Contents

[LightGBM Model 1](#_Toc152992702)

[Exploring preprocessing techniques 3](#_Toc152992703)

[Initial Trial 3](#_Toc152992704)

[Initial Trial with Standard Scaler Applied 4](#_Toc152992705)

[Initial Trial with k-fold cross validation 4](#_Toc152992706)

[Initial Trial with PCA 5](#_Toc152992707)

[Summary for Preprocessing Techniques 5](#_Toc152992708)

[Exploring Feature Engineering 6](#_Toc152992709)

[Removing Less Important Feature 7](#_Toc152992710)

[Adding First and Second Derivatives 7](#_Toc152992711)

[Adding Imbalance Features 11](#_Toc152992712)

[Separate Model for the dependent variable 14](#_Toc152992713)

[Summary of feature engineering 17](#_Toc152992714)

[Exploring Hyperparameter Tuning 20](#_Toc152992715)

[Reference 22](#_Toc152992716)

# 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, 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 processing and setting equals to the initial run.
  + Test the effect of using normal k-fold cross-validation (non-time series data) with the rest of processing and setting equals 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 include the first derivative, which is the price of current minus the price of previous 10 seconds, and the second derivative, this 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 defined 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

# 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 follow:

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 into the training data of the second fold (see figure 2 for illustration), thus the training data set will continue expands.

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 TimeSeriesSplit setting of 9 (which split the data into 10 parts) with 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 affect to the performance.

A screenshot of a computer screen

Description automatically generated

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

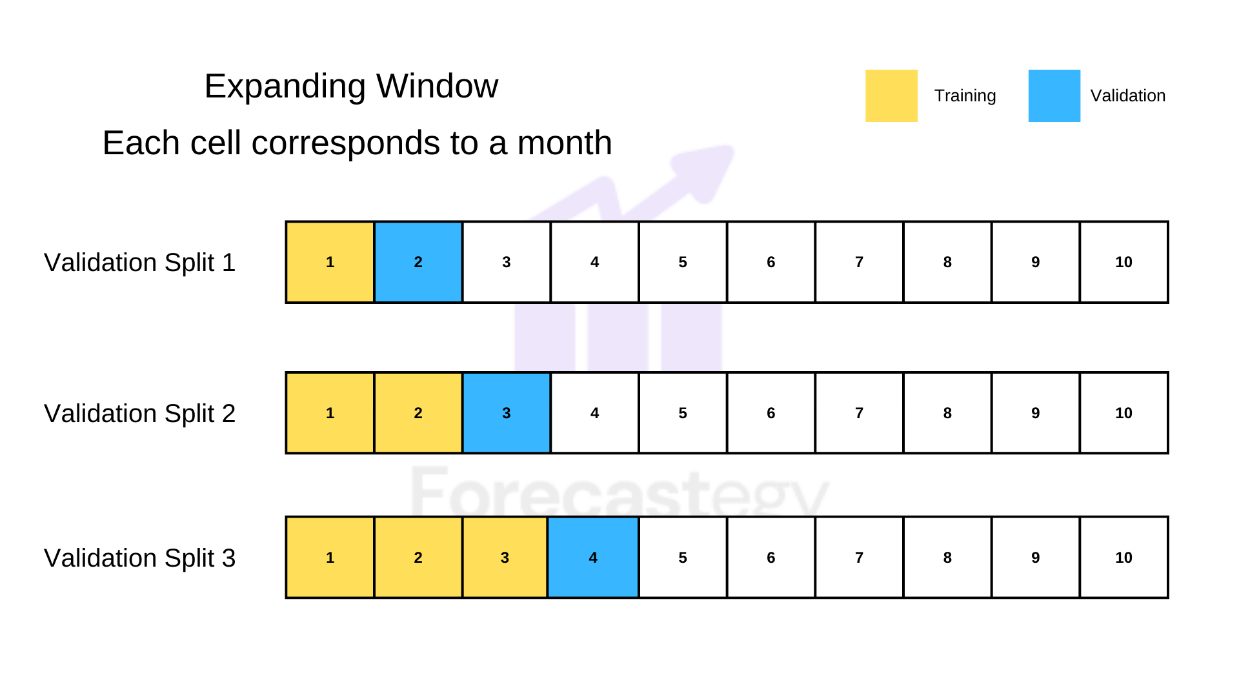


Figure 2: Expanding Window Time Series Split Validation (Source: (Filho, 2023))

## 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 table1. 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 underfit 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 contain 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 table1), the good results applied only to the smaller dataset with NaN values available. Furthermore the model wont 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 to 0 while dropping other rows with NaN value. This approach preserves more than 99% of the training data. As shown in table 1, this approach yielda minimal improvement compared to training with data without the PCA transformation.

## Summary for 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 conclude that,

* The training data exhibits stationary property with the 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 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 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 Feature

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. Comparing 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, hence 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 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.

Figure 6 & 7 show the performance and feature importance (gain & split) for model trained with the first and second derivatives of the reference price. As shown in the figure the first and second derivatives have 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.

Figure 8 & 9 show the performance and feature importance (gain & split) for model trained with the first and second derivatives of the eight potential trending features. the first and second derivatives have 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 minimal improvement on the training results compared to the initial base model, and negligible improvement compared with model trained using with additional first and second derivative 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 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, figure 10 & 11. As compared to the base model in table 2, there is minor improvement on the training results with the imbalance 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 minor degradation of the overall training performance when only using two imbalance features. Hence the suggestion is to keep all imbalance 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 need to predict both and . Thus the idea is to check if we create model to predict this 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 figure 13 & 15. Based on the results, for target wap, only the columns wap, ask\_price, bid\_price, 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 index, besides the target\_wap & target, none of the column have moderate or stronger relationship. The top five columns with weak relationship are wap, reference\_price, bid\_price, ask\_price, 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 score 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 performance of the models for different feature sets

# Exploring Hyperparameter Tuning

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

Selected Hyperparameter for tuning are as follow:

* n\_estimators (default=100) - range up to 10,000, controls the number of decision tree, 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. Smaller number leads to slower learning rate, need to use early stopping round to control.
* 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 number 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 value increase the training speed while larger value 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 hyperparameters 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 perform slightly better with better average & best training and validation scores, as well as test score upon submission.

The second run includes the following hyperparameter.

* 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 32-256.

A computer screen shot of a black screen

Description automatically generated

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

A graph with a red line

Description automatically generated

Figure : Optimization History Plot for first tuning

A graph with blue bars

Description automatically generated

Figure : Hyperparameter Importance ranking for first tuning.

A computer screen shot of a black screen

Description automatically generated

Figure : Best hyperparameters setting for second 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 |  |  |  |  |  |
| Third Tuning |  |  |  |  |  |

Table : Summary comparison of 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