Time series forecasting using gradient boosting algorithms

Iolanta Barysheva
Belarusian State University
Minsk, Republic of Belarus
e-mail: baryshevaiolanta@gmail.com

Vasilevsky Konstantin
Belarusian State University
Minsk, Republic of Belarus
e-mail: VasilevskyK@bsu.by

***Abstract*— This study investigates the efficiency of gradient boosting algorithms, particularly XGBoost, in time series forecasting. We optimize the parameters using RandomizedSearchCV and apply the model to daily stock prices of the Ethereum cryptocurrency. Additionally, we compare the prediction performance of XGBoost with two other models, LightGBM and CatBoost. Our findings reveal that the LightGBM model outperforms both CatBoost and XGBoost in terms of accuracy for time series prediction.**

Keywords—gradient boosting, financial time series forecasting, XGBoost, LightGBM, CatBoost.

I. INTRODUCTION

Time series forecasting is a critical aspect of various domains, including finance, sales, and weather prediction. Accurate predictions enable businesses and organizations to make informed decisions, optimize resource allocation, and improve overall efficiency.

One highly effective technique that has gained popularity in recent years is gradient boosting. In this article, we will explore how gradient boosting algorithms, such as XGBoost, LightGBM and CatBoost, can be utilized for time series forecasting.

II. XGBOOST ALGORITHM

Gradient boosting is an ensemble learning method that combines multiple weak predictive models, typically decision trees, to create a strong predictive model. It works by iteratively fitting new models to the residuals of the previous models, reducing the overall prediction error. This iterative process continues until a predefined stopping criterion is met.

XGBoost [1] is composed of classification and regression tree, and uses gradient tree boosting to implement a multi-tree ensemble learning algorithm. The sum of the predicted values for the sample for each tree is the predicted values for the XGBoost model, which is defined as follows:

$$\hat{y}_i = \phi(x_i) = \sum_{k=1}^{K} f_k(x_i) \quad (1)$$

Among them, K represents the total number of trees, $\hat{y}_i$ refers to the prediction result of the i sample, and $f_k(x_i)$ refers to the prediction of the k sample in the x_i number.

The objective function for XGBoost is:

$$Obj = \sum_{i=1}^{n} l(y_i, \hat{y}_i^{(t)}) + \sum_{k=1}^{K} \Omega(f_k) \quad (2)$$

$$\Omega(f) = \gamma T + \frac{1}{2}\lambda\|\omega\|^2 \quad (3)$$

The objective function consists of two parts, the loss function and the regularization term, $\sum_{i=1}^{n} l(y_i, \hat{y}_i^{(t)})$ is the loss function, used to measure the difference between the real result and the predicted result, and $\sum_{k=1}^{K} \Omega(f_k)$ is the regularization term, which can effectively prevent the tree structure from being too complicated and avoid the model from over fitting. In the regularization term, the first term γT is used to control the complexity of the tree, and the second term $\frac{1}{2}\lambda\|\omega\|^2$ is used to control the weight fraction of the leaf nodes.

III. DATA INTERPRETATION AND PREPROCESSING

The financial time series data collected in this paper are all from Binance cryptocurrency historical market database. The data set is Ethereum cryptocurrency daily stock price data from May 2018 to May 2023.

The stock price data are the opening price, closing price, highest price, lowest price and trading volume.

In the realm of time series analysis, it is common for most problems to exhibit either external or internal features that can aid in model performance. To address this, it is essential to incorporate appropriate feature engineering techniques.

To begin, we can introduce fundamental features such as lag values derived from the available numeric features, which are widely employed in time series problems. However, it is important to note that when predicting the stock price for a given day, we cannot utilize the feature values from that same day, as they will be unavailable during actual inference. Consequently, we must rely on statistical measures such as the mean and standard deviation of lagged values.

In this study, we will employ three distinct sets of lagged values. The first set will consist of lagged values from the previous day, providing insight into immediate trends. The second set will encompass lagged values spanning a period of seven days, serving as a proxy for weekly metrics. Lastly, the third set will encompass lagged values spanning a period of 30 days, acting as a proxy for monthly metrics. By incorporating these lagged values, we can capture relevant temporal patterns and enhance the predictive capabilities of our model.

In the realm of boosting models, the inclusion of datetime features, such as day of week, hour, day, and month, can greatly enhance their performance. By incorporating these temporal components into the model, valuable information regarding the time aspect of the data is provided. This inclusion improves the model's understanding of temporal patterns. Consequently, the inclusion of datetime features is a highly beneficial technique for enhancing the performance of boosting models.

We need to pre-process the data and check the outliers and missing values. For the missing values, the method we use is to take the average of two adjacent numbers, then normalize the data, and divide the training set and test set.

The predicted stock in the model is closing prices.

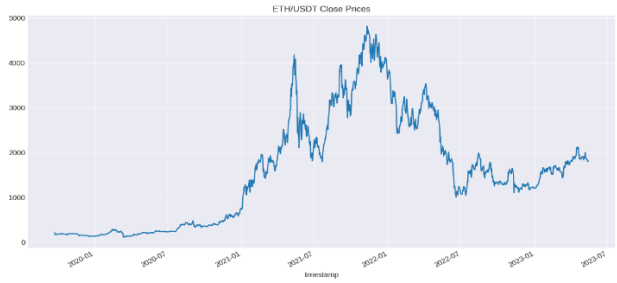

Fig. 1. Test statistic, critical values and p-value of ADF test

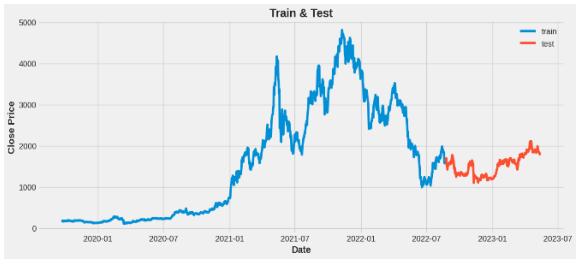

Fig. 2. Test statistic, critical values and p-value of ADF test

IV. Educational Data Analysis

The Augmented Dickey-Fuller (ADF) test is a widely used statistical test in econometrics to determine the stationarity of time series data. Stationarity is a fundamental assumption in time series models, as it guarantees the constancy of statistical properties over time.

A stationary time series maintains constant statistical properties, including mean, variance, and autocovariance, over time. On the other hand, a non-stationary time series exhibits changing trends, seasonality, or other patterns.

The ADF test [2] is an extension of the Dickey-Fuller test, initially developed by economists David Dickey and Wayne Fuller in 1979. The augmented version incorporates higher-order autoregressive processes, enabling more accurate testing of stationarity.

The ADF test is based on the null hypothesis that a unit root exists in the time series, indicating non-stationarity. The alternative hypothesis suggests that the time series is stationary. By rejecting the null hypothesis, we conclude that the time series is indeed stationary.

When conducting an ADF test, we obtain a test statistic and compare it to critical values at various significance levels. The test statistic is negative, and its magnitude determines the strength of evidence against the null hypothesis.

A more negative test statistic (i.e., farther from zero) provides stronger evidence to reject the null hypothesis of non-stationarity. Conversely, a less negative test statistic (i.e., closer to zero) fails to reject the null hypothesis.

Critical values are predetermined thresholds that assist in determining the significance level at which we can reject the null hypothesis. These values depend on the sample size, desired level of significance, and the type of ADF test (e.g., with or without a trend).

Another approach to interpreting ADF test results is by examining the p-value associated with the test statistic. The p-value represents the probability of obtaining a test statistic as extreme as the observed one, assuming the null hypothesis is true. If the p-value is lower than the chosen significance level (e.g., 0.05), we reject the null hypothesis.

TABLE I. The Augmented Dickey-Fuller test statistics analysis: Test statistic, critical values and p-value of ADF test

ADF test statistics	
Test statistic	-1.53896
p-value	0.51416
1% Critical value	-3.43535
5% Critical value	-2.86375
10% Critical value	-2.56795

The p-value is clearly greater than 0.05 significance level, we cannot reject the null hypothesis and conclude that the time series we are working with has a unit root.

Gradient boosting algorithms have emerged as powerful tools for time series forecasting. By effectively capturing temporal dependencies, handling non-stationarity, and incorporating seasonality and trends, these algorithms can provide accurate predictions in various domains. However, it is important to carefully engineer features, tune hyperparameters, and apply regularization techniques to optimize the model's performance. As time series forecasting continues to gain importance, leveraging the capabilities of gradient boosting algorithms will undoubtedly play a significant role in driving accurate and reliable predictions.

V. Hyperparameter Tuning

Hyperparameters play a crucial role in the performance of machine learning models, including the XGBoost regressor. These parameters, which are set before the learning process begins, control the behavior of the model. Finding the optimal combination of hyperparameters is a challenging task that can significantly impact the model's performance. We introduce RandomizedSearchCV [3], a technique that automates the search for the best hyperparameter combination in XGBoost regressor models.

Hyperparameters are parameters that are not learned from the data but are set manually. They determine the behavior of the model and can greatly influence its performance. Tuning these hyperparameters is essential to achieve optimal results. RandomizedSearchCV is a method that simplifies the process of searching for the best hyperparameter combination by randomly sampling from a predefined search space.

To begin the hyperparameter tuning process, it is necessary to define the hyperparameters and their corresponding possible values. For instance, the learning rate, maximum depth, number of estimators, and other relevant hyperparameters can be specified.

RandomizedSearchCV requires a parameter grid, which is a dictionary where each key represents a hyperparameter name, and the corresponding value is a list of possible values for that hyperparameter. This parameter grid represents the search space from which the hyperparameters will be sampled.

Next, an instance of the RandomizedSearchCV class needs to be created. This instance should include the XGBoost regressor model, the parameter grid, and other optional parameters such as the number of iterations and cross-validation settings.

Once the RandomizedSearchCV object is set up, the fit method can be called, passing the training data. RandomizedSearchCV will then perform a specified number of iterations, randomly sampling hyperparameter combinations from the search space. For each combination, the XGBoost regressor model is trained and evaluated using cross-validation.

After the search is complete, RandomizedSearchCV provides the best hyperparameter combination found during the search. This selection is based on a specified scoring metric, such as mean squared error or R-squared. The best hyperparameter combination can be accessed using the best_params_ attribute of the RandomizedSearchCV object.

Finally, the XGBoost regressor model can be retrained using the best hyperparameters obtained from RandomizedSearchCV on the entire training dataset. This step ensures that the final model incorporates the optimized hyperparameters.

By utilizing RandomizedSearchCV, a wide range of hyperparameter combinations can be efficiently explored. This technique enables the identification of the hyperparameter values that yield the best performance for the XGBoost regressor model. RandomizedSearchCV simplifies the task of finding the optimal hyperparameter combination.

TABLE II. PARAMETER OPTIMIZATION RESULT: TUNED HYPERPARAMETERS

Best params:	objective': 'reg:squarederror', 'colsample_bylevel': 0.6, 'colsample_bytree': 1, 'gamma': 0.6, 'learning_rate': 0.2,'max_depth': 7, 'min_child_weight': 7, 'n_estimators': 1577, 'subsample': 0.7
Best validation score	-1.48428

To facilitate a comprehensive comparison of the three gradient boosting models, including CatBoost and LightGBM models, it is imperative that each model undergoes the same set of actions as previously described. Specifically, this entails the implementation of hyperparameter optimization using RandomizedSearchCV. By ensuring consistency in the experimental setup, we can effectively evaluate and contrast the performance of these models.

VI. RESULTS

The test set prediction results obtained by all models are demonstrated below. The predicted stock in the model is closing prices.

VII. CONCLUSION

To evaluate the accuracy of the models in forecasting the target variable, RMSE and MAE are calculated for each

Fig. 3. XGBoost prediction

Fig. 4. CatBoost prediction

Fig. 5. LightGBM prediction: Error metrics

TABLE III. RMSE AND MAE SUMMARY OF DIFFERENT GRADIENT BOOSTING MODELS

	RMSE	MAE
XGBoost	163.7086	128.3656
LightGBM	133.7627	103.9441
CatBoost	206.9712	157.1234

model's predictions. RMSE measures the average difference between the predicted and actual values, giving more weight to larger errors. MAE measures the average absolute difference between the predicted and actual values, treating all errors equally. The RMSE and MAE values obtained from each model are compared. Lower values indicate better performance, as they represent smaller errors between the predicted and actual values. Both metrics should be considered to gain a comprehensive understanding of the models' performance.

Our study investigates the factors that contribute to LightGBM's potential for achieving lower error metrics compared to XGBoost and Catoost. The analysis focuses on four key aspects: handling of categorical features, training speed, handling of imbalanced data, and regularization techniques. The findings highlight the advantages of LightGBM in these areas and emphasize the importance of selecting the most suitable model for specific time series forecasting tasks.

LightGBM incorporates a specialized technique called Light Gradient-Based One-Hot Encoding to efficiently handle categorical features. This approach converts categorical features into numerical values, reducing memory usage and expediting the training process. In contrast, XGBoost and CatBoost employ different methods for handling categorical features, which may not be as efficient for time series forecasting tasks.

LightGBM is renowned for its fast training speed, attributed to its leaf-wise tree growth strategy. By prioritizing the leaves that are likely to yield the greatest loss reduction, LightGBM adopts a depth-first tree growth approach. This strategy proves advantageous for time series forecasting, particularly when dealing with large datasets. Conversely, XGBoost and CatBoost adopt a level-wise tree growth strategy, which can be comparatively slower.

Time series forecasting often involves imbalanced data, where certain periods exhibit more frequent or significant events. LightGBM addresses this issue through the is_unbalance parameter, which automatically adjusts the weights of positive and negative samples during training. This feature proves beneficial when working with imbalanced time series datasets. While XGBoost and CatBoost also offer techniques to handle imbalanced data, LightGBM's built-in functionality simplifies the process.

LightGBM provides a range of regularization techniques, including L1 and L2 regularization, to mitigate overfitting and enhance generalization. These techniques prove particularly valuable in time series forecasting, where noise or outliers can lead to overfitting. While XGBoost and CatBoost also offer regularization options, LightGBM's implementation may be more effective in certain scenarios.

It is important to note that the performance of these models can vary depending on the specific dataset and problem at hand. Therefore, it is recommended to conduct experiments with different models and hyperparameters to identify the optimal solution for a given time series forecasting problem.

REFERENCES

[1] Xu J., He J., Gu J., Wu H., Wang L., Zhu Y., Wang Z., He X. and Zhou Z, "Financial Time series prediction based on XGBoost and generative adversarial networks," International Journal of Circuits, Systems and Signal Processing, 2022, pp. 637–645.

[2] R. Harris, "Testing for unit roots using the augmented Dickey-Fuller test," Economics Letters, 1992, pp. 381–386.

[3] C. V. Priscilla and D. P. Prabha, "Influence of Optimizing XGBoost to handle Class Imbalance in Credit Card Fraud Detection," 2020 Third International Conference on Smart Systems and Inventive Technology, 2020.

[4] Y. Wang and G. Ye, "Forecasting method of stock market volatility in time series data based on mixed model of ARIMA and XGBoost," China Communications, 2020, pp. 205–221.

[5] T. Cinto, A. L. S. Gradvohl, G.P. Coelho and A.E.A. Da Silva, "Solar flare forecasting using time series and extreme gradient boosting ensembles," Solar Physics, 2020.