CS6140

1. Introduction

In the dynamic and complex realm of financial markets, accurate predictions of stock prices are crucial for a range of stakeholders, from individual investors to large financial institutions. This paper presents an in-depth investigation into the efficacy of four advanced predictive models: LightGBM Regression, XGBoost, Linear Regression, and ARIMA, in forecasting stock prices. These models represent a blend of machine learning techniques and traditional statistical approaches, each with unique strengths in handling large datasets, capturing complex nonlinear relationships, and addressing time-series forecasting challenges.

The motivation for this study stems from the ongoing quest for more reliable and precise forecasting tools in the volatile domain of stock trading. Enhanced predictive accuracy can significantly contribute to risk management, investment strategy formulation, and overall market stability. This research is particularly timely and relevant in the context of the "Optiver - Trading at the Close" Kaggle competition, which emphasizes the importance of predicting stock prices accurately at market close. By comparing and contrasting the performance of these models on historical stock, futures, and ETF data, this paper aims to provide insights and practical recommendations for market participants and algorithmic traders. The findings are expected to contribute valuable perspectives to the field of financial forecasting and support the development of more efficient trading strategies.

1. Description of data
   1. data feature
      1. stock\_id: Identifier for the stock.
      2. date\_id: Identifier for the date.
      3. seconds\_in\_bucket: Time in seconds.
      4. imbalance\_size: Size of imbalance.
      5. imbalance\_buy\_sell\_flag: Flag indicating buy or sell imbalance.
      6. reference\_price: Reference price.
      7. matched\_size: Size of matched orders.
      8. far\_price: Far price.
      9. near\_price: Near price.
      10. bid\_price: Bid price.
      11. bid\_size: Size of the bid.
      12. ask\_price: Ask price.
      13. ask\_size: Size of the ask.
      14. wap: Weighted average price.
      15. time\_id: Identifier for the time.
      16. row\_id: Unique row identifier.
      17. currently\_scored: Boolean flag indicating if currently scored.
   2. Number of unique stock\_id, date\_id
      1. number of unique stock\_id count: 200
      2. number of unique date\_id count: 481
   3. Correlation Analysis
      1. pearson\_correlation
         1. imbalance\_size: 0.0010597508079042894
         2. reference\_price: -0.04473762032423138
         3. matched\_size: 0.0006216808516379594
         4. far\_price: -0.0018037281170692225
         5. near\_price: -0.002115035448593967
         6. bid\_price: -0.04775542189749235
         7. bid\_size: -0.017739631092372504
         8. ask\_price: -0.04838886498913048
         9. ask\_size: 0.012752139801386852
         10. wap: -0.056194051943220885
      2. spearman\_correlation
         1. imbalance\_size: 0.0010833034541324852
         2. reference\_price: -0.03203920862358736
         3. matched\_size: 0.0017232624948999111
         4. far\_price: -0.005374061289319017
         5. near\_price: -0.007609943577612957
         6. bid\_price: -0.03414298133949177
         7. bid\_size: -0.05963073825519416
         8. ask\_price: -0.03501851042029415
         9. ask\_size: 0.057489574839903836
         10. wap: -0.04282104755286715
      3. Sorted pearson\_correlation and spearman\_correlation

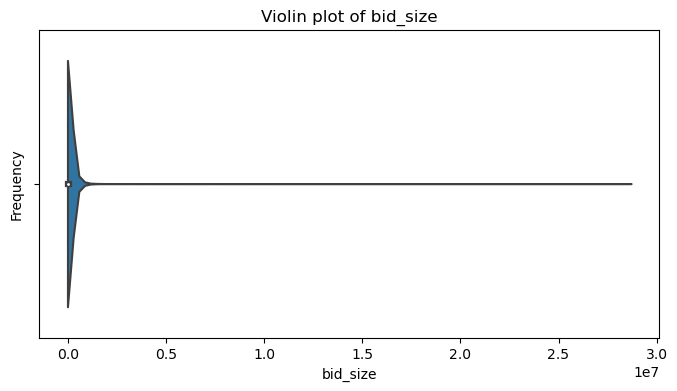
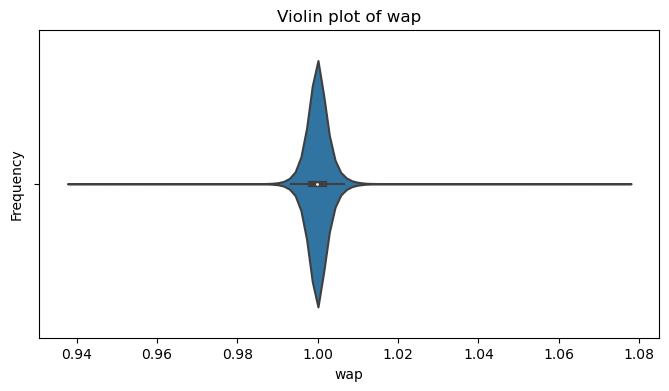
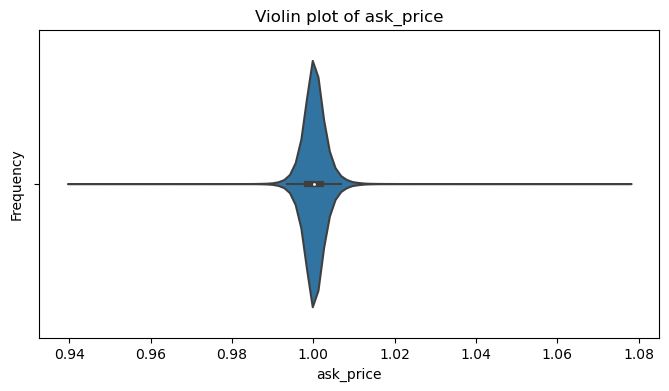
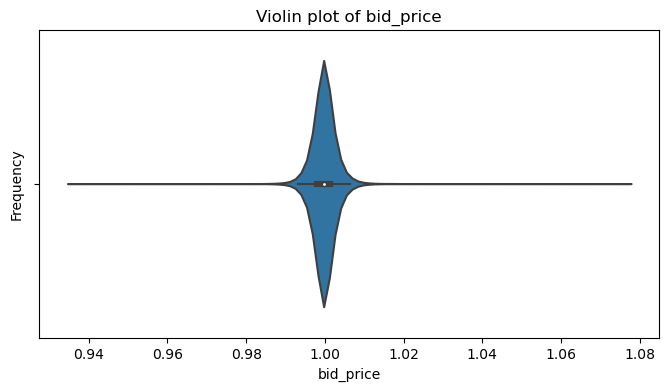
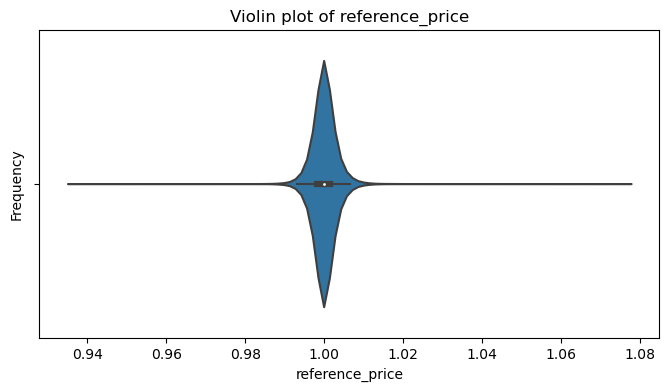
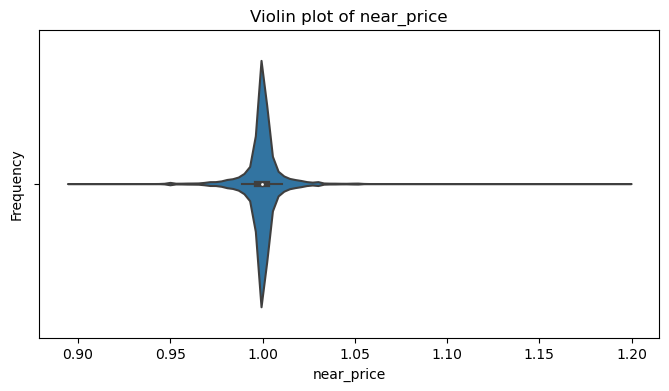
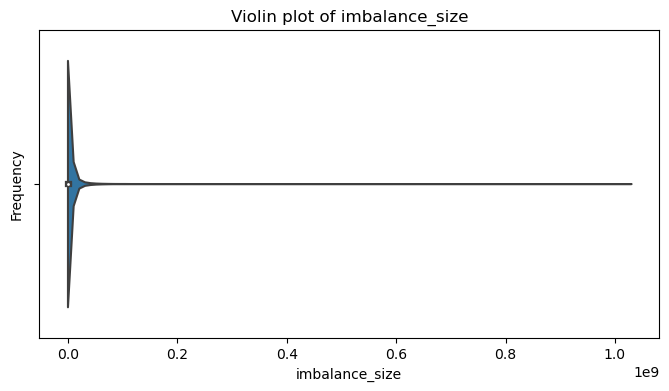
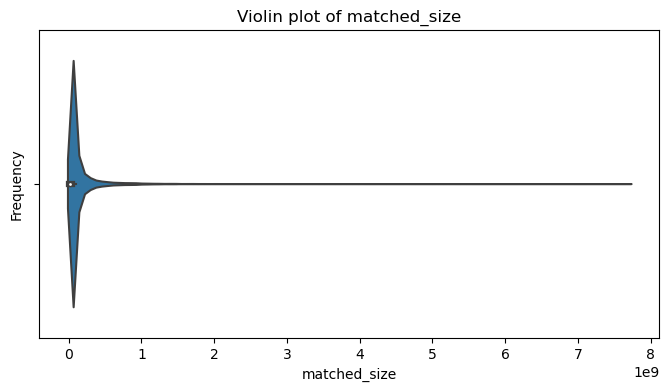
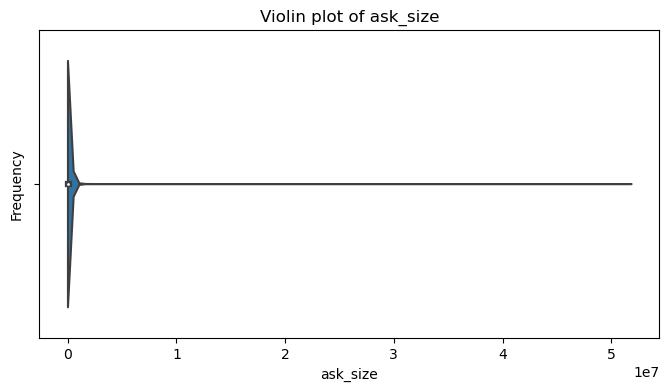
Sorted Pearson Correlation (absolute value):

* + - 1. wap: -0.056194051943220885
      2. ask\_price: -0.04838886498913048
      3. bid\_price: -0.04775542189749235
      4. reference\_price: -0.04473762032423138
      5. bid\_size: -0.017739631092372504
      6. ask\_size: 0.012752139801386852
      7. imbalance\_size: 0.0010597508079042894
      8. matched\_size: 0.0006216808516379594
      9. far\_price: -0.0018037281170692225
      10. near\_price: -0.002115035448593967

Sorted Spearman Correlation (absolute value):

1. bid\_size: -0.05963073825519416
2. wap: - 0.04282104755286715
3. ask\_price: -0.03501851042029415
4. bid\_price: -0.03414298133949177
5. reference\_price: -0.03203920862358736
6. near\_price: -0.007609943577612957
7. far\_price: -0.005374061289319017
8. ask\_size: 0.057489574839903836
9. matched\_size: 0.0017232624948999111
10. imbalance\_size: 0.0010833034541324852
    1. EDA, Distributions of data

Violin plot for all numeric attributes



1. Linear regression
   1. Preprocessing

Fill empty data with mean value

* 1. Features and target variable
     1. Features
        1. stock\_id
        2. seconds\_in\_bucket
        3. imbalance\_size
        4. imbalance\_buy\_sell\_flag
        5. reference\_price
        6. matched\_size
        7. far\_price
        8. near\_price
        9. bid\_price
        10. bid\_size
        11. ask\_price
        12. ask\_size
        13. wap
        14. time\_id
        15. row\_id
     2. Target variable

target

* 1. Train set and test set:
     1. Train dataset: 0.8
     2. Test dataset: 0.2
  2. Result

mean\_squared\_error: 6.315 (running)

mean\_absolute\_error: running

1. Literature review
   1. Overview of Stock Price Prediction Methods: Begin with a general overview of stock price prediction methods. Highlight the evolution from traditional statistical models to modern machine learning techniques, emphasizing the growing complexity and volume of financial data.
   2. LightGBM Regression in Finance: Discuss the emergence of LightGBM as a powerful machine learning tool in finance. Cite studies that have successfully employed LightGBM for financial time series prediction, noting its efficiency in handling large datasets and its ability to deal with overfitting.
   3. XGBoost's Role in Predictive Modeling: Examine the application of XGBoost in stock market forecasting. This section should cover its algorithmic advancements over traditional models and its strengths in capturing non-linear patterns.
   4. Linear Regression in Financial Markets: Revisit the role of Linear Regression, a fundamental statistical method, in predicting stock prices. This section can include its historical importance and current relevance, especially in simpler predictive models or as a baseline for comparison.
   5. The Use of ARIMA for Time-Series Forecasting: Detail the significance of ARIMA in time-series analysis, particularly in finance. Explain its methodology and how it has been traditionally used to forecast stock market trends, noting its limitations in dealing with non-stationary data.
   6. Comparative Studies and Hybrid Approaches: Highlight research that has compared these methods or used them in combination. Discuss the benefits and drawbacks of hybrid models, which integrate machine learning with traditional statistical methods for enhanced prediction accuracy.
   7. Recent Innovations and Trends: Conclude the review by discussing recent innovations and emerging trends in stock price prediction. This could include advancements in deep learning, the integration of alternative data sources, and the adaptation of these models to rapidly changing market conditions.

### Introduction to XGBoost:

XGBoost (Extreme Gradient Boosting) is an optimized distributed gradient boosting library designed to be highly efficient, flexible, and portable, and provides a parallel tree boosting (also known as GBDT, GBM) that solves many data science problems in a fast and accurate way.

Experiment and Improvement Attempts with XGBoost:

* Cross-validation and Initial runs:
  + Initial run using all features in the training dataset with Scikit-learn’s TimeSeriesSplit for Cross-Validation, with default setting for XGBRegressor.
  + Testing the effect of applying StandardScaler() on data preprocessing.
  + Evaluating the impact of using standard k-fold cross-validation.
* Exploring Feature Engineering:
  + PCA.
  + Testing the addition of engineered features, such as the movement of the reference price (first and second derivatives)
* Hyperparameter Tunings of the XGBoost Model:
  + Tuning hyperparameters like eta, gamma, max\_depth, min\_child\_weight, min\_delta\_step, subsample, colsample\_bytre, lambda, and alpha.
  + Using methods like grid search or random search for systematic hyperparameter optimization.

Results and Interpretation:

* Present the outcomes of the different preprocessing techniques, feature engineering, and hyperparameter tuning.
* Include accuracy metrics, feature importance, and other relevant insights.

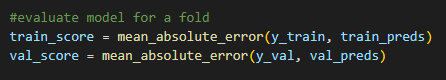
### XGBoost Model Initial Run Feedback

Model Performance Overview:

The initial trial of the XGBoost model has been conducted using a TimeSeriesSplit cross-validation with 10 folds. The model was set up with a reg:absolute error objective, utilizing a histogram-based tree method optimized for GPU usage. Random states were fixed across the environment to ensure reproducibility of results. No data scaling or feature selection preprocessing steps were applied prior to model training:

Model Training and Validation:

During the cross-validation process, the model was trained on each fold, and predictions were made for both training and validation sets. The mean absolute error (MAE) was used to evaluate the model's performance, and the results for each fold were recorded.



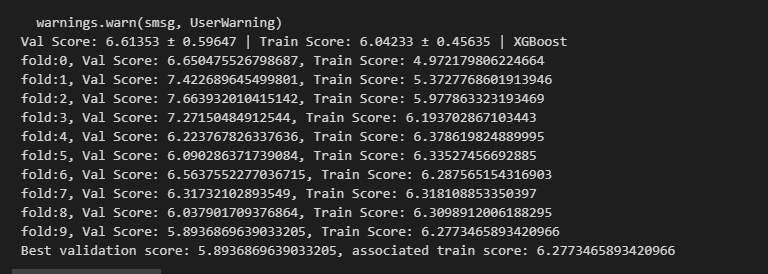


Figure: time series split scores

* Validation Scores: The validation scores across the ten folds show some variance, with the lowest being approximately 6.05 and the highest around 7.66. This suggests that the model's performance isn't consistent across different segments of the time series data.
* Training Scores: Training scores are relatively close to the validation scores within each fold, which is a good indication that the model isn't severely overfitting to the training data. However, the training scores are consistently higher than the validation scores, which is expected and also indicates some level of overfitting.
* Best Performance: The best validation score is around 5.89, which occurred in fold 9, with an associated training score of approximately 6.27.
* Fluctuations: The noticeable fluctuations in scores from fold to fold could be due to several factors
  + Changes in market behavior over time, which the model is sensitive to.
  + Possible overfitting to particular patterns present in certain folds but not others.

### The default model parameters may not be optimal across all segments of the time series.

### XGBoost Model Second Run Feedback with StandardScaler

Experiment Setup:

In the second run of the XGBoost model, the training features were scaled using StandardScaler. This aims to standardize the features by removing the mean and scaling to unit variance. Standardizing can often lead to better model performance especially for algorithms that are sensitive to the scale of the data.

Model Training and Validation:

The model training followed the same TimeSeriesSplit cross-validation with 10 folds as in the initial run. The only change in the process was the introduction of feature scaling.

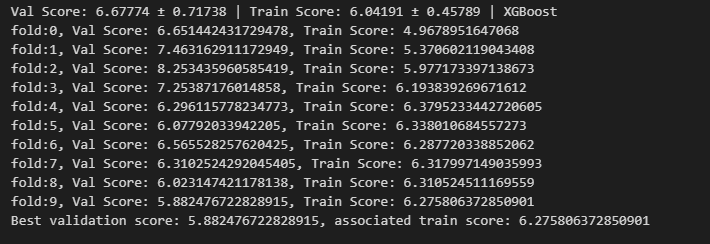


Figure: feature scaling scores

Model Performance Insights:

* Validation and Training Scores: Similar to the initial run, the validation and training scores across the ten folds varied. The scores ranged from approximately 6.1 to 7.4 on the validation set and from around 4.9 to 6.2 on the training set.
* Impact of Scaling: Applying StandardScaler seems to have impacted the validation scores positively, as indicated by a lower mean validation score compared to the initial run without scaling.
* Best Fold: The fold with the best validation score was fold 9, with a validation score of approximately 5.88 and an associated training score of around 6.27. This suggests that the model has managed to generalize well on this particular segment of the data.
* Score Variability: The standard deviation of the validation scores is lower than in the initial run, indicating that the model's performance is more consistent across different folds after scaling the features.

### XGBoost Model Analysis with K-Fold Cross-Validation

Experiment Summary:

In this iteration, the XGBoost model was subjected to a K-Fold cross-validation scheme with 10 splits. This strategy deviates from the previous time series split, potentially adjusting for temporal dependencies differently. The model parameters remained consistent with the initial setup to maintain comparability.

Model Performance Overview:

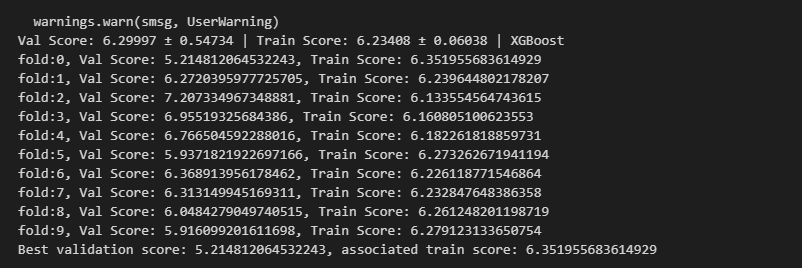


Figure: k-folder split scores

* Validation Scores: The model exhibited a range of validation scores, with the lowest being approximately 5.21 and the highest around 7.21. The variance in validation scores across folds suggests different levels of model performance for various segments of the dataset.
* Training Scores: Training scores were relatively stable, fluctuating between approximately 6.23 and 6.35. These scores are slightly more consistent than the validation scores, which is an expected outcome as the model is directly learning from the training data.

Consistency and Variability:

* The standard deviation of the validation scores in the K-Fold cross-validation is approximately 0.543, which is lower than the standard deviation observed in the previous time series split runs. This reduced standard deviation indicates a more consistent model performance across the different folds when using a K-Fold strategy.

Best Fold Performance:

* The best validation score was achieved in fold 0, with a score of approximately 5.21, and the associated training score for that fold was around 6.35. This suggests that for this particular data partition, the model's predictive accuracy was the highest.

### XGBoost Model Feedback with PCA Feature Engineering

Experiment Context:

The experiment incorporated Principal Component Analysis (PCA) into the feature engineering process. PCA is used for dimensionality reduction, transforming the data into a smaller set of uncorrelated variables while retaining most of the information. In this case, PCA was applied after standardizing the features and imputing missing values with the mean.

PCA Implementation:

* The PCA analysis determined that three principal components were sufficient to capture 95% of the variance in the data. This substantial reduction in feature space aims to focus the model on the most informative aspects of the data.

Model Training and Evaluation:

* The transformed dataset with three principal components was then used to train the XGBoost model using K-Fold cross-validation with 10 splits.
* The model's performance was evaluated using the mean absolute error (MAE) metric for both the training and validation sets.

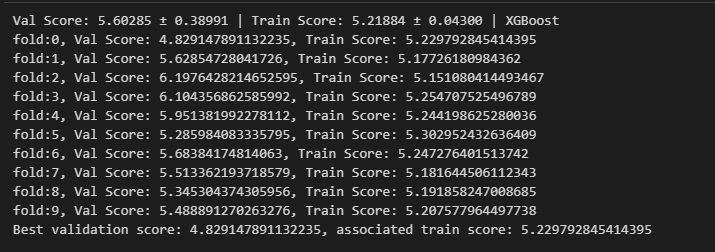


Figure: k-folder split with pca scores

Model Performance Overview:

* Validation Scores: The model achieved a range of validation scores across the 10 folds, from approximately 4.82 to 6.19. The variation in scores indicates how the model performed across different subsets of the transformed feature space.
* Training Scores: The training scores were consistent, with values closely clustered around the mean training score, which suggests the model trained effectively on the reduced feature set.

Consistency and Variability:

* The standard deviation of the validation scores in this PCA-based experiment was lower compared to the initial run without PCA, indicating improved consistency in the model's performance across folds.

Best Model Selection:

* The fold with the best validation performance (fold 0) achieved a score of approximately 4.82, and the associated training score was around 5.23, suggesting that the model was able to make accurate predictions on that particular subset of the data.

Concerns and Recommendations:

* Data Leakage: There is a risk that the model's performance is overestimated due to data leakage from using PCA on the entire dataset.

### **Feature Importance Analysis Report**

The feature importances derived from the XGBoost model provides valuable insights into which features are most influential in predicting the target variable. Here is a summary of the findings from the time series split and K-folder split run:

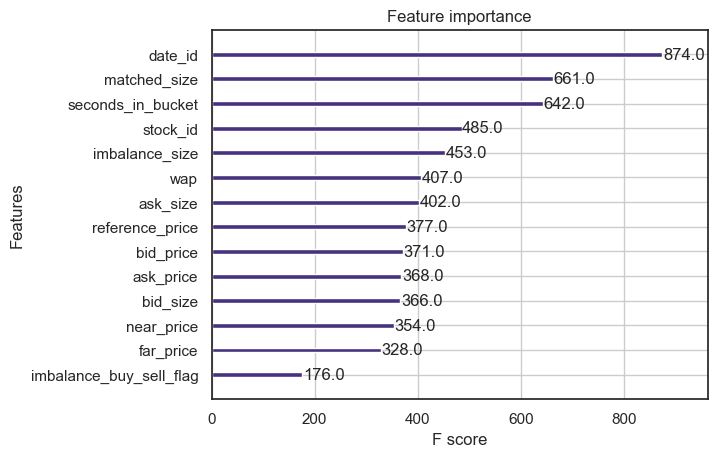


Figure: Feature importance from time series split

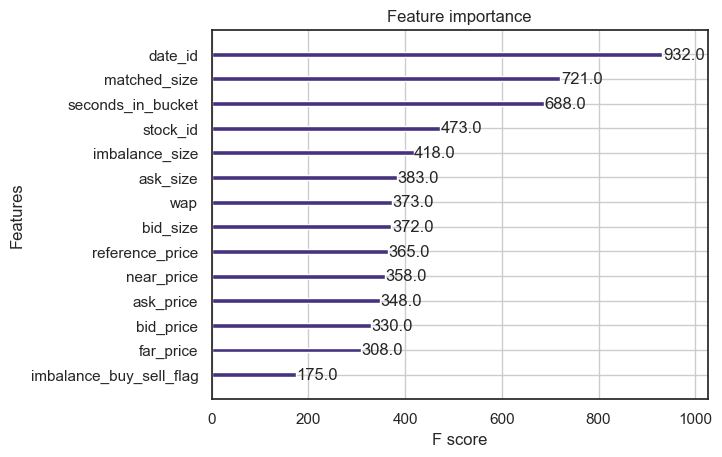


Figure: Feature importance from k-folder split

Comparing these results may reveal whether the model’s reliance on features changes when the temporal dimension is handled differently:

Temporal Features:

* date\_id: This feature consistently ranks as the most important across both TSS and K-Fold models, with even higher importance in the K-Fold validation (932 vs. 874). This underlines the strong effect of date-related factors on the target variable.
* seconds\_in\_bucket: There is a noticeable increase in importance in the K-Fold validation (688) compared to TSS (642), suggesting that this feature's predictive power is not solely dependent on the temporal order of data.

Transaction Size Features:

* matched\_size: Exhibits increased importance in the K-Fold cross-validation (721) versus TSS (661), maintaining its status as a highly predictive feature across both models.

Price Features:

* reference\_price, bid\_price, ask\_price: These features showed a slight decrease in importance in the K-Fold results, indicating a potential dependency on the temporal sequence of data for their predictive strength in the TSS model.
* wap (Weighted Average Price): The importance of this composite price feature is stable across both models, demonstrating its consistent value in predicting the target variable.

Order Size Features:

* bid\_size and ask\_size: Their importances remain relatively consistent across both models, with slight variations. This suggests that order sizes have a stable influence regardless of the validation strategy.

LightGBM Model

LightGBM (Light Gradient Boosting Machine) is a distributed high-performance framework that uses decision trees for ranking, classification, and regression tasks (Saha, 2023). In contrast to the level-wise(horizontal) growth in XGBoost, LightGBM carries out leaf-wise (vertical) growth that results in more loss reduction, and higher accuracy while being faster (Saha, 2023). However, this approach may lead to the overfitting of the training data (Saha, 2023).

For the experiment and the improvement attempts, the training data set will be used. The data set will be split for cross-validation with k = 10. The experiments with LightGBM are organized as below:

* Exploring preprocessing techniques:
  + An initial run was done with all the features available in the training data set, using Scikit-learn’s TimeSeriesSplit for Cross-Validation, without scaling the data using the StandardScaler(), and using all default hyperparameter values of LGBMRegressor.
  + Test the effect of applying StandardScaler() with the rest of the processing and setting equal to the initial run.
  + Test the effect of using normal k-fold cross-validation (non-time series data) with the rest of the processing and setting equal to the initial run.
  + Test the effect of applying PCA (principal component analysis) on the training data set
* Exploring Feature Engineering
  + The first test will check if removing seemingly unrelated features will affect or improve the training performance.
  + The second test will check if extra engineered features help.
    - First extra sets– incorporate the movement of the reference price, this includes the first derivative, which is the price of the current minus the price of the previous 10 seconds, and the second derivative, which is the change of the first derivative in the last 10 seconds.
    - Second extra sets – add the first and second derivatives for all 8 prices.
    - Third extra sets – Add the imbalance factors
  + The third test set tries to predict the two dependent variables which define the final target separately and combine the result.
* Exploring hyperparameter tunings of the LightGBM model
  + The hyperparameters considered include n\_estimators, learning\_rate, num\_leavers, max\_depth, min\_data\_in\_leaf, max\_bin, min\_gain\_to\_split and subsample.

Exploring preprocessing techniques

Initial Trial

Figure 1 shows the training and validation scores for the initial run using all features available in the training data set and Scikit-learn’s TimeSeriesSplit of 10 for cross-validation. The main observations are as follows:

Observation 1: The average validation scores are about 5% higher than the average training scores, this indicates potential overfitting of the model to the training data.

Observation 2: The validation score seems to be improving when more data is used for training. This indicates that the series might be stationary without a trend or seasonal component. Note when using the Scikit-learn’s TimeSeriesSplit function, the testing data of the first fold is added to the training data of the second fold (see Figure 2 for illustration), thus the training data set will continue to expand.

Since the number of days in the training data set is 480, one concern is the original split (TimeSeriesSplit of 10 will split the training data set into 11 parts) may contain fractional data of the day which affect the performance of the model. A quick test was performed using the TimeSeriesSplit setting of 9 (which split the data into 10 parts) with the result shown in table 1. As the resulting models didn’t perform significantly better than the previous models, it is concluded that the number of splits has no significant effect on the performance.

A screenshot of a computer screen

Description automatically generated

*Figure 1: Training scores and Validation scores for the initial run*

A screenshot of a computer

Description automatically generated

*Figure 2: Expanding Window Time Series Split Validation (Source: (Filho, 2023))*

Initial Trial with Standard Scaler Applied

This test applied StandardScaler() to the following numerical categories: imbalance\_size, reference\_price, matched\_size, far\_price, near\_price, bid\_price, bid\_size, ask\_price and ask\_size. The results are shown in Table 1. It doesn’t seem to have significant improvement compared to training data without StandardScaler() applied.

Initial Trial with k-fold cross-validation

This test uses the k-fold instead of TimeSeriesSplit for cross-validation. The results are shown in Figure 3. On average the model performs better compared to the model trained using the TimeSeriesSplit. Besides, there are two observations:

Observation 1: The is one part of the data (the first part) where the model predicts well.

Observation 2: The model performs better in the validation set when it is underfitting the training data. This happens when the validation score is lower than the training score achieved.

A screenshot of a computer screen

Description automatically generated

*Figure 3:* *Training scores and Validation scores for the initial run with k-fold cross validation*

Initial Trial with PCA

In this trial, Principle Component Analysis (PCA) was performed on the training data set hoping to improve the decision boundaries for the LightGBM model. However as PCA does not handle NaN values, there are two choices. The first choice is to drop all rows whenever one of the columns contains NaN values. However, there are too many NaN values in the far\_price and near\_price (about 2.8 million each), and dropping them will reduce the training data size to about 0.4 of the original. Although this approach yields good results (see Table 1), the good results applied only to the smaller dataset with NaN values available. Furthermore, the model won't be able to deal with new testing data that contains the NaN values. Hence this approach was not selected.

The second choice is to replace all the NaN values in the far\_price and near\_price columns with 0 while dropping other rows with NaN values. This approach preserves more than 99% of the training data. As shown in Table 1, this approach yields a minimal improvement compared to training with data without the PCA transformation.

Summary of Preprocessing Techniques

Table 1 shows the summary of the comparison of the performance of the models using different preprocessing techniques. Based on the results, it can be concluded that,

* The training data exhibits stationary property with variations independent of time, hence processing the data in time series is not necessary.
* The best-performing model is likely to be the model that underfits the training data.

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Test Title | Training Score | | Validation Score | |
| Average | Best Validate | Average | Best |
| Initial Run | 6.14128 | 6.34218 | 6.47656 | 5.88523 |
| With TimeSeriesSplit of 9 | 6.15360 | 6.34312 | 6.47738 | 5.90742 |
| With StandardScaler | 6.14125 | 6.34143 | 6.48478 | 5.88302 |
| With kfold | 6.29844 | 6.42056 | 6.32005 | 5.23449 |
| With PCA | 6.29813 | 6.42039 | 6.31976 | 5.23347 |
| With PCA(drop na) | 5.59804 | 5.69312 | 5.63493 | 4.77072 |

*Table 1: Summary comparison of the performance of the models for different preprocessing techniques*

Exploring Feature Engineering

Figure 4 shows the feature importance (Gain) for all the features used to train the best model. It seems that the feature ask\_size and bid\_size play a significant role compared to the other features. Figure 5 shows the feature importance (Split) for all the features used to train the best model. The distribution of the feature importance is more even compared to the feature importance (Gain).

A graph with numbers and text

Description automatically generated

*Figure 4:* *Feature Importance (Gain) for the base model*

A graph with numbers and a bar

Description automatically generated with medium confidence

*Figure 5: Feature Importance (Split) for the base model*

Removing Less Important Features

Based on the feature importance (Gain & Split) for the best base model shown before, an experiment was carried out by removing the four least important features (near\_price, stock\_id, far\_price, imbalance\_buy\_sell\_flag). The results are shown in table 2. Compared with the initial run, there is a slight drop in the average training score and validation score performance. As the extra features don’t affect the training speed and contribute slightly to the performance of the models, it is decided to keep these four features for future training.

Adding First and Second Derivatives

The idea is to try to capture the movement trend (first and second derivative) for the eight potential trending features (matched\_size, bid\_price, ask\_price, wap, ask\_size, bid\_size, reference\_price, and imbalance\_size). Taking the reference price as an example, the hope is that if the model knows the relative movement of the reference price (change in price in last 10 seconds) and the relative change in movement (change of change in price in last 10 seconds) it can learn better the direction where the target is heading.

Figures 6 & 7 show the performance and feature importance (gain & split) for the model trained with the first and second derivatives of the reference price. As shown in the figure the first and second derivatives have a minor impact on the gain and split of the model. As shown in Table 2, there is minimal improvement in the training results compared to the initial base model.

Figures 8 & 9 show the performance and feature importance (gain & split) for the model trained with the first and second derivatives of the eight potential trending features. the first and second derivatives have a minor impact on the gain and split of the model, with the second derivatives generally having less impact compared to the first derivatives. As shown in Table 2, there is a minimal improvement in the training results compared to the initial base model, and negligible improvement compared with the model trained using additional first and second derivatives of the reference price.

A graph with text on it

Description automatically generated

*Figure 6: Feature Importance (Gain) for model trained with first and second derivatives of reference price*

A graph with text on it

Description automatically generated

*Figure 7: Feature Importance (Gain) for model trained with first and second derivatives of reference price*

A screenshot of a graph

Description automatically generated

*Figure 8: Feature Importance (Gain) for model trained with first and second derivatives of 8 features*

A graph with text on it

Description automatically generated

*Figure 9: Feature Importance (Split) for a model trained with first and second derivatives of 8 features.*

Adding Imbalance Features

The imbalance features were proposed by one of the Kaggle competition participants (zhezhou, 2023). The results are shown in Table 2, Figures 10 & 11. As compared to the base model in Table 2, there is a minor improvement in the training results with the imbalanced features added in.

Based on Figures 10 & 11, one might wonder if only two imbalance features (imb\_s1 and imb\_s2) will be enough to improve the result. The test was carried out with only these two additional imbalance features. As shown in Table 2, there is a minor degradation of the overall training performance when only using two imbalanced features. Hence the suggestion is to keep all imbalanced features. The resulting test score using test data provided by the competition is 5.39 as shown in figure 13.

A graph with text and numbers

Description automatically generated

*Figure 10:* *Feature Importance (Gain) for model trained with imbalance features*

A graph with text and numbers

Description automatically generated

*Figure 11: Feature Importance (Split) for model trained with imbalance features*

A white rectangular object with a black line

Description automatically generated with medium confidence

*Figure 12: Test scores for the model trained with imbalance features*

Separate Model for the dependent variable

The prediction target is defined using the following formula (Forbes et al., 2023):

From the formula, only is known by the model, the model needs to predict both and . Thus the idea is to check if we create a model to predict these two term separately, will it perform better compared to the model predicting their result (the Target) directly?

Tests were conducted using two models with and without the imbalance factors with results shown in Table 2. It can be concluded that the two model approaches didn’t yield any improvement, probably due to the errors being compounded between the two models.

Pearson correlation and Spearman Correlation analysis were carried out between the columns and the target wap and target change in index separately. The results are shown in Figures 13 & 15. Based on the results, for target wap, only the columns wap, ask\_price, bid\_price, and reference\_price have moderate (absolute value of 0.3 – 0.5) relationship with the future wap, and this coincides well with the feature importance (gain & split) shown in figure 14 for the model to predict future wap.

For target change in the index, besides the target\_wap & target, none of the columns have a moderate or stronger relationship. The top five columns with weak relationships are wap, reference\_price, bid\_price, ask\_price, and near\_price. This coincides well with the feature importance (gain & split) shown in Figure 16 for the model to predict future wap.

A screenshot of a computer screen

Description automatically generated A screen shot of a computer

Description automatically generated

*Figure 13: Correlation between the columns and the target\_wap, sorted by absolute value*

A graph with text and numbers

Description automatically generatedA graph with numbers and text

Description automatically generated

*Figure 14: Feature Importance (Gain & Split) for the model to predict future wap*

A screen shot of a computer

Description automatically generated A screenshot of a computer

Description automatically generated

*Figure 15: Correlation between the columns and the target\_index, sorted by absolute value*

A graph with text and numbers

Description automatically generatedA graph with numbers and lines

Description automatically generated

*Figure 16: Feature Importance (Gain & Split) for the model to predict change in index*

Summary of feature engineering

Based on the experiment done against a set of features, it can be concluded that the best feature set will include the derivatives on the reference price and all the imbalance features. A final test was run using the best feature set. As shown in Table 2, this indeed generates the best average training scores, best average validation scores, and best validation scores from the cross-validation.

However, as it will be difficult to obtain the first and second derivatives with the test data provided, it is decided to use the imbalance factors only as extra features. The resulting test score is 5.39. The 2 models approach to predict the dependent variables separately was not selected as the validation scores and test scores are no better than the single model approach.

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| Test Title | Training Score | | Validation Score | | Test Score  (Submission) |
| Average | Best Validate | Average | Best |
| Initial Run with kfold | 6.29844 | 6.42056 | 6.32005 | 5.23449 | N/A |
| 4 features removed | 6.30591 | 6.42768 | 6.32560 | 5.23688 | N/A |
| With derivatives on Reference price | 6.29823 | 6.41934 | 6.31806 | 5.23228 | N/A |
| With derivatives on all trend features | 6.29823 | 6.41934 | 6.31804 | 5.23228 | N/A |
| With all imbalance features | 6.26616 | 6.38593 | 6.28535 | 5.1937 | 5.39 |
| With only two imbalance features | 6.29743 | 6.41905 | 6.31597 | 5.22403 | N/A |
| With derivatives on reference price & all imbalance features | 6.26397 | 6.38409 | 6.28376 | 5.19254 | N/A |
| 2 model | 6.42320 | 6.54786 | 6.47963 | 5.28060 | 5.4613 |
| 2 model with imbalance features | 6.39075 | 6.51506 | 6.43437 | 5.26175 | 5.4391 |

*Table 2: Summary comparison of the performance of the models for different feature sets*

Exploring Hyperparameter Tuning

An experiment with Hyperparameter tuning was carried out using the Optuna library with reference to one of the online guides (T., 2023).

Selected Hyperparameter for tuning are as follows:

* n\_estimators (default=100) – set to 10,000, controls the number of decision trees, a higher number may result in overfitting and longer training time. Often tuned together with the learning rate.
* learning\_rate (default=0.1) - range between 0.01 – 0.3, controls the learning speed. A smaller number leads to a slower learning rate, and the need to use early stopping rounds to terminate the training early to avoid excessive long training duration.
* num\_leaves (default=31) range between 20 – 3000, controls the number of decision leaves in a single tree. Range limit also depends on the max\_depth and should be 2^(max\_depth) according to LGBM documentation.
* max\_depth(default=-1) – range between 3 – 12, control the level of the tree. Lower numbers may lead to underfitting, higher numbers may lead to overfitting.
* min\_data\_in\_leaf(default=20) – range between 200 – 10000, specifies the minimum number of observations that fit the decision criteria in a leaf. Lower numbers may lead to overfitting.
* max\_bin(default=255) – range between 63 – 256. Smaller values increase the training speed while larger values increase the accuracy (Bahmani, 2023). For GPU training max\_bin limit is 256.
* min\_gain\_to\_split(default=0) – range between 0 – 15. Similar to XGBoosts’s gamma. Can be used as extra regularization in large parameter grids.

Figure 17 shows the best hyperparameter setting obtained from the first run. Figure 18 & Figure 19 shows the Optimization history and hyperparameter importance. As can be seen from Table 3, the model trained with tuned parameters performs slightly better with better average & best training and validation scores, as well as test score upon submission.

A computer screen shot of a black screen

Description automatically generated

*Figure 17: Best hyperparameters setting for first tuning (note optimization goal is minimum validation score achieved)*

A graph with a red line

Description automatically generated

*Figure 18: Optimization History Plot for first tuning*

A graph with blue bars

Description automatically generated

*Figure 19: Hyperparameter Importance ranking for first tuning.*

The second run included all parameter choices from the first run with the following additions/modifications:

* subsample (or bagging\_fraction, default=1) – specify the percentage of rows used per tree building iteration. Improved generalization but also speed of training (Bahmani, 2023).
* Change the n\_estimators to 1000
* Modify the range for num\_leaves to be within 2^(max\_depth-1) – 2^(max\_depth)
* Modify the range for max\_bin to between 32 and 256.

The best hyperparameter settings obtained are shown in Figure 20. As shown in Table 3, the average training scores and validation scores improved compared to the baseline, but the submission test score is slightly worse.

A computer screen shot of a black screen

Description automatically generated

*Figure 20: Best hyperparameters setting for second tuning (note optimization goal is minimum average validation score achieved)*

The third run included all parameter choices from the second run with only one modification:

* Change the n\_estimators setting from the default value (1000) to a suggestion range between 50 to 500 with a step of 50. As higher n\_estimators increased the training time required significantly with a potential risk of overfitting, this attempt is to find the n\_estimator that allows the model to train fast and generalize better.

The best hyperparameter settings obtained are shown in Figure 21. As shown in Table 3, the average training scores and validation scores improved compared to the baseline, and the submission test score is the best achieved thus far.

A computer screen shot of a number

Description automatically generated

*Figure 21: Best hyperparameters setting for third tuning (note optimization goal is minimum average validation score achieved)*

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| Test Title | Training Score | | Validation Score | | Test Score  (Submission) |
| Average | Best Validate | Average | Best |
| Baseline (with imbalance factors) | 6.26616 | 6.38593 | 6.28535 | 5.1937 | 5.39 |
| First Tuning | 6.19929 | 6.31821 | 6.27048 | 5.1798 | 5.3823 |
| Second Tuning | 6.22611 | 6.34055 | 6.27049 | 5.1807 | 5.3906 |
| Third Tuning | 6.23874 | 6.35838 | 6.27021 | 5.1807 | 5.3785 |

*Table 3: Summary comparison of the performance of the models for different hyperparameter tuning trial*

Reference

1. Saha, S. (2023, August 8). *XGBoost vs lightgbm: How are they different*. neptune.ai. <https://neptune.ai/blog/xgboost-vs-lightgbm>
2. Filho, M. (2023, July 12). How to do time series cross-validation in Python. Forecastegy (Alt + H). <https://forecastegy.com/posts/time-series-cross-validation-python/>
3. Forbes, T., Macgillivray, J., Pietrobon, M., Dane, S., & Demkin, M. (2023). Optiver - Trading at the Close. https://kaggle.com/competitions/optiver-trading-at-the-close
4. zhezhou, Y. (2023, September 23). *🥇🥇baseline LGB, xgb and catboost🥇🥇*. Kaggle. https://www.kaggle.com/code/yuanzhezhou/baseline-lgb-xgb-and-catboost/notebook
5. T., B. (2023, April 8). Kaggler’s Guide to lightgbm hyperparameter tuning with optuna in 2021. Medium. <https://towardsdatascience.com/kagglers-guide-to-lightgbm-hyperparameter-tuning-with-optuna-in-2021-ed048d9838b5>
6. Bahmani, M. (2023, September 4). Understanding LIGHTGBM parameters (and how to tune them). neptune.ai. https://neptune.ai/blog/lightgbm-parameters-guide