000
	0.8614	0.9552	0.9791	0.9628	0.8132	0.9557	0.9323	0.0000
XGB & DFC	0.0026	0.0007	0.0053	0.0117	0.0009	0.0021	0.0014	0.0010
	0.9973	0.9990	0.9948	0.9845	0.9988	0.9977	0.9982	0.9870
	0.0210	0.0294	0.0065	0.0497	0.1812	0.0114	0.0338	0.7232
	0.7147	0.9660	0.9870	0.8941	0.3127	0.9310	0.9065	-0.8025
XGB & PSH	0.00003	0.00009	0.00003	0.00003	0.00003	0.00002	0.00008	0.00002
	0.99997	0.99999	0.99997	1.00000	1.00000	0.99998	1.00000	0.99998
	0.21508	0.03499	0.00819	0.12182	0.02055	0.01409	0.09098	0.21134
	0.78492	0.96501	0.99181	0.87818	0.97945	0.98591	0.90902	0.78866
RFR	0.0004	0.0005	0.0005	0.0009	0.0003	0.0003	0.0002	0.0003
	0.9996	0.9995	0.9995	0.9991	0.9997	0.9997	0.9998	0.9997
	0.0014	0.0024	0.0018	0.0041	0.0010	0.0014	0.0008	0.0016
	0.9986	0.9976	0.9982	0.9959	0.9990	0.9986	0.9992	0.9984

Table 2: Performance comparison of models (train/test MSE and R^2).

References

Chương, P. H. 2020. Tác động của đại dịch Covid-19 đến nền kinh tế Việt Nam. *Tạp chí Kinh tế và Phát triển*, 274:12.

Nguyễn, T. N. Q. and Võ, T. H. L. 2019. Tác động của một số yếu tố kinh tế vĩ mô đến chỉ số giá chứng khoán tại Việt Nam. *TẠP CHÍ KHOA HỌC ĐẠI HỌC MỞ THÀNH PHỐ HỒ CHÍ MINH-KINH TẾ VÀ QUẢN TRỊ KINH DOANH*, 14(3):47–63.

Nguyễn, T. T. V. 2010. Cuộc khủng hoảng kinh tế thế giới năm 2008–2009 và tác động đối với kinh tế-xã hội Việt Nam. Luận văn ThS. Quan hệ quốc tế: 60 31 40, Trường Đại học Khoa học Xã hội và Nhân văn.

M. Ghosh and R. Gor. 2022. Stock price prediction using support vector regression and k-nearest neighbors: A comparison. *International Journal of Engineering Science Technologies*, 6(4):1–9.

S. Gumparthi. 2017. Relative strength index for developing effective trading strategies in constructing optimal portfolio. *International Journal of Applied Engineering Research*, 12(19):8926–8936.

Sarika Keswani, Veerma Puri, and Rimjhim Jha. 2024. Relationship among macroeconomic factors and stock prices: cointegration approach from the indian stock market. *Cogent Economics & Finance*, 12(1):2355017.

Dr Polamuri, Kudipudi Srinivas, and A. Mohan. 2019. Stock market prices prediction using random forest and extra tree regression. *International Journal of Recent Technology and Engineering*, 8:1224 – 1228.

S. Prabhakaran. 2023. Arima model – complete

guide to time series forecasting in python. https://www.machinelearningplus.com/time-series/arima-model-time-series-forecasting-python/. Machine Learning Plus.