Model Evaluation on LightGBM vs. Catboost:
A Study on Nasdaq-listed Stock Closing Auction Price Prediction

INF2207H S: Practical Elements of Responsible AI Development

Junwei Shen & Che Zhu

April 14, 2024

Table of Contents

Abstract 2
1. Introduction 2
2. Methodology 2
    2.1 Dataset 2
    2.2 Preprocessing & Feature Engineering 3
    2.3 Modelling 3
3. Results 4
    3.1 EDA 4
    3.2 Accuracy Results Comparison 6
    3.3 Feature Importance Comparison 7
4. Discussion: 9
5. Conclusion 10
Reference 11
Appendix 11
    Feature data dictionary 11
    Code: Memory Optimization 12
    Code: Feature Engineering 12

LGBM vs. CatBoost

Evaluation Process: Suppose there is a feature about industry info, we can only include a set of industries so the model would be biased.
An example of industrial category analysis -> ethical problem (公司作假)
Usage of the model: ethical issue

Abstract

1. Introduction

In the rapidly evolving environment of financial markets, accurately predicting market movements is a cornerstone for trading strategies. With the advantages of machine learning, several algorithms push the limits of predictive analysis, making it possible to accurately forecast future price movements given vast quantities of market data. Therefore, this study will utilize two cutting-edge gradient boosting models, LightGBM and Catboost, to predict the future price movements of stocks through a synthetic index composed of NASDAQ-listed stocks. Through a methodical approach encompassing data preprocessing, feature engineering, and model training and tuning, we strive to minimize the mean absolute error (MAE), and thus increase the accuracy of our predictions. In addition, considering the responsible learning aspects in AI development, the study will present a model comparison analysis of the two gradient-boosting models. In doing so, we not only compare the efficacy of LightGBM and CatBoost in the domain of stock market prediction but also contribute to the broader understanding of how machine learning can be harnessed to navigate the complex environment of financial markets.

2. Methodology

This section of the report will discuss the methodologies implemented for the model comparison between LightGBM and Catboost. It is important to note that all the coding and modeling were performed on Google Colab using the coding language Python.

2.1 Dataset

The study utilizes a dataset containing historical data for the daily ten-minute closing auction on the NASDAQ stock exchange. We obtained the training data via Kaggle through the competition "Optiver - Trading at the Close", which can be found at https://www.kaggle.com/competitions/optiver-trading-at-the-close/overview. Specifically, the outcome we are measuring is the 60-second future move in the weighted average price (wap) of the synthetic index, where the index is a custom weighted index of Nasdaq-listed stocks constructed by Optiver. The target is defined by the equation:

$$Target = \left(\frac{StockWAP_{t+60}}{StockWAP_{t}} - \frac{IndexWAP_{t+60}}{IndexWAP_{t}} \right) * 10000$$

In terms of explanatory variables, the dataset contains mostly numerical features including stock_id, date_id, imbalance_size, imbalance_buy_sell_flag, buy-side imbalance, reference_price, matched_size, far_price, near_price, [bid/ask]_price, [bid/ask]_size, wap, and seconds_in_bucket. A detailed description of each feature can be found in the Appendix Feature data dictionary section. In addition, a set of 200,000 samples are randomly selected from the 1048575 population to create a representation for our study.

2.2 Preprocessing & Feature Engineering

To improve model accuracy (as mentioned before to minimize MAE) and efficiency, we have introduced several preprocessing techniques and feature engineering processes.

$$MAE = \frac{1}{n}\sum_{i=1}^{n} |yi - xi|$$

Firstly, we have optimized the memory usage of pandas DataFrames by downcasting the data types of its columns to more memory-efficient types while maintaining the data integrity (see Appendix Code: Memory Optimization for example). Moreover, we have constructed new features based on given explanatory variables. We believe that this will provide more suitable information for modelling. Specifically, 1) we have defined a function to calculate various features related to the market's balance, such as volumn (sum of ask_price and bid_price), liquidity_imbalance (difference between bid size and ask size relative to their sum), etc.; and 2) we have defined functions to consider the time/log-effects, such as dow (the day of the week, calculated based on the date_id), time_to_market_close (remaining time until the market closes, assuming a closing time 540 seconds after the start of the bucket), etc. (see Appendix Code: Feature Engineering for example). In addition, the train-test split was utilized on the training dataset mentioned in the dataset section, where the variable “Target” is the only outcome we are measuring. Lastly, the null values are removed for both split train and test datasets.

2.3 Modelling

Both LightGBM and Catboost models are constructed under identical computing environments on Google Colab to ensure fairness. The two models were trained on the same explanatory features, including all of the original features and new features generated by feature engineering. For hyperparameter tuning, we have utilized the same tool “Optuna” to achieve the objective of “minimizing mae”. However, due to time constraints and different models’ computing power consumption, we have run 200 trials on the LightGBM model and 50 trials on the Catboost Model for the best hyperparameters. We have then measured MAE to compare the models’ performance through predictions on the testing set. In addition, the study also includes an investigation of feature importance after fitting and constructing our model.

3. Results

3.1 EDA

The EDA explores the relationship between different features and the outcome variable, and the study mainly utilizes the EDA results for feature selection and engineering purposes. It is important to note that both the LightGBM and Catboost models refer to the same EDA results. Below in Figure 1: Correlation Matrix of originally given variables, we have investigated the correlation between original variables. The result illustrates several considerations: 1) There exists several correlations between different variables (e.g.far_price vs. imbalance_size, seconds_in_bucket vs. matched size, etc.); 2) besides relatively highly correlated with bid_price, ask_price, bid_size and reference_price, the target variable also has a medium-level correlation with time_id, this suggests creating new variables to explain time/lag effects in feature engineering process.

	stock_id	date_id	seconds_in_bucket	imbalance_size	imbalance_buy_sell_flag	reference_price	matched_size	far_price	near_price	bid_price	bid_size	ask_price	ask_size	wap	target	time_id	bid_ask_spread
stock_id	nan	nan	nan	nan	nan	nan	nan	nan	nan	nan	nan	nan	nan	nan	nan	nan	nan
date_id	nan	nan	nan	nan	nan	nan	nan	nan	nan	nan	nan	nan	nan	nan	nan	nan	nan
seconds_in_bucket	nan	nan	1.000000	-0.784569	0.846462	-0.637386	0.897932	0.606272	0.568233	-0.605080	0.709487	-0.692569	0.368454	-0.632420	0.366416	1.000000	-0.539575
imbalance_size	nan	nan	-0.784569	1.000000	-0.774157	0.444467	-0.957382	0.894541	0.874577	0.416353	-0.402439	0.503421	-0.160142	0.441818	-0.021499	-0.784569	0.503625
imbalance_buy_sell_flag	nan	nan	0.846462	-0.774157	1.000000	-0.282239	0.893207	0.898726	0.879842	-0.253212	0.642836	-0.336374	0.248013	-0.272449	0.037020	0.846462	-0.455093
reference_price	nan	nan	-0.637386	0.444467	-0.282239	1.000000	-0.442871	0.808334	0.829946	0.955525	-0.273796	0.983706	-0.255517	0.974539	-0.519676	-0.637386	0.310357
matched_size	nan	nan	0.897932	-0.957382	0.893207	-0.442871	1.000000	0.733979	0.678607	-0.429605	0.571235	-0.517917	0.254120	-0.454004	0.128201	0.897932	-0.512129
far_price	nan	nan	0.606272	0.894541	0.898726	0.808334	0.733979	1.000000	0.991773	0.806569	0.493661	0.808334	0.160093	0.823064	-0.037264	0.606272	-0.334107
near_price	nan	nan	0.568233	0.874577	0.879842	0.829946	0.678607	0.991773	1.000000	0.827728	0.445619	0.829946	0.142960	0.843453	-0.078234	0.568233	-0.338061
bid_price	nan	nan	-0.605080	0.416353	-0.253212	0.955525	-0.429605	0.806569	0.827728	1.000000	-0.276919	0.979302	-0.244418	0.993973	-0.627353	-0.605080	0.077576
bid_size	nan	nan	0.709487	-0.402439	0.642836	-0.273796	0.571235	0.493661	0.445619	-0.276919	1.000000	-0.323280	0.545915	-0.277065	0.485468	0.709487	-0.278076
ask_price	nan	nan	-0.692569	0.503421	-0.336374	0.983706	-0.517917	0.808334	0.829946	0.979302	-0.323280	1.000000	-0.273585	0.992036	-0.576848	-0.692569	0.277763
ask_size	nan	nan	0.368454	-0.160142	0.248013	-0.255517	0.254120	0.160093	0.142960	-0.244418	0.545915	-0.273585	1.000000	-0.283535	0.363111	0.368454	-0.187551
wap	nan	nan	-0.632420	0.441818	-0.272449	0.974539	-0.454004	0.823064	0.843453	0.993973	-0.277065	0.992036	-0.283535	1.000000	-0.598772	-0.632420	0.168900
target	nan	nan	0.366416	-0.021499	0.037020	-0.519676	0.128201	-0.037264	-0.078234	-0.627353	0.485468	-0.576848	0.363111	-0.598772	1.000000	0.366416	0.136146
time_id	nan	nan	1.000000	-0.784569	0.846462	-0.637386	0.897932	0.606272	0.568233	-0.605080	0.709487	-0.692569	0.368454	-0.632420	0.366416	1.000000	-0.539575
bid_ask_spread	nan	nan	-0.539575	0.503625	-0.455093	0.310357	-0.512129	-0.334107	-0.338061	0.077576	-0.278076	0.277763	-0.187551	0.168900	0.136146	-0.539575	1.000000

Figure 1: Correlation Matrix of originally given variables

In Figure 2 and Figure 3, we explore the trend of various features over the time variable seconds_in_bucket. The results illustrated the following:

- imbalance_size: Appears to fluctuate significantly over time, with peaks and valleys, representing the size of an imbalance in order book depth.
- imbalance_buy_sell_flag: Shows binary-like values, indicating the direction of the imbalance (buying or selling) at different times.
- reference_price: Varies over time, indicating changes in a baseline price for comparison purposes.
- matched_size: Shows a step-like increase, indicating accumulated matched trades over time.
- far_price and near_price: Both prices show fluctuations, which could represent different levels of prices at which orders were placed away from the mid-price.
- bid_price and ask_price: These show bid and ask prices over time, critical for understanding the spread and liquidity created in feature engineering.

- bid_size and ask_size: Represent the sizes at bid and ask prices, respectively, which can indicate market depth.
- wap: this is an important metric to assess the target at which trades are executed.
- The prices follow a similar trend, with all experiencing a significant drop and then a rise. This concurrent movement suggests a market event that affected all prices similarly (possibly a periodic effect).
- The target line shows a significant divergence from price lines, indicating that the target variable might not be directly correlated with these prices (a derivative metric like volatility, and returns could be better choices for the feature).
- The target line fluctuating in a different scale implies that its behavior or scale is different from that of the prices, which could complicate the task of predicting it directly from these prices without considering other factors or transformations.

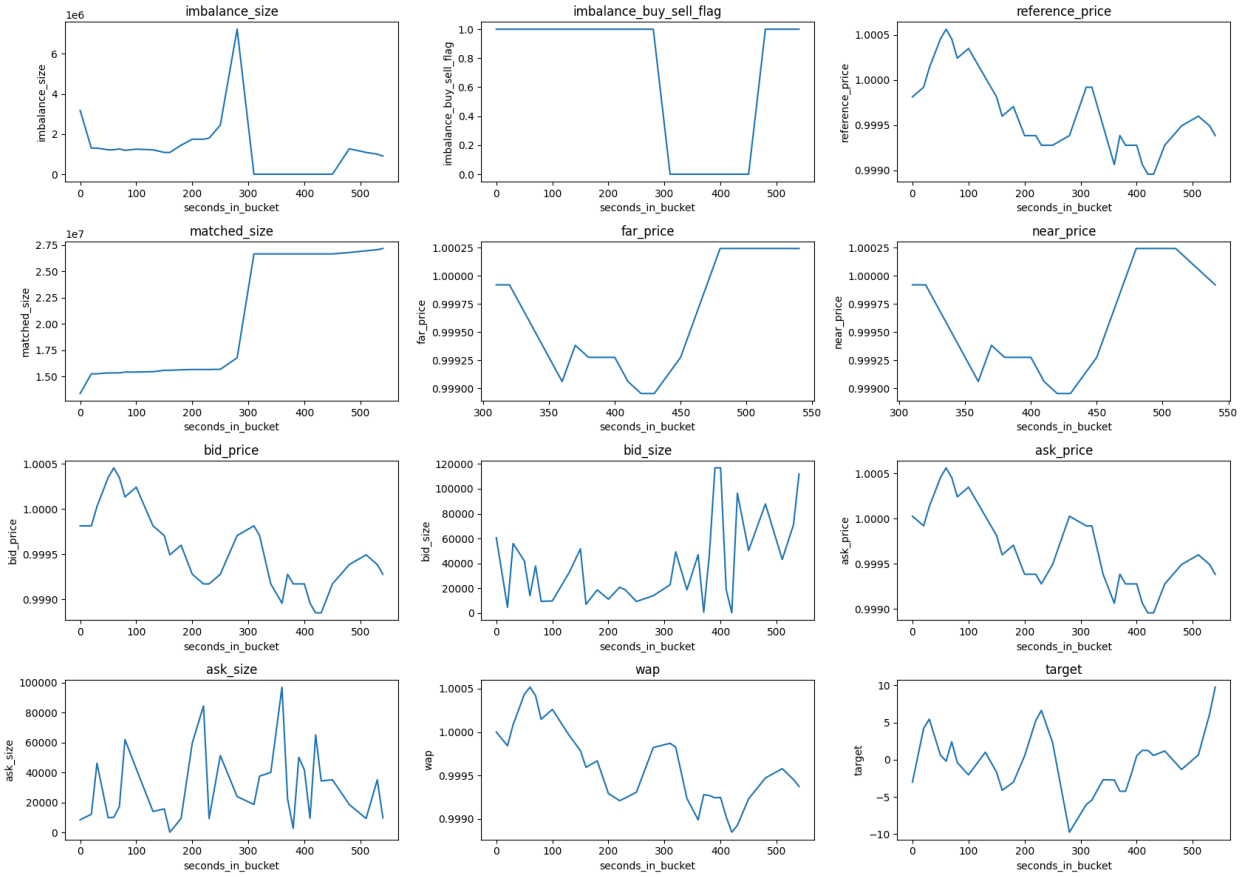

Figure 2: seconds-in_bucket vs. all other Features

Figure 3: seconds_in_bucket vs. all other Features on the same axes

3.2 Accuracy Results Comparison

Figure 4, Figure 5, and Table 1 illustrate the result of the major evaluation metric of this study, MAE. Specifically, compared to a variance of the target (59.80), global minimum (-103.03), and global maximum (113.18), both models have relatively small MAE. The Catboost model slightly outperforms the LightGBM model with an MAE of 4.89 (<4.986 in the LightGBM model). However, it is important to mention that there is a notable difference in runtime for these two models. We stated earlier that due to computing power and time constraints, the study was only able to run 50 trials of Optuna hyperparameter tuning for the Catboost model, while it was able to run 200 trials for the LightGBM under similar conditions within the same period of time. In addition, the runtime for the final model training is also significantly longer for the Catboost model compared to the LightGBM model.

```
--------------------------------------------------------------------------------------------------
Model LGBM 5 : LGBMRegressor(bagging_fraction=1.0, bagging_freq=8,
              colsample_bytree=0.875635157953628,
              learning_rate=0.09675713378758799, max_depth=13,
              min_child_samples=38, n_estimators=494, num_leaves=146,
              objective='mae', reg_alpha=0.031598830941276745,
              reg_lambda=0.26466457985079583, subsample=0.8846236833272005)
--------------------------------------------------------------------------------------------------
[LightGBM] [Warning] bagging_fraction is set=1.0, subsample=0.8846236833272005 will be ignored. Current value: bagging_fraction=1.0
[LightGBM] [Warning] bagging_freq is set=8, subsample_freq=0 will be ignored. Current value: bagging_freq=8
[LightGBM] [Warning] bagging_fraction is set=1.0, subsample=0.8846236833272005 will be ignored. Current value: bagging_fraction=1.0
[LightGBM] [Warning] bagging_freq is set=8, subsample_freq=0 will be ignored. Current value: bagging_freq=8
LGBM's MAE CV Score:>  5.051
LGBM's MAE Test Score:>  4.986
--------------------------------------------------------------------------------------------------
```

Figure 4: LightGBM MAE Result

```python
# Define CatBoost Regressor
model = CatBoostRegressor(
    iterations=726,
    learning_rate=0.089162404263726,
    depth=10,
    eval_metric='MAE',
    random_seed=42,
    l2_leaf_reg=4.7111294990008,
    border_count=139,
    verbose=False
)

# Fit model
model.fit(
    X_train,
    y_train,
    eval_set=(X_test, y_test),
    cat_features=None,
    use_best_model=True
)

# Make predictions
predictions = model.predict(X_test)

# Evaluate the model
mae = mean_absolute_error(y_test, predictions)
print(f'Mean Absolute Error on validation set: {mae}')
```

```
Mean Absolute Error on validation set: 4.891842609226176
```

Figure 5: Catboost MAE Result

	Statistic	Value
0	Variance	59.796947
1	Minimum	-103.030205
2	Maximum	113.179688

Table 1: Target variance, max, and min in the Testing set

3.3 Feature Importance Comparison

Figure 6 below illustrates the results of the (Top 40) feature importance ranking of LightGBM and Catboost models. It provides insights into how each model processes and values different information. Note that the models have different significance scaling, this is because the LightGBM model does not have the built-in get_features() function. As such, we defined a function for evaluating the importance of the features in LightGBM. Specifically, certain features are highly ranked in both results, implying that these features are strong predictors regardless of the model type. For instance, matched_size and bid_price are significant in both models. They have a strong correlation with the target variable and might be consistently useful across other modelling approaches as well. However, some features are given different significance levels in the ranking, this is the result of the algorithms' functionalities. LightGBM employs a leaf-wise (best-first) tree growth strategy, which may make it more sensitive to certain features early on in

the training process. This will potentially assign them more importance. On the other hand, Catboost tends to create balanced trees, which would distribute importance more evenly across features or emphasize different features due to the symmetric tree growth strategy. In addition, Catboost has the advantage of detecting complex interactions between features, meaning that if a feature is part of a significant interaction, it might be more important in the Catboost model compared to the LightGBM model. Detailed exploration of the models will be given in later sections.

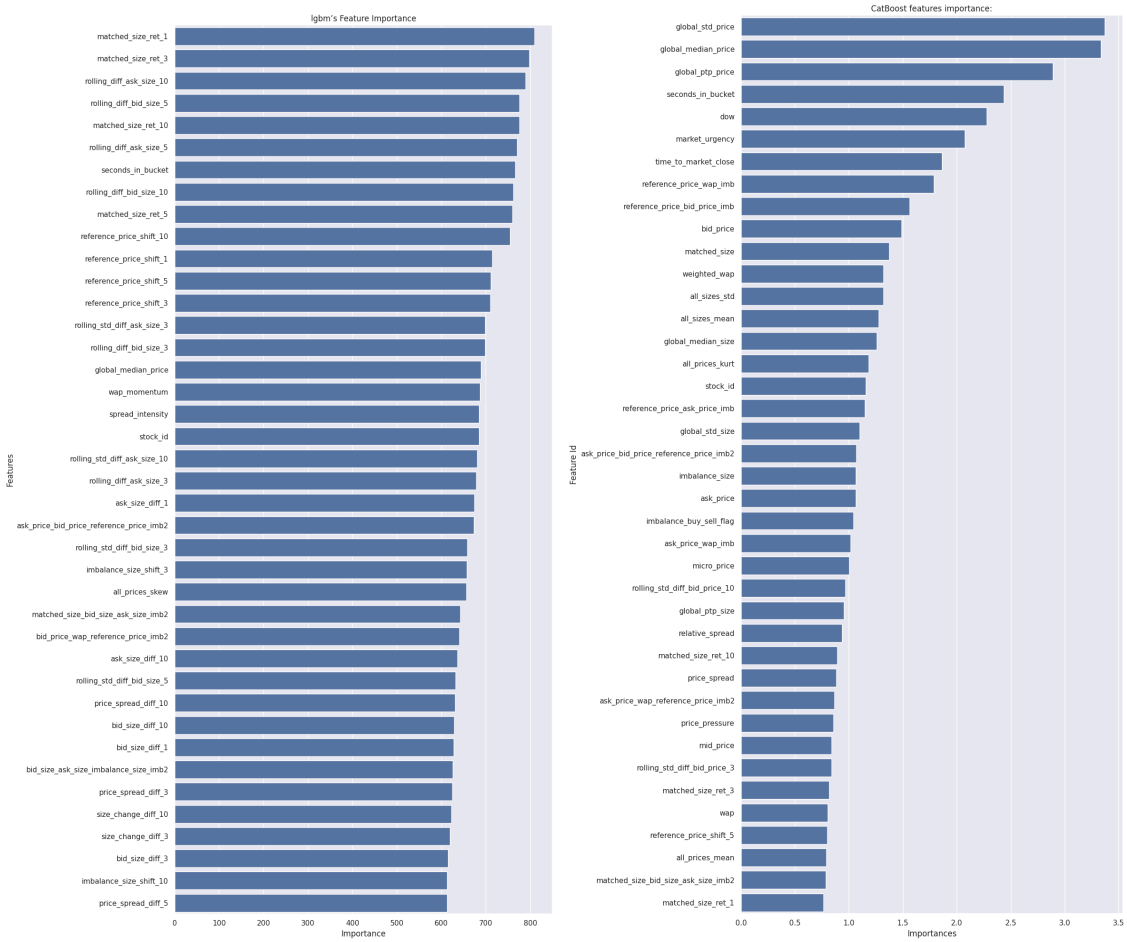

Figure 6: Feature importance of LightGBM (left) vs. Catboost (right)

4. Discussion:

4.1. Model Comparison

Real-world data collected from the stocking market often contains large amounts of categorical data such as industry classification or company name. Despite the data utilized in this study being formed solely by numerical entries, the implications from the models' performance illustrate the distinctions between Catboost and LightGBM As a part of the gradient boosting family, CatBoost and LightGBM are both popular tree-based algorithms that are known for their efficiency speed, and performance, but distinguish themselves through unique optimization and handling of categorical data.

For CatBoost, the library is designed to provide optimized compatibility for categorical data without the need for extensive data preprocessing like one-hot encoding. Unlike traditional gradient boosting methods, Catboost avoids suffering from prediction shifts introduced by overlapped use of train-test data by dividing the dataset into multiple random permutations. In terms of categorical data handling, Catboost converts categorical values into a numerical ‘bundle’ through the ordered boosting and target encoding techniques to reduce the risk of overfitting. The model therefore gains unique advantages when categorical data is frequently present in the desired data. On the other hand, LightGBM handles categorical features by converting them to integers and then using these integers to construct histograms. Under its histogram-based algorithm, the intrinsic relationships between categories are at risk of being overseen. As the library is designed for distributed and efficient training, the superior capability to process larger datasets as well as equipped with gradient-based one-side sampling and exclusive feature bundling, grant LightGBM's unparalleled ability to handle all kinds of complex ML tasks. Overall, benefiting from Ordered Boosting and symmetric trees techniques, Catboost gains computing advantages by ensuring predictions are not overly dependent on any specific arrangement or correlation present in the training data, therefore, better performance when data contains correlated features like the auction data used for the study (illustrated in the correlation map). However, LightGBM is able to produce predictions with a similar level of accuracy with much less computing time.

4.2. Bias Consideration

While the released data from the stock closing auction is considered to have high accountability and transparency due to financial service regulations, biases could occur during the training process, and by external influences.

Unlike models that are used to predict natural patterns or customer behaviour, stock market behaviour as a human product is prone to non-stationary and fast structural changes caused by major economic shifts (2008 finance crisis), regulatory changes (reduction of interest), Capital intervention (short selling) and global events (warfare between Russia and Ukraine). Trained

model by design and learn from history and will implicitly assume the past pattern will repeat in the future. This assumption can introduce biases and hurt the ability of generalization when models fail to adopt the momentary trend in the market. Moreover, although data collected from closing auction is relatively objective without much human involvement, the process of data selection could still introduce biases when the data included is not representative of real-world scenarios, especially when only a particular sector of the market or a specific time period with atypical market behaviour is select for the machine to learn. In practice, when data from certain companies or sectors are deliberately filtered out during the selection process, the reduced predicting power on these companies can be used to gain competitive advantages by misleading investors to make faultful trading decisions.

4.3. Mitigation Measures

To meet the challenge of frequent structural change in the stock market, some mitigation measures can be applied. First, use rigorous cross-validation techniques and out-of-sample testing to guard against overfitting. In terms of the data source, one should consider using a wide range of market conditions and consider using data on companies that failed or were delisted to avoid survivorship bias. Externally, enhancing regulation and public supervision shall force the increase in model transparency and accountability.

5. Conclusion

Reference

Ke, G., Meng, Q., Finley, T., Wang, T., Chen, W., Ma, W., Ye, Q., & Liu, T.-Y. (1970, January 1). *LightGBM: A highly efficient gradient boosting decision tree*. Advances in Neural Information Processing Systems. https://papers.nips.cc/paper/6907-lightgbm-a-highly-efficient-gradient-boosting-decision-tree

Optiver - trading at the close. Kaggle. (n.d.).https://www.kaggle.com/competitions/optiver-trading-at-the-close/overview

State-of-the-art open-source Gradient Boosting Library with categorical features support. CatBoost. (n.d.). https://catboost.ai/

Welcome to LIGHTGBM's documentation!.LightGBM 4.0.0 documentation. (n.d.). https://lightgbm.readthedocs.io/en/stable/

Appendix

Feature data dictionary

stock_id - A unique identifier for the stock. Not all stock IDs exist in every time bucket.
date_id - A unique identifier for the date. Date IDs are sequential & consistent across all stocks.
imbalance_size - The amount unmatched at the current reference price (in USD).
imbalance_buy_sell_flag - An indicator reflecting the direction of auction imbalance.
buy-side imbalance; 1
sell-side imbalance; -1
no imbalance; 0
reference_price - The price at which paired shares are maximized, the imbalance is minimized and the distance from the bid-ask midpoint is minimized, in that order. Can also be thought of as being equal to the near price bounded between the best bid and ask price.
matched_size - The amount that can be matched at the current reference price (in USD).
far_price - The crossing price that will maximize the number of shares matched based on auction interest only. This calculation excludes continuous market orders.
near_price - The crossing price that will maximize the number of shares matched based auction and continuous market orders.
[bid/ask]_price - Price of the most competitive buy/sell level in the non-auction book.
[bid/ask]_size - The dollar notional amount on the most competitive buy/sell level in the non-auction book.
wap - The weighted average price in the non-auction book.

seconds_in_bucket - The number of seconds elapsed since the beginning of the day's closing auction, always starting from 0.

Note: Dictionary provided by Kaggle Optiver - Trading at the Close Competition data description at https://www.kaggle.com/competitions/optiver-trading-at-the-close/data.

Code: Memory Optimization

```
def reduce_mem_usage(df, verbose=0):
    start_mem = df.memory_usage().sum() / 1024**2
    for col in df.columns:
        col_type = df[col].dtype
        if col_type != object:
            c_min = df[col].min()
            c_max = df[col].max()

            if str(col_type)[:3] == "int":
                if c_min > np.iinfo(np.int8).min and c_max < np.iinfo(np.int8).max:
                    df[col] = df[col].astype(np.int8)
                elif c_min > np.iinfo(np.int16).min and c_max < np.iinfo(np.int16).max:
                    df[col] = df[col].astype(np.int16)
                elif c_min > np.iinfo(np.int32).min and c_max < np.iinfo(np.int32).max:
                    df[col] = df[col].astype(np.int32)
                elif c_min > np.iinfo(np.int64).min and c_max < np.iinfo(np.int64).max:
                    df[col] = df[col].astype(np.int64)
            else:

                if c_min > np.finfo(np.float16).min and c_max < np.finfo(np.float16).max:
                    df[col] = df[col].astype(np.float32)
                elif c_min > np.finfo(np.float32).min and c_max < np.finfo(np.float32).max:
                    df[col] = df[col].astype(np.float32)
                else:
                    df[col] = df[col].astype(np.float32)
    if verbose:
        logger.info(f"Memory usage of dataframe is {start_mem:.2f} MB")
        end_mem = df.memory_usage().sum() / 1024**2
        logger.info(f"Memory usage after optimization is: {end_mem:.2f} MB")
        decrease = 100 * (start_mem - end_mem) / start_mem
        logger.info(f"Decreased by {decrease:.2f}%")
    return df
```

Code: Feature Engineering

```
def imbalance_features(df):
    # Define lists of price and size-related column names
    prices = ["reference_price", "far_price", "near_price", "ask_price", "bid_price", "wap"]
    sizes = ["matched_size", "bid_size", "ask_size", "imbalance_size"]

    df["volume"] = df.eval("ask_size + bid_size")
    df["mid_price"] = df.eval("(ask_price + bid_price) / 2")
    df["liquidity_imbalance"] = df.eval("(bid_size-ask_size)/(bid_size+ask_size)")
    df["matched_imbalance"] = df.eval("(imbalance_size-matched_size)/(matched_size+imbalance_size)")
    df["size_imbalance"] = df.eval("bid_size / ask_size")

    for c in combinations(prices, 2):
        df[f"{c[0]}_{c[1]}_imb"] = df.eval(f"({c[0]} - {c[1]})/({c[0]} + {c[1]})")

    for c in [['ask_price', 'bid_price', 'wap', 'reference_price'], sizes]:
        triplet_feature = calculate_triplet_imbalance_numba(c, df)
        df[triplet_feature.columns] = triplet_feature.values

    df["stock_weights"] = df["stock_id"].map(weights)
    df["weighted_wap"] = df["stock_weights"] * df["wap"]
    df['wap_momentum'] = df.groupby('stock_id')['weighted_wap'].pct_change(periods=6)

    df["imbalance_momentum"] = df.groupby(['stock_id'])['imbalance_size'].diff(periods=1) / df['matched_size']
    df["price_spread"] = df["ask_price"] - df["bid_price"]
    df["spread_intensity"] = df.groupby(['stock_id'])['price_spread'].diff()
    df['price_pressure'] = df['imbalance_size'] * (df['ask_price'] - df['bid_price'])
    df['market_urgency'] = df['price_spread'] * df['liquidity_imbalance']
    df['depth_pressure'] = (df['ask_size'] - df['bid_size']) * (df['far_price'] - df['near_price'])

    df['spread_depth_ratio'] = (df['ask_price'] - df['bid_price']) / (df['bid_size'] + df['ask_size'])
    df['mid_price_movement'] = df['mid_price'].diff(periods=5).apply(lambda x: 1 if x > 0 else (-1 if x < 0 else 0))

    df['micro_price'] = ((df['bid_price'] * df['ask_size']) + (df['ask_price'] * df['bid_size'])) / (df['bid_size'] + df['ask_size'])
    df['relative_spread'] = (df['ask_price'] - df['bid_price']) / df['wap']
```

```python
# Calculate various statistical aggregation features
for func in ["mean", "std", "skew", "kurt"]:
    df[f"all_prices_{func}"] = df[prices].agg(func, axis=1)
    df[f"all_sizes_{func}"] = df[sizes].agg(func, axis=1)


for col in ['matched_size', 'imbalance_size', 'reference_price', 'imbalance_buy_sell_flag']:
    for window in [1,3,5,10]:
        df[f"{col}_shift_{window}"] = df.groupby('stock_id')[col].shift(window)
        df[f"{col}_ret_{window}"] = df.groupby('stock_id')[col].pct_change(window)

# Calculate diff features for specific columns
for col in ['ask_price', 'bid_price', 'ask_size', 'bid_size', 'weighted_wap','price_spread']:
    for window in [1,3,5,10]:
        df[f"{col}_diff_{window}"] = df.groupby("stock_id")[col].diff(window)

#V4 feature
for window in [3,5,10]:
    df[f'price_change_diff_{window}'] = df[f'bid_price_diff_{window}'] - df[f'ask_price_diff_{window}']
    df[f'size_change_diff_{window}'] = df[f'bid_size_diff_{window}'] - df[f'ask_size_diff_{window}']
```

```python
    #V5 - rolling diff
    # Convert from pandas to Polars
    pl_df = pl.from_pandas(df)

    #Define the windows and columns for which you want to calculate the rolling statistics
    windows = [3, 5, 10]
    columns = ['ask_price', 'bid_price', 'ask_size', 'bid_size']

    # prepare the operations for each column and window
    group = ["stock_id"]
    expressions = []

    # Loop over each window and column to create the rolling mean and std expressions
    for window in windows:
        for col in columns:
            rolling_mean_expr = (
                pl.col(f"{col}_diff_{window}")
                .rolling_mean(window)
                .over(group)
                .alias(f'rolling_diff_{col}_{window}')
            )

            rolling_std_expr = (
                pl.col(f"{col}_diff_{window}")
                .rolling_std(window)
                .over(group)
                .alias(f'rolling_std_diff_{col}_{window}')
            )

            expressions.append(rolling_mean_expr)
            expressions.append(rolling_std_expr)

    # Run the operations using Polars' lazy API
    lazy_df = pl_df.lazy().with_columns(expressions)
```

```python
# Execute the lazy expressions and overwrite the pl_df variable
pl_df = lazy_df.collect()

# Convert back to pandas if necessary
df = pl_df.to_pandas()
gc.collect()

df['mid_price*volume'] = df['mid_price_movement'] * df['volume']
df['harmonic_imbalance'] = df.eval('2 / ((1 / bid_size) + (1 / ask_size))')

for col in df.columns:
    df[col] = df[col].replace([np.inf, -np.inf], 0)

return df
```