Stock Price Prediction

Nguyen Minh Cuong [1, *], Dang Kieu Trinh [1, *], Nguyen Viet Long[1, *], Nguyen Phan Anh Vu[1, *], Nguyen Tung Phong[1, *], Ngo Trung Dung[1, *]

Abstract

This project focuses on predicting the prices of Vietnamese stocks using a combination of advanced statistical and machine learning models. Accurate stock price prediction is crucial for investors and financial analysts to make informed decisions. To achieve this, we employ a variety of models, each leveraging different techniques and strengths. Our methodology includes the following models: Autoregressive Integrated Moving Average (ARIMA), Multiple Linear Regression (MLR), Support Vector Regression (SVR), Random Forest, XGBoost, and AdaBoost. Each model is designed to capture different aspects of the stock price data, from linear relationships to complex, non-linear patterns. The project includes comprehensive data preprocessing, feature engineering, and hyperparameter tuning for each model to ensure optimal performance. The models are evaluated based on their predictive accuracy and robustness, with results compared to identify the best-performing approach for Vietnamese stock price prediction. Our findings demonstrate the potential of combining multiple models to enhance prediction accuracy, offering valuable insights for investors and contributing to the field of financial forecasting in emerging markets like Vietnam.

Keywords

time series forecasting; machine learning,

[1] *The School of Information and Communication Technology - Hanoi University of Science and Technology*

*__Corresponding author__:
cuong.nm210140@sis.hust.edu.vn,
trinh.dk214933@sis.hust.edu.vn,
long.nv225506@sis.hust.edu.vn,
vu.npa225466@sis.hust.edu.vn,
phong.nt225517@sis.hust.edu.vn,
dung.nt225487@sis.hust.edu.vn

Contents

1. Introduction

Predicting stock prices is a complex and challenging task that has significant implications for investors, financial analysts, and economic policymakers. In emerging markets such as Vietnam, accurate stock price prediction is particularly valuable due to the rapid economic growth and evolving market dynamics. This project aims to enhance the prediction accuracy of Vietnamese stock prices by leveraging a diverse set of statistical and machine learning models.

The Vietnamese stock market has experienced substantial growth and increased investor interest over the past decade. However, its volatility and unique market characteristics pose challenges for traditional prediction methods. To address these challenges, our project employs a multifaceted approach, utilizing both time series forecasting and machine learning techniques.

The primary objective of this project is to develop a robust and accurate model for predicting the prices of Vietnamese stocks. By leveraging a combination of advanced statistical and machine learning techniques, we aim to identify the most effective approach for capturing the unique dynamics of the Vietnamese stock market. The specific goals of the project include:

- Model Development: To develop and implement six dif-

ferent models—ARIMA, Multiple Linear Regression (MLR), Support Vector Regression (SVR), Random Forest, XGBoost, and AdaBoost—for stock price prediction.

- Data Preprocessing and Feature Engineering: To preprocess historical stock price data and engineer relevant features that enhance the predictive power of the models.
- Model Evaluation and Comparison: To evaluate the performance of each model based on predictive accuracy and robustness, and to compare their results to identify the best-performing approach.
- Optimization: To optimize each model through hyperparameter tuning and other techniques to achieve the highest possible accuracy.
- Insights and Recommendations: To provide actionable insights and recommendations for investors and financial analysts based on the predictive models.

The expected outcomes of this project are as follows:

- Enhanced Prediction Accuracy: By employing a diverse set of models, we anticipate achieving higher prediction accuracy compared to using a single model. Each model's strengths will complement the others, resulting in more reliable predictions.
- Comprehensive Model Evaluation: A thorough evaluation and comparison of the six models will highlight their respective strengths and weaknesses, providing a clear understanding of which models are best suited for different aspects of stock price prediction in the Vietnamese market.
- Actionable Insights for Investors: The project aims to generate valuable insights for investors, helping them make more informed decisions based on accurate stock price predictions.
- Contribution to Financial Forecasting: The findings will contribute to the broader field of financial forecasting, particularly in emerging markets like Vietnam. The project will demonstrate the effectiveness of integrating multiple predictive models and offer a framework for similar studies in other markets.
- Scalable and Adaptable Framework: The methodologies and models developed in this project will be scalable and adaptable, allowing them to be applied to other stocks and markets with appropriate modifications.

By achieving these objectives and outcomes, the project will not only enhance the understanding of stock price prediction in the Vietnamese market but also provide a valuable tool for investors and contribute to the advancement of financial forecasting methodologies.

.

2. Data preparation

2.1 Data collection

The sample data are the daily data up to 24 June 2023. The data of the study were collected from Investing at `https://investing.com` .The target variable in the experiment is the close price of stock in USD.

Our dataset consists of 7 attributes: Trading date, Open, High, Low, Close, Adj Close and Volume.

2.2 Data analysis

Choosing the right stock code is crucial, as a random selection provides no meaningful insights. As we investigated, there are 11 different stock market sectors, according to the most commonly used classification system, known as the Global Industry Classification Standard (GICS) ,which are Energy, Materials, Industrials, Consumer Discretionary, Consumer Staples, Healthcare, Financials, Information Technology, Communication Services, Utilities, and Real Estate . Here, for the sake of time and easier analysis, we decide to getting data from 3 of 11 above sectors and choose the stock index that has the largest market capitalization(total value of the company, representing the size of the business) in each sector in HOSE stock exchange:

- Materials: – HPG: Hoa Phat Group GSC
- Utilities : – GAS: Petro VietNam gas JS
- •Industrials: – CTG: Viet Nam Joint Stock Commercial Bank for Industry and Trade

This section will examine the relationship between macroeconomic factors and political events, demonstrating their influence on stock market performance. It will also explore broader economic trends impacting the market. Several major events and trends can be identified during the period covered by the graph(Figure1):

- 2008-2009 Financial Crisis: During this period, there is a noticeable decline in stock prices, reflecting the global economic turmoil and its impact on the Vietnamese market.
- 2012-2015: The graph shows a period of relative stability and gradual growth, which can be correlated with economic reforms and improvements in the global economy.
- 2016-2020: Fluctuations in this period could be linked to both local political changes and international trade tensions.
- 2020-2022: The sharp movements in this period may be attributed to the COVID-19 pandemic and its economic repercussions, along with recovery efforts and stimulus measures.

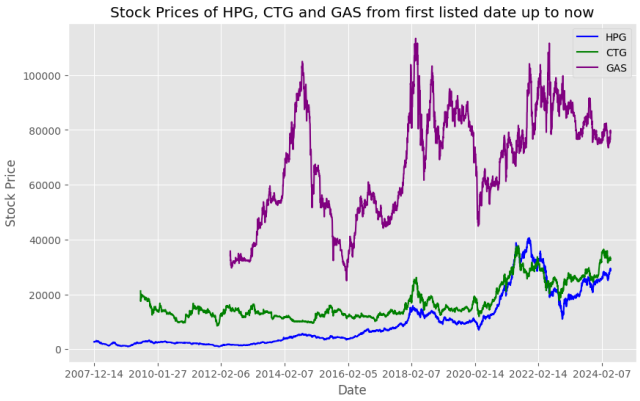

Figure 1. Stock Price of HPG, CTG and GAS from first listed date up to now

The analysis clearly indicates that the stock prices of HPG, CTG, and GAS have been influenced by various macroeconomic and political events over the years. Significant declines and growth periods correspond to major global and local events, demonstrating the interconnected nature of the stock market with broader economic and political trends. Understanding these influences is crucial for investors seeking to make informed decisions in the stock market.

2.3 Data preprocessing and feature selection

Data preprocessing The collected dataset contains gaps in data for certain days. This is likely due to non-trading days, such as holidays or weekends, or potential errors in the data source website. After removing these days with missing data, the dataset has been cleaned and is now suitable for model construction.

203	2010-04-28	14014.932617	14014.932617	13916.236328	13965.583984	13965.583984	160889
204	2010-04-29	13817.539063	13965.583984	13817.539063	13965.583984	13965.583984	420675
205	2010-04-30	null	null	null	null	null	null
206	2010-05-03	null	null	null	null	null	null
207	2010-05-04	14113.629883	14459.068359	13965.583984	14409.719727	14409.719727	1175288
208	2010-05-05	14311.022461	14311.022461	14113.629883	14162.977539	14162.977539	306487

Figure 2. Raw data with null values

Feature selection This correlation matrix analysis demonstrates that the stock price metrics (Open, High, Low, Close, Adj Close) are highly interrelated, moving in tandem with each other. This is expected as these metrics are derived from the same underlying stock price movements. However, the trading volume, while positively correlated with the stock prices, does not show as strong a relationship, indicating that other factors may influence trading volume independently of price movements.

In conclusion, understanding these correlations helps in identifying the redundancy among the price features and the relatively independent role of trading volume in stock price analysis. This insight is valuable for investors and analysts who aim to develop predictive models or investment strategies based on these features.

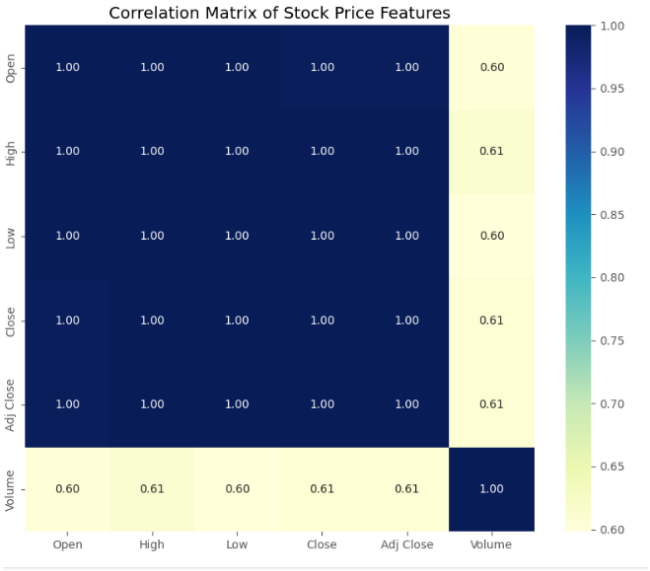

Figure 3. Correlation Matrix of Stock Price Features

3. Method

3.1 Statistical methods

3.1.1 ARIMA

Introduction The ARIMA (AutoRegressive Integrated Moving Average) model is a widely used statistical analysis model for time series forecasting. This section details the process of implementing an ARIMA model to forecast stock prices, including the theoretical background with mathematical formulas, algorithm, and model evaluation. The model we use can forecast a stationary time series or a non-stationary time series which can be made stationary.

Data preparation Only the Trading Date and Close values were selected. The dataset was split into training and testing sets. The training set consisted of historical data used to train the ARIMA model, while the testing set was used to evaluate the model's performance. A 70-30 split was employed. The 'Close' for train data values were normalize within a range 0 and 1.

Stationarity A (weak) stationary time series satisfies properties: mean and variance of the series is the same for all time t and the covariance (also correlation) between x_t and x_{t-h} (x_t denote the value of a time series at time t)is the same for all t at each lag h = 1,2,3,.. This means the series has no trend and no seasonal pattern. We need to check for stationarity.

Simple Moving Average(SMA) A K-day SMA is calculated by averaging the values over a sliding window of K days. For

a time series $_t$, the K-day SMA at time t is given by:

$$SMA_t = \frac{1}{K}\sum_{i=0}^{K-1} x_{t-i}$$

While it can help visualize trends and detect small parts of stationarity, it is not a good tool for detecting stationarity.

Autocorrelation Function (ACF) [1] The Autocorrelation Function (ACF) measures the correlation between a time series and its lagged versions. To compute the ACF for a time series: Firstly, calculate the mean (μ) of the time series:

$$\mu = \frac{1}{N}\sum_{t=1}^{N} x_t$$

Secondly, compute the autocovariance of each lag k (γ_k):

$$\gamma_k = \frac{1}{N}\sum_{t=k+1}^{N} (x_t - \mu)(x_{t-k} - \mu)$$

where γ_k is the autocovariance at lag k.

Thirdly, compute the variance (γ_0) of the time series :

$$\gamma_0 = \frac{1}{N}\sum_{t=1}^{N} (x_t - \mu)^2$$

Next, Compute the autocorrelation at lag k (denoted as ρ_k) which is the autocovariance at lag k normalized by the variance:

$$\rho_k = \frac{\gamma_k}{\gamma_0}$$

Finally, we repeat the above calculations for all lags $k = 1, 2, \ldots,$ max_lag. We then plot the ACF plot to see if the ACF. For a stationary time series, the ACF typically decays quickly to zero, indicating that correlations between observations decrease as the time lag increases.

Partial Autocorrelation Function (PACF) [1] The PACF at lag k, denoted as , measures the direct relationship between x_t and x_{t-k} while removing the effects of the intermediate lags ($x_{t-1}, x_{t-2}, \ldots, x_{t-(k-1)}$). For a stationary time series, the PACF also tends to drop quickly to zero after a few lags. To compute the PACF at lag k (ϕ_{kk}) : We fit an autoregressive model of order k (AR(k)) to the time series. The AR(k) model is given by:

$$x_t = \phi_1 x_{t-1} + \phi_2 x_{t-2} + \cdots + \phi_k x_{t-k} + \varepsilon_t$$

where $\phi_1, \phi_2, \ldots, \phi_k$ are the coefficients of the AR model, and ε_t is the white noise error term which follow the normal distribution of mean 0 .

The Yule-Walker equations are used to estimate the parameters of the above AR model. The Yule-Walker equations for an AR(k) process are:

$$\gamma_h = \sum_{j=1}^{k} \phi_j \gamma_{h-j}, \quad \text{for } h = 1, 2, \ldots, k$$

where γ_h is the autocovariance at lag h. To solve for the coefficients $\phi_1, \phi_2, \ldots, \phi_k$, we can represent the Yule-Walker equations in matrix form:

$$\begin{pmatrix} \gamma_0 & \gamma_1 & \cdots & \gamma_{k-1} \\ \gamma_1 & \gamma_0 & \cdots & \gamma_{k-2} \\ \vdots & \vdots & \ddots & \vdots \\ \gamma_{k-1} & \gamma_{k-2} & \cdots & \gamma_0 \end{pmatrix} \begin{pmatrix} \phi_1 \\ \phi_2 \\ \vdots \\ \phi_k \end{pmatrix} = \begin{pmatrix} \gamma_1 \\ \gamma_2 \\ \vdots \\ \gamma_k \end{pmatrix}$$

We need solve this system of linear equations to find the coefficients $\phi_1, \phi_2, \ldots, \phi_k$. We may use matrix algebra and the Levinson-Durbin recursion algorithm which maps the autocovariance γ_h to a Toeplitz diagonal-constant matrix. The PACF at lag k, denoted as ϕ_{kk}, is simply the k-th coefficient in the AR(k) model. For example, ϕ_{kk} is the last coefficient obtained from solving the Yule-Walker equations.

Augmented Dickey-Fuller Test The Augmented Dickey-Fuller (ADF) test is a powerful tool for testing the stationarity of a time series. By testing the presence of a unit root, it helps determine whether a time series is stationary or non-stationary. The procedure involves fitting a regression model, computing the test statistic, and comparing it to critical values to make a decision. A result of the p-value smaller than 0.05 is a strong evidence against the null hypothesis, the data has no unit root and is stationary.

Differencing Differencing is a common technique used to achieve stationarity, which is the Integrated (I) part of the ARIMA model. The Integrated part, denoted by d, represents the number of times differencing is needed to make the series stationary. To calculate the first Difference ($d = 1$), we take the difference between consecutive observations:

$$\Delta x_t = x_t - x_{t-1}$$

If the first difference does not achieve stationarity, further differences can be taken:

$$\Delta^d x_t = \Delta(\Delta^{d-1} x_t)$$

where Δ^d denotes d times differencing.

ARIMA model The ARIMA model is represented as ARIMA(p, d, q), where:

- p : The order of the AutoRegressive (AR) component, which represents the number of lag observations included in the model. The AR component is based on the idea that the past values in the time series can be used to predict future values.
- d : is the differencing order.
- q : is the size of the moving average window. which represents the number of lagged forecast errors in the prediction equation . The MA component is based on the idea that the past forecast errors can help predict future values .

The mathematical representation of the ARIMA model is as follow:

$$x_t = c + \phi_1 x_{t-1} + \ldots + \phi_p x_{t-p} + \theta_1 \varepsilon_{t-1} + \ldots + \theta_q \varepsilon_{t-q} + \varepsilon_t \quad (1)$$

Where: x_t: The value of the time series at time t. c: The constant term. ϕ_k: Autoregressive (AR) coefficients. x_{t-p}: Lagged values of degree p of the time series. θ_g: Moving Average (MA) coefficients. ε_t: Forecast errors or residuals at time t. p: The order of the Autoregressive (AR) process. q: The order of the Moving Average (MA) process.

Akaike Information Criterion (AIC) [2] The Akaike Information Criterion (AIC) is a widely used metric for comparing the relative quality of statistical models, particularly in the context of time series analysis and forecasting. Suppose that we have a statistical model of some data. Let k be the number of estimated parameters in the model. Let $\hat{L}$ be the maximized value of the likelihood function for the model. Then the AIC value of the model is the following. The AIC for a model is defined as:

$$\text{AIC} = -2 \cdot \ln(\hat{L}) + 2 \cdot \text{k}$$

A model that has a small AIC value is generally considered a better model. There are different ways to compute the likelihood function. The statsmodel python library use maximum likelihood estimator for the computation.

ARIMA Algorithm work flow

Step 1: Load the time series data.
Step 2: Split the data into training and testing sets.
Step 3: Scale the data using MinMaxScaler.
Step 4: Check for stationarity
Step 4.1: Check visually by plotting the simple moving average.
Step 4.2: Check visually by plotting the ACF plot and PACF plot
Step 4.3: Use the Augmented Dickey-Fuller test.
If data is non-stationary, difference the data until it becomes stationary.
Step 5: Perform grid search to find the best (p, d, q) parameters:
For each combination of (p, d, q):
Step 5.1: Fit the ARIMA model.
Step 5.2: Calculate the AIC.
Step 5.3: Select the model with the lowest AIC.
Step 6: Fit the ARIMA model with the best (p, d, q) parameters on the training set.
Step 7: Forecast the test set forecast method:
Step 7.1: Initialize history with the training data.
Step 7.2: For each time step in the test set:
Step 7.2.1: Fit the ARIMA model on the history.
Step 7.2.2: Forecast the next value.
Step 7.2.3: Append the forecasted value to the predictions and update the actual observation value to the history.
Step 8: Rescale the forecasted values and the confidence intervals to the original scale.
Step 9: Plot the actual and forecasted values.
Step 10: Evaluate the model using RMSE, MAPE and R2 metrics.

3.2 Machine Learning methods

3.2.1 Multiple Linear Regression

Ridge Regression

Ridge Regression Ridge Regression is a technique used for analyzing multiple regression data that suffer from multicollinearity. When multicollinearity occurs, least squares estimates are unbiased, but their variances are large, which may lead to an overfitted model. Ridge Regression adds a degree of bias to the regression estimates, reducing their standard errors.

Regularization Ridge Regression incorporates a regularization term to the ordinary least squares (OLS) cost function. This regularization term is a penalty proportional to the square of the magnitude of the coefficients. The goal is to shrink the coefficients, thus imposing a constraint on their size.

Cost Function The cost function in Ridge Regression is modified from the ordinary least squares (OLS) regression by adding a penalty term:

$$W* = argmin \sum_{i=1}^{M} (y_i - A_i W)^2 + \lambda \sum_{j=1}^{n} {}_{(}W_j)^2$$

Here, λ is the regularization parameter, which controls the strength of the penalty term. When $\lambda = 0$, Ridge Regression is equivalent to OLS regression. As λ increases, the impact of the penalty term becomes more significant, shrinking the regression coefficients.

Regularization Parameter (λ) The regularization parameter λ controls the degree of regularization applied to the model. A larger λ value increases the penalty for large coefficients, leading to more shrinkage. Choosing an appropriate λ is crucial for balancing bias and variance.

Bias-Variance Tradeoff Ridge Regression introduces bias into the model, which can help to reduce variance and thus avoid overfitting. The tradeoff is controlled by the regularization parameter λ, and finding the optimal value is essential for model performance.

Algorithm The Ridge Regression algorithm can be summarized in the following steps:

Input: - Training set $\{(x_1, y_1), (x_2, y_2), \ldots, (x_n, y_n)\}$ - Regularization parameter λ

Initialize: - Design matrix A with dimensions $n \times m$ (including a column of ones for the intercept) - Response vector y with dimensions $n \times 1$

Steps:

1. **Compute the Ridge Regression Coefficients:** The coefficients W* are computed using the formula:

$$W* = (A^T A + \lambda I_{n+1})^{-1} A^T y$$

Here, I is the identity matrix of dimensions $m \times m$.

2. **Make Predictions:** Use the computed coefficients to make predictions for new x:

$$y_x = W_0 * + W_1 * x_1 + ... + W_n * x_n$$

Advantages of Ridge Regression :

1.Handling multicollinearity: Ridge Regression helps reduce community multicollinearity in data by shrinking the coefficients of independent variables, helping to reduce overfitting and improve the model's ability to synthesize.

2.Overfitting Reduction: By applying a penalty on the coefficients, Ridge Regression helps reduce overfitting, especially when the quantity is exceptional or when the data has a lot of collinearity.

3.Explainability: Ridge Regression retains specific symbols in the models, does not completely remove them like other types of methods such as Lasso Regression. This makes it easier to deal with the expected results.

4.Easy to develop: Ridge Regression is a simple and easy to develop method. It can be applied in many recovery operations and does not require many data conditions.

Disadvantages of Ridge Regression :

1.Does not remove features: Ridge Regression does not remove all unimportant features from the model, but only shrinks them to near zero. This can result in the model still containing conservation Protect unnecessary special equipment, increasing model complexity.

2.Difficulty understanding results: Regression coefficients obtained from Ridge Regression can be difficult to understand and interpret. This reduces the transparency of the model and reduces the confidence in the decision work.

3. Depends on the parameter λ: The effectiveness of Ridge Regression depends greatly on the choice of the parameter λ. Choosing the wrong λ value can lead to a null, overfitting, or underfitting model.

Improving Model Accuracy 1. Create upper and lower bounds for the 'Close' column of the dataset and remove outliers(the data outside the bound). This method reduces model complexity. However, it can affect the distribution and characteristics of the data set.

2. GridSearchCV was used to test the Ridge Regression model with different lambda values. The results were evaluated using the cross-validation method to find the most optimal lambda.

3.2.2 Support Vector Regression

Support Vector Regression Support Vector Regression (SVR) is a type of supervised learning algorithm used for regression tasks, meaning it predicts continuous output variables. It's based on Support Vector Machines (SVM), which are primarily used for classification tasks. SVR works by finding a hyperplane in an n-dimensional space (where n is the number of features) that best fits the data. Unlike traditional linear regression, SVR aims to minimize the error while ensuring that predictions fall within a certain margin of tolerance. This margin is controlled by two parameters: epsilon (ε) and C. The SVR algorithm involves:

- Identifying support vectors: Data points that lie within or on the margin boundaries.
- Constructing a regression hyperplane: The hyperplane is determined by the support vectors and aims to minimize the error between predicted and actual target values while maximizing the margin.

Parameter Tuning SVR includes several parameters that significantly impact model performance. The key parameters to tune are:

- **Kernel function:** Selecting an appropriate kernel function (e.g., linear, polynomial, radial basis function) that best captures the underlying patterns in the data. In this scenario, the rbf kernel is used due to its high flexibility and its ability to capture non-linear relationships without explicitly transforming the data.
- **Regularization parameter (C):** Balancing the trade-off between minimizing errors on the training data and controlling the complexity of the model. Higher values of C allow for more flexible decision boundaries, while lower values emphasize margin maximization.
- **Epsilon (ε):** Specifies the margin of tolerance where no penalty is given to errors. This margin determines the width of the tube within which predictions are not penalized.
- **Gamma (γ):** The gamma parameter determines the influence of individual training samples on the decision boundary. It essentially controls the shape of the decision boundary and the flexibility of the SVR model. A small gamma value means that the decision boundary is smooth, while a large gamma value results in a more complex and wiggly decision boundary.

SVR Algorithm The SVR algorithm aims to find a function $f(x)$ that has at most ε deviation from the actual targets for all training data points, while also ensuring that the function is as flat as possible.

Input:

- A set of n examples forming the Training Set.

$$TS_n = \{(x_1, y_1), (x_2, y_2), \ldots, (x_n, y_n)\}$$

- A kernel function $K(x_i, x_j)$.
- Regularization parameter C.

- Epsilon parameter ε.

Initialize:

- Compute the kernel matrix K using the chosen kernel function.

Prediction: For a new data point x:

$$f(x) = \sum_{i=1}^{n} (\alpha_i - \alpha_i^*) K(x_i, x) + b$$

Where b is determined from the support vectors.

Dual Problem in SVR The SVR optimization problem can be formulated in its dual form, which is often more computationally efficient, especially when using kernel methods. The dual formulation involves Lagrange multipliers α_i and α_i^* and is given by the following quadratic programming problem:

$$\begin{aligned} \text{maximize} \quad & -\frac{1}{2} \sum_{i=1}^{n} \sum_{j=1}^{n} (\alpha_i - \alpha_i^*)(\alpha_j - \alpha_j^*) K(x_i, x_j) \\ & + \varepsilon \sum_{i=1}^{n} (\alpha_i + \alpha_i^*) - \sum_{i=1}^{n} y_i (\alpha_i - \alpha_i^*) \\ \text{subject to} \quad & \sum_{i=1}^{n} (\alpha_i - \alpha_i^*) = 0 \\ & 0 \leq \alpha_i, \alpha_i^* \leq C \end{aligned}$$

In this formulation: - α_i and α_i^* are the Lagrange multipliers. - $K(x_i, x_j)$ is the kernel function which computes the similarity between the data points x_i and x_j. - ε is the width of the margin around the regression function where no penalty is given for errors.

The dual problem allows the use of kernel functions to handle non-linear relationships by mapping input features into higher-dimensional spaces without explicitly performing the transformation.

Improving Model Accuracy To enhance the accuracy of the SVR model, the following strategies can be employed:

1. **Hyperparameter Tuning:** Use techniques like Grid Search or Random Search to find the optimal values for C, ε, and kernel parameters.

2. **Feature Scaling:** Ensure that all input features are scaled (e.g., using StandardScaler) so that they contribute equally to the decision function.

3. **Cross-Validation:** Use k-fold cross-validation to ensure that the model generalizes well to unseen data.

4. **Kernel Selection:** Experiment with different kernels (linear, polynomial, RBF) to identify the one that best captures the underlying patterns in the data.

Algorithm Implementation In our project, we use the following algorithm:

Input:

Initially set of n examples are considered as Training Set.

$$TS_n = \{(x_1, y_1), (x_2, y_2), \ldots, (x_n, y_n)\}$$

Kernel function rbf: $\mathrm{K}(\mathrm{x}_i, x_j) = \exp\left(-\frac{\|x_i - x_j\|^2}{2\sigma^2}\right)$
Regularization parameter $C = 1000$
Epsilon parameter $\varepsilon = 0.01$
Gamma parameter $\gamma = 0.001$
Initialize: Compute the kernel matrix K
Prediction: For a new data point x:

$$f(x) = \sum_{i=1}^{n} (\alpha_i - \alpha_i^*) K(x_i, x) + b$$

For Better Accuracy To improve the accuracy of our SVR model, we tried these strategies:

1. **Hyperparameter Tuning:** Performed a Grid Search to find the optimal values for C, ε, and the kernel parameters. This involved systematically testing a range of values and selecting the combination that yielded the best cross-validation performance.

2. **Feature Scaling:** Applied StandardScaler to ensure that all features had zero mean and unit variance. This helped in improving the convergence of the SVR algorithm and the accuracy of the model.

3. **Cross-Validation:** Used 5-fold cross-validation to ensure that the model's performance was robust and not overfitting to the training data. This technique provided a better estimate of the model's accuracy on unseen data.

4. **Kernel Selection:** Experimented with different kernel functions. While the RBF kernel generally provided good results, we also tested polynomial and linear kernels to find the best fit for our data.

3.2.3 RandomForest

RandomForest Random Forest (Random Decision Forest) is and ensemble machine learning algorithm that operates by constructing a multitude of Decision Trees at the training time. Each tree is constructed using a random subset of the data set to measure a random subsets of features in each partition. This randomness introduces variability among individual trees, reducing the risk of overfitting and improving overall prediction performance.

Decision Trees Decision Trees is a machine learning algorithm which recursively partitions the input space based on the values of input features, aiming to minimize impurity and maximize information gain at each step. Decision Trees, however, can suffer from overfitting, especially with deep trees or noisy datasets. By aggregating the predictions of multiple trees, ensemble models reduce the risk of overfitting and enhance the robustness of the overall model.

Ensemble Methods Ensemble learning combines multiple classifiers (Decision Trees) to improve accuracy. The methods used for predicting stock price is Bagging (Bootstrap Aggregation).

- **Bagging:** involves randomly selecting data samples (with replacement) from the training set. Random Forest Regressor is used in this case to aggregating predictions through averaging the number of votes. Bagging reduces variance and is suitable with unstable like Decision Trees.

$$\hat{y} = \frac{1}{N}\sum_{i=1}^{N}\hat{y}_i$$

Where:

- $\hat{y}_i$ is the prediction from the i-th model.
- N is the total number of base models.
- $\hat{y}$ is the final aggregated prediction obtained by averaging the individual predictions.

For better accuracy To improve the accuracy, we can try tuning the parameters by perform Grid Search to find the optimal values for each parameters. We also introduce 5-fold cross-validation in the Grid Search to avoid overfitting to the training data and provide a better overall model's accuracy in prediction.

1. Tune max depth parameter: deep trees can memorize noise in the training data, leading to overfitting. By restricting the depth of individual trees, the training data can be prevented from being fitted too closely and helps the model generalize better to unseen data.
2. Limit the number of max features: by considering only a subset of features for each tree, we introduce randomness and reduce correlation, as well as reduce training and prediction times.
3. Have a minimum number of samples required to be at a leaf node: when constructing a tree, a split point is considered only if it leaves a certain number of training samples in each left and right branches. Its main purpose is to avoid overfitting by having smaller leaf sizes.
4. Setting samples leaf parameter: the depth of the decision tree is controlled by not consider splits when a node has fewer than certain number samples. Fewer splits leads to faster training and prediction times while also helps avoid overfitting by excessive splitting.
5. Tune the number of trees in forest: while having a large number of Decision Trees tends to enhance the Random Forest's predictive power and robustness, it is more reasonable to have a right number to balance the computing time with efficiency.

3.2.4 AdaBoost

Adaboost Adaboost (Adaptive Boosting) is a machine learning algorithm primarily used for classification tasks. While it is commonly associated with classification, it can also be adapted for regression problems using a technique called AdaBoost Regression. AdaBoost regression is based on the same underlying principles as AdaBoost for classification but with some modifications to fit the regression setting. [3] The key algorithms used in gradient boosting for regression include:

Gradient Boosted Decision Trees (GBDT) also known as Gradient Boosting Trees or Gradient Boosted Regression Trees (GBRT), is a popular algorithm for regression tasks. It combines multiple decision trees in a boosting fashion to iteratively improve predictions. In each iteration, a decision tree is trained to minimize the residual errors from the previous iterations, effectively fitting the model to the remaining errors.

Learning Rate The learning rate parameter controls the contribution of each weak learner (decision tree) to the final prediction. A lower learning rate makes the algorithm more conservative, requiring more iterations to converge but potentially achieving better performance.

Weak Learner Selection The weak learner used in gradient boosting for regression is typically a decision tree. The decision trees used are usually shallow (with a small number of levels or nodes) to prevent overfitting. Commonly, decision trees with a maximum depth of 1 (stumps) or a small maximum depth are used.

Residuals and Gradient Descent In each iteration, the model learns the residuals (the differences between the predicted and actual values) from the previous iteration. It fits the weak learner to these residuals using a form of gradient descent to minimize the loss function (e.g., mean squared error). The weak learner is added to the ensemble, and the process is repeated iteratively.

Ensemble Combination The final prediction of the gradient boosting model is the sum of the predictions from all the weak learners, weighted by a factor related to the learning rate.

Algorithm In our project we use this algorithm:

Input:

Initially set of n examples are considered as Training Set.

$$IP_n = \{(x_1, y_1), (x_2, y_2), \ldots, (x_n, y_n)\}$$

DecisionTreeRegressor denoted as Weak Learner
Integer ITR specifying total number of iterations.

Initialize:

$$W_1^i = (\frac{1}{n})$$

$$W_1 = (w_1^1, w_1^2, ..., w_1^n) = (\frac{1}{n}, \frac{1}{n}, ..., \frac{1}{n})$$

The ensemble $P = \emptyset$
For $itr = 1, 2, \ldots, ITR$:

Step 1. Get an example R_{itr} from IP_n using distribution W_{itr}

Step 2. P_{itr} is built using the training set R_{itr}

Step 3. Calculate

$$E_{itr} = \frac{\sum_{i:P_{itr}(x_i)\neq y_i} W^i_{itr}}{}$$

and $\alpha_{itr} = 0.5 \ln\left(\frac{1-E_{itr}}{E_{itr}}\right)$

Step 4. Update the weight:

$$W^i_{itr+1} = \text{normalize}(W_{itr} \cdot \exp(-\alpha_{itr} \cdot l_{itr}(x_i)))$$

Output: The ensemble

$$P = \{p_1, p_2, \ldots, p_{ITR}\}$$

and $A = \{\alpha_1, \alpha_2, \ldots, \alpha_{ITR}\}$

For better accuracy To improve the accuracy of our model we tried these stategies

1. Increase the number of weak learners: AdaBoost combines multiple weak learners to create a strong learner. Adding more weak learners (e.g., decision stumps) to the ensemble can potentially improve the accuracy. However, there is a limit to the number of weak learners beyond which the model may overfit the training data.

2. Use more complex weak learners: Instead of using simple decision stumps as weak learners, you can use more complex models such as decision trees with higher maximum depth. This can allow the weak learners to capture more intricate patterns in the data, potentially improving accuracy. However, be cautious of overfitting as complex weak learners may increase the risk of overfitting the training data.

3. Tune the learning rate: The learning rate controls the contribution of each weak learner to the final prediction. Experiment with different learning rates to find the optimal balance between convergence speed and accuracy. A lower learning rate makes the algorithm more conservative and can potentially lead to better accuracy by taking smaller steps towards the optimal solution.

3.2.5 XGBoost

XGBoost stands for "Extreme Gradient Boosting" and is an optimized implementation of the gradient boosting machine learning algorithm. At its core, XGBoost is an ensemble of decision trees that are trained in a sequential manner. Each new tree is trained to correct the errors made by the previous trees in the model. The final prediction is the sum of the predictions from all the individual trees. [4]

Additive Training XGBoost trains the decision trees additively. Mathematically, we can represent the prediction $\hat{y}$ at step t as:

$$\hat{y}^{(t)} = \hat{y}^{(t-1)} + f^{(t)}(\mathbf{x}) \tag{2}$$

Where:

- $\hat{y}^{(t-1)}$ is the prediction at the previous step
- $f^{(t)}(\mathbf{x})$ is the new decision tree added at step t to correct the errors from the previous prediction

The objective is to find the $f^{(t)}(\mathbf{x})$ that minimizes the overall loss function L, which measures how well the model fits the training data:

$$\text{obj}^{(t)} = \sum_{i=1}^{n} L(y_i, \hat{y}_i^{(t-1)} + f^{(t)}(\mathbf{x}_i)) + \Omega(f^{(t)}) \tag{3}$$

Here:

- $\Omega(f^{(t)})$ is a regularization term that penalizes the complexity of the new tree to avoid overfitting
- L is a differentiable convex loss function that measures the difference between the prediction $\hat{y}_i^{(t-1)} + f^{(t)}(\mathbf{x}_i)$ and the target y_i

Gradient Boosting To find the optimal $f^{(t)}(\mathbf{x})$, XGBoost uses gradient boosting. It approximates the loss function using a second-order Taylor expansion:

$$\text{obj}^{(t)} \approx \sum_{i=1}^{n} [L(y_i, \hat{y}_i^{(t-1)}) + g_i f^{(t)}(\mathbf{x}_i) + \frac{1}{2} h_i f^{(t)}(\mathbf{x}_i)^2] + \Omega(f^{(t)}) \tag{4}$$

Where g_i and h_i are the first and second order gradients of the loss function with respect to the prediction at step $(t-1)$.

By rearranging terms, we can rewrite the objective as:

$$\text{obj}^{(t)} = \sum_{i=1}^{n} [g_i f^{(t)}(\mathbf{x}_i) + \frac{1}{2} h_i f^{(t)}(\mathbf{x}_i)^2] + \Omega(f^{(t)}) + \text{constant} \tag{5}$$

The optimal $f^{(t)}(\mathbf{x})$ is then found by minimizing this simplified objective.

Tree Structure Score To build the tree structure, XGBoost greedily selects the split points that result in the maximum gain. The gain is defined as:

$$\text{gain} = \frac{1}{2}\left[\frac{G_L^2}{H_L+\lambda} + \frac{G_R^2}{H_R+\lambda} - \frac{(G_L+G_R)^2}{H_L+H_R+\lambda}\right] - \gamma \tag{6}$$

Where:

- G_L and H_L are the sum of the first and second order gradients for the instances in the left leaf node
- G_R and H_R are the sum of the first and second order gradients for the instances in the right leaf node
- λ is the L2 regularization term on leaf weights
- γ is the minimum loss reduction required to make a split

This formula essentially measures how much the loss would be reduced by introducing a split, while penalizing leaf complexity to prevent overfitting.

Shrinkage and Column Subsampling After each new tree is added, XGBoost scales the contribution of the tree by a factor η (learning rate) to slow down the learning. This shrinkage technique helps prevent overfitting.

To further prevent overfitting, XGBoost also employs column subsampling, where each tree is trained on a random subset of features. This reduces correlation between the trees and makes the model more robust.

4. Result

The accuracy of the selected datasets was assessed through repeated trials, with the results summarized in the tables below.:

Agl	GAS		CTG		HPG	
	R2	RMSE	R2	RMSE	R2	RMSE
Arima	0.99	1500.76	0.98	416.62	0.99	160.02
MLR	0.89	1732.37	0.97	152.51	0.93	235.33
SVR	0.998	687.93	0.999	184.34	0.99	112.23
RF	0.99	497.03	0.99	136.63.	0.99	92.12
XGBoost	0.99	1281.85	0.99	123.48	0.99	143.21
AdaBoost	0.992	1755.63	0.99	454.39	0.99	150.33

Figure 4. Train set results

Agl	GAS		CTG		HPG	
	R2	RMSE	R2	RMSE	R2	RMSE
Arima	0.93	1800.81	0.97	588.73	0.99	571.11
MLR	0.89	2141.26	0.97	208.99	0.93	302.47
SVR	0.998	705.32	0.999	178.72	0.99	110
RF	0.98	818.86	0.96	239.34	0.99	181.02
XGBoost	0.96	2263.28	0.91	263.56	0.94	287.93
AdaBoost	0.88	2133.28	0.96	701.29	0.97	590.21

Figure 5. Test set results

Summary In conclusion, the model demonstrates acceptable performance in predicting stock prices. This provides valuable insight into market trends, particularly when considering stable policy environments. However, it's important to note that sudden policy shifts, like national bankruptcies or stock takeovers, can significantly impact the model's accuracy.

5. Future Work

In future research we will focus on enhancing the model's robustness and accuracy. This includes developing methods to incorporate unexpected policy changes and market events, exploring additional data sources and technical indicators to improve feature engineering,better parameter tunning and creating a user-friendly platform for practical implementation. By addressing these areas, the model can be made more reliable for investment decision-making and contribute to a better understanding of stock market trends.

References

[1] A. P. Changquan Huang. Applied time series analysis and forcasting with python.

[2] H. K. Choi. Stock price correlation coefficient prediction with arima-lstm hybrid model.

[3] C. R. R. K. V. M. Dr Shirin Bhanu Koduri, Loshma Gunisetti and D.Ganesh. Prediction of crop production using adaboost regression method.

[4] W. Li, Y. Yin, X. Quan, and H. Zhang. Gene expression value prediction based on xgboost algorithm. *Frontiers in Genetics*, 10:1077, 2019.