Predict US stocks closing movements: Optiver - Trading at the Close

Zhenbang Feng [*1] **Yuxuan Gao** [*1]

Abstract

This paper presents our approach and findings in the Kaggle Competition "Optiver - Trading at the Close", focusing on predicting the closing price movements of Nasdaq-listed stocks. We developed a sophisticated model using an ensemble of LightGBM (LGBM) and CatBoost (CBR), enhanced with extensive feature engineering. Our model effectively forecasts stock closing prices by leveraging data from the order book and closing auction, achieving a Mean Absolute Error (MAE) of 5.3297 on the test set. This performance ranked us 32nd out of 4,015 teams. The paper delves into feature engineering techniques, including stock liquidity and directional movement, temporal dependencies, and the behavior of similar stocks. Our methodology emphasizes the importance of understanding market dynamics and the unpredictability of stock movements, aligning with the principles of Quantitative Research.

1. Introduction

Stock markets are dynamic and high-pressure environments where time is of the essence. The tension grows as the trading day nears its conclusion, reaching its peak during the crucial final ten minutes. During this period, characterized by heightened instability and rapid price shifts, the course of the global economic narrative for the day is significantly influenced. The Nasdaq Stock Exchange wraps up each trading day with the Nasdaq Closing Cross auction, determining the official closing prices for all listed securities. These closing prices are fundamental benchmarks for investors, analysts, and other market participants when assessing the performance of individual securities and the overall market.

The Kaggle Competition "Optiver - Trading at the Close" (1) is an entry point for us to investigate this issue . Our primary goal of this project is to develop a sophisticated model capable of forecasting the closing price movements of hundreds of stocks listed on the Nasdaq Stock Exchange. This endeavor involves leveraging data from both the order book and the closing auction. The insights derived from the auction are particularly valuable, offering potential for applications in price adjustments, the assessment of supply and demand dynamics, and the identification of advantageous trading opportunities.

Our final model, an ensemble of LightGBM (LGBM) and CatBoost (CBR), is the result of extensive research and the use of various data engineering and machine learning techniques, including tree-based and deep transformer models. Its strong performance on the test set with MAE $= 5.3297$ demonstrates its effectiveness in predicting the closing prices of Nasdaq-listed stocks, ranking 32/4015 (Teams) in public leaderboard as of Dec 2023.

2. Related Work

2.1. Nearest Neighbors in Volatility Prediction

A previous "Optiver Realized Volatility Prediction" competition on Kaggle provided participants with the challenge of predicting the realized volatility of stocks (2). One notable solution from this competition utilized a nearest neighbors approach to enhance the prediction accuracy (3). This solution employed time-series cross-validation by reverse engineering the order of the time-id. The data normalization technique was cleverly reversed using tick size to recover the original prices before normalization. This method also utilized a t-SNE dimensionality reduction technique to help recover the order of the time-id with sufficient accuracy. Another pivotal point of this approach was the application of Nearest Neighbor features to aggregate features from nearby time periods, which accounted for most of their score improvement. The modeling section utilized an ensemble of LightGBM, MLP, and a 1D-CNN model, drawing inspiration from a previous competition's architecture.

In our final model, we employed the K-Nearest Neighbors (KNN) algorithm as a pivotal component during the data engineering stage. This algorithm was applied across the entire dataset with the specific aim of categorizing the data into k distinct clusters. The criterion for this categorization was primarily based on the feature-based proximity of the

[*]Equal contribution [1]University of Southern California. Correspondence to: Zhenbang Feng <jasonfen@usc.edu>, Yuxuan Gao <jasongao@usc.edu>.

Proceedings of the 38^{th} International Conference on Machine Learning, PMLR 139, 2021.

data points. Subsequent to this clustering, a comprehensive aggregation procedure was conducted within each of these k clusters. The essence of this procedure involved the calculation of several key statistical metrics for the data points encompassed within each cluster. These metrics notably included the mean, median, and standard deviation, among other relevant statistical measures. These aggregated metrics were then utilized as new features for the data.

2.2. Predicting Stock Prices Using LSTM

Roondiwala *et al.* (4) tackled stock price prediction using recurrent neural networks (RNNs) and Long Short-Term Memory (LSTM) networks. They collected historical stock data of NIFTY 50 from the National stock exchange, specifically from 2011 to 2016. The data sequences were divided into 1180 samples for training and 132 samples for validation. Their methodology relied on RMSprop as the optimizer and involved the normalization of each sequence vector. After various simulations, they found that a feature set containing High, Low, Open, and Close values, combined with 500 epochs of training, gave them the best results. Their work underscores the potential of RNNs, especially LSTMs, in predicting stock prices with reasonable accuracy.

2.3. Deep Transformer Models in Index Prediction

Wang et al. (5) delve into the applications of deep learning for stock market index predictions. They specifically turn their attention towards the Transformer architecture, originally designed for natural language processing tasks, for its potential applicability in the realm of financial forecasting. The Transformer's encoder–decoder structure paired with its multi-head attention mechanism is touted as providing a nuanced grasp of the intricate rules governing stock market movements. Wang et al. conduct back-testing experiments on a selection of the world's principal stock market indices, including the CSI 300, S&P 500, Hang Seng Index, and Nikkei 225. Their studies span a decade, from the start of 2010 to the end of 2020. These experiments are aimed at comparing the performance of the Transformer model against traditional deep learning models and the classic buy & hold strategy. Their findings are indicative of the Transformer's performance both in terms of prediction accuracy and net value analysis.

Prior to finalizing our model, we also experimented with training a deep transformer model. This model demonstrated a reasonable level of performance, achieving a Mean Absolute Error (MAE) of 5.3806 on the test set. Despite multiple attempts at refining the model through various data engineering techniques, we observed no significant improvement in its performance.

3. Dataset and Evaluation

We are using the dataset provided by Optiver for "Optiver - Trading at the Close" Kaggle competition (1). This dataset contains historic data for the daily ten minute closing auction on the NASDAQ stock exchange. The data set contains the following critical features.

3.1. Relevant Features

- 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.
- 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.

$$\frac{\text{bid\_price} * \text{bid\_size} + \text{ask\_price} * \text{ask\_size}}{\text{bid\_size} + \text{ask\_size}}$$

- seconds_in_bucket - The number of seconds elapsed since the beginning of the day's closing auction, always starting from 0.
- target - The 60 second future move in the wap of the stock, less the 60 second future move of the synthetic index. Only provided for the train set.

$$\text{Target} = \left(\frac{\text{StockWAP}_{t+60}}{\text{StockWAP}_t} - \frac{\text{IndexWAP}_{t+60}}{\text{IndexWAP}_t} \right) \times 10000$$

3.2. Evaluation

The model is evaluated on the Mean Absolute Error (MAE) between the predicted return and the observed target. The formula is given by:

$$\text{MAE} = \frac{1}{n}\sum_{i=1}^{\infty}|y_i - x_i|$$

Where:

- n is the total number of data points.
- y_i is the predicted value for the i^{th} data point.
- x_i is the observed value for the i^{th} data point.

3.3. Cross Validation and Testing

The training set comprises over five million rows of historic data for the daily ten minute closing auction on the NASDAQ stock exchange. We employ k-fold (k=5) cross-validation to divide these data points into training and development sets. The mean of the cross-validation scores is taken as the validation score. In this study, the testing data is kept hidden and is retrieved via Optiver API requests. We use the scores from the "Optiver - Trading at the Close" competition leaderboard as a measure for testing performance.

4. Methods

In this section, we outline the methodologies we are adopting for our experiments. We commence by introducing the baseline approach, followed by a presentation of the models we are currently using and the process of our feature engineering.

4.1. Baseline

The baseline model for this competition is a constant predictor, denoted as $prediction(x) = 0, \forall x$. This means it predicts a value of 0 for any input, without any feature engineering. The Mean Absolute Error (MAE) for the baseline model on the test set is 5.4650. One reason the baseline performs fairly well in terms of MAE is that the distribution of the target value, which our model aims to predict, is centered around 0. See Figure 1 for the distribution of the target in the training set.

4.2. Our Models

4.2.1. Model Selection

In this phase of the study, two gradient boosting models were employed and trained on the same training data. In testing, an ensemble of the two models is utilized to improve the accuracy of prediction.

Figure 1. Distribution of Target (in Training Set)

LightGBM (LGBM): light gradient boosting framework based on decision trees. Gradient boosting is a potent ensemble technique that constructs a series of decision trees and optimally combines their outputs to enhance prediction accuracy. Unlike traditional boosting, gradient boosting focuses on the residuals (errors) from preceding trees and fits new trees to these residuals. Consequently, it can quickly correct errors and enhance prediction accuracy. A major factor in choosing this model is the size of the training data. Stock prediction typically demands large-scale historical data recorded at second intervals. LightGBM is highly esteemed for its swift training capabilities and its proficiency in processing large datasets without significant memory consumption, making it a suitable choice for this task.

CatBoost (CBR): another gradient boosting framework which among other features attempts to solve for Categorical features using a permutation driven alternative compared to the classical algorithm. A major factor in choosing this model is that the process of feature engineering generates many additional important features. CatBoost (CBR), with its built-in support for regularization and randomized boosting, helps prevent the model from overfitting, potentially leading to improved performance on test data.

Transformers: Our Transformer model (Figure 2) takes inspiration from the natural language processing domain, where a sentence is understood as a sequence of words or tokens. Similarly, we conceptualize the simultaneous prediction of 200 stock targets at time t as interpreting a sentence comprised of 200 'words'. Each 'word' in this analogy represents an individual stock. In this model, we apply positional encoding to embed stock identities, allowing the model to recognize and learn patterns based on the unique

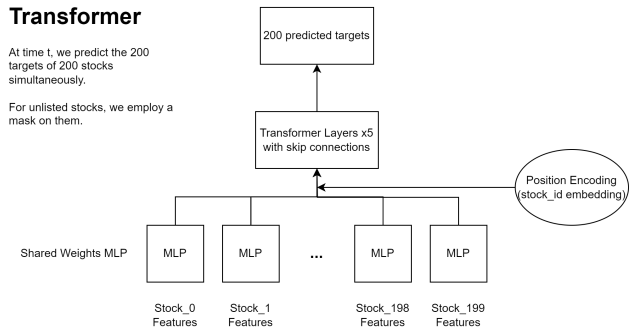

Figure 2. Transformer Architecture

identity of each stock, akin to understanding the role of each word in a sentence.

Despite the theoretical elegance of this analogy, our Transformer model encountered significant challenges in both cross-validation and testing (experiment results shown in Section 5.3). We tried techniques to refine the model, yet its performance did not exceed our LGBM and CBR models. This suggested possible underfitting. The underpinning issue could be rooted in the fundamental difference between language and stocks. While words in a sentence exhibit a high degree of interdependency that Transformer can capture, stocks, although interacting, are influenced by a multitude of external, often unpredictable factors. Unlike words, stocks do not follow a grammatical structure, and their 'meaning' or behavior is not as fixed. The Transformer model was excluded from our final ensemble model after our consideration.

4.2.2. Hyper-parameters

The following important hyper-parameters (not an exhaustive list) are shown for our LGBM and CBR models. These parameters were chosen based on the intuition of making the models deeper while mitigating the risk of overfitting (we did not have enough time and computing resources to do extensive hyper-paramer searching).

LightGBM (LGBM)

- num_leaves: Maximum number of leaves in one tree. It was initially set to 150 but was changed to 180 to create a more complex model.
- max_depth: Maximum depth of the trees. It was initially set to 6 but was changed to -1 (no limit) to create a more complex model.
- n_estimators: Number of boosting iterations. It was initially set to 500 but was changed to 1200 to create a more complex model.
- reg_alpha & reg_lambda: Parameters for L1 regularization and L2 regularization respectively. These regularizations prevent overfitting when training a more complex model.

CatBoost (CBR)

- max_depth: Maximum depth of the trees. It was initially set to 7 but was changed to 10 to create a more complex model.
- min_data_in_leaf: Minimum number of training samples in a leaf. CatBoost does not search for new splits in leaves with samples count less than the specified value.
- l2_leaf_reg: Coefficient at the L2 regularization term of the cost function. It was initially set to 1.5, but was changed to 30 to prevent overfitting when training a more complex model.

4.3. Feature Engineering

4.3.1. Imbalance Features (imb)

The imbalance features capture the difference between buy and sell orders for a given stock. This difference provides insight into the stock's liquidity and directional movement. For instance, if buy orders significantly outnumber sell orders, it may indicate an upward price movement.

Given two prices, a and b, the imbalance feature is calculated as:

$$\text{imb} = \frac{a-b}{a+b}$$

For three prices, a, b, and c, the feature is extended by considering the maximum (max), middle (mid), and minimum (min) values:

$$\text{imb2} = \frac{\text{max} - \text{mid}}{\text{mid} - \text{min}}$$

4.3.2. Stock_id Aggregate Features

These features provide aggregated information for each stock, capturing the general behavior and trend of the stock. By grouping by the stock ID, we can derive aggregate statistics that provide a high-level view of the stock's performance.

Algorithm 1 Stock_id Aggregate Features

Input: data D, columns $cols$
for each column c in $cols$ **do**
  Group D by $stock\_id$ into G
  for each group g in G **do**
    Compute sum(g), mean(g), std(g), median(g)
  end for
end for

4.3.3. Lag Features

Lag features capture the temporal dependencies in the data by using past values to predict future ones.

Raw Lag Features: For a given column, we take values from previous rows (lag) of the same date.

Sub Lag Features: This captures the difference between a column's value and its lagged value.

Lags Means (Rolling avg): This represents the rolling mean of the column values, capturing the average behavior over a window of past values.

Algorithm 2 Lag Features

Input: data D, lag columns lag_cols
for $lag\_step = 1$ **to** 6 **do**
  for each column c in lag_cols **do**
    Create raw lag feature for c by shifting lag_step
    Create sub lag feature = $D[c]$ - lag(c, lag_step)
  end for
  Compute rolling means
end for

4.3.4. Aggregate K Nearest Neighbors Clustering based on Stock_id Aggregate Features

This captures the behavior of stocks that are similar to the target stock, based on the aggregate features. By considering a specified number of nearest neighbors, we can get a sense of how similar stocks are performing.

For desire columns "wap", "imb_s2", "imbalance_size", "wap_std", we compute aggregate functions (mean, std, median) for neighbor_sizes = [5, 10, 20, 40].

Algorithm 3 Aggregate K Nearest Neighbors Features

Input: data D, desired columns $cols$, sizes $sizes$, computed nearest neighbors indices $indices$
Output: data with Nearest Neighbors Features D'
for each column c in $cols$ **do**
  for each size s in $sizes$ **do**
    Select s nearest neighbors of each data point from $indices$ into $Neighbors$
    for each set n in $Neighbors$ **do**
      Compute mean(n) and store in D'
      Compute std(n) and store in D'
      Compute median(n) and store in D'
    end for
  end for
end for
return D'

4.3.5. Daily Rolling Mean

The Daily Rolling Mean is calculated on day level to smooth out the short-term fluctuations and highlight the longer-term trend for each stock. This is achieved by computing the mean of selected features for each $date\_id$ within a stock and then taking the rolling mean over specified window periods.

Algorithm 4 Daily Rolling Mean Calculation

Input: data D, desire columns $cols$, windows W
for each window w in W **do**
  for each feature c in $cols$ **do**
    Compute daily rolling mean for c with w days
    Compute difference between c and its rolling mean
  end for
end for

4.3.6. Stock Technical and More Stock Rolling Features

In addition to the Daily Rolling Mean, we incorporate a suite of technical indicators and more complex rolling features to capture the technical behavior of stocks. These include but are not limited to Moving Average Convergence Divergence (MACD), Bollinger Bands, Historical Volatility, and Relative Strength Index (RSI). Detailed implementations of these features and additional rolling features that consider various aspects of the stocks' technical data are included in Appendix B.

These indicators serve as proxies for various market sentiments and behaviors such as momentum, volatility, and trend strength which are essential in modeling the stock movements.

After generating all the features mentioned above, we obtained approximately 500 features compared with the 13 raw features provided in the dataset (see section 3.1).

5. Experiments

5.1. Feature Experiments

In the quest to develop a better predictive model, we ran a series of experiments to analyze the impact of various feature engineering methods on the model's performance.We built the features iteratively, which means we continually added new features that we thought were useful to the previous versions. The experiments' results are derived from the ensembling of two models: LightGBM (LGBM) and CatBoost (CBR). The ensemble strategy we employ here is taking the average of predictions from both models.

Test scores are taken from the "Optiver - Trading at the

Close" competition leaderboard, determined by a set of hidden test data. CV scores are obtained as the mean of Cross Validation scores in Training.

FEATURES	TEST SCORE	LGBM CV	CBR CV
BASELINE	5.4650	-	-
LGBM + CBR			
IMBALANCES	5.3635	6.2419	6.2507
+STOCKID AGG	5.3626	6.2232	6.2397
+LAG FEATURE	5.3431	6.1263	6.1567
+KNN AGG	5.3426	6.0591	6.0819
+DAILY ROLLING	5.3379	5.9543	5.9762
+TECHNICALS	5.3322	5.9040	5.9069
+STOCK ROLLING	**5.3297**	5.8529	5.8625

Table 1. Feature Experiment Results

5.2. Ensemble Experiment

To assess the efficacy of model ensembling, we also conducted an experiment to compare the results of using LightGBM and CatBoost seperately with Ensemble model (taking the average of predictions from both models).

MODEL	TEST SCORE
ENSEMBLE	**5.3426**
LIGHTGBM	5.3460
CATBOOST	5.3449

Table 2. Ensemble Experiment Results (Project Midpoint)

MODEL	TEST SCORE
ENSEMBLE	**5.3297**
LIGHTGBM	5.3318
CATBOOST	5.3317

Table 3. Ensemble Experiment Results (Final Models)

5.3. Transformer Experiment

The Transformer model utilizes all features of the final versions of LGBM and CBR as shown above (Section 5.1). Table 4 presents the best results obtained from our Transformer model, which performs better than the Baseline (Section 4.1), but worse than LGBM and CBR models. The complete log of experiments is not emphasized here because we did not include this model in our final set of models.

5.4. Insights

From the feature experiments, it is evident that as we added helpful feature engineering, the model's cross-validation and test scores improved. Especially, the addition of Lag features and KNN aggregate shows a considerable positive effect on the model's performance and the later added rolling features also give the model strong improvements. The final model

FEATURES	TEST SCORE	TRANSFORMER CV
BASELINE	5.4650	-
TRANSFORMER		
ALL FEATURES	5.3806	6.3598

Table 4. Transformer Experiment Results

Comparing LightGBM and CatBoost, it's interesting to observe their performance differences. While LightGBM consistently outperforms CatBoost in cross-validation scores, it slightly lags behind in the ensemble test score. This divergence suggests that although LightGBM may have a better average performance during model validation, CatBoost slightly edges out in the ensemble test, hinting at its robustness in testing scenarios. This phenomenon could be attributed to the differences in how each algorithm processes the feature set and manages overfitting, with LightGBM's focus on efficient handling of large datasets and its gradient-based one-side sampling (GOSS) possibly contributing to its cross-validation success. Conversely, CatBoost's unique approach to combating overfitting could explain its marginally better performance in the ensemble experiment.

When ensembled, LightGBM and CatBoost produce a better score than using either individually, providing an improvement of around 0.003 in the test score. This phenomenon illustrates the strength of combining diverse models to merge their strengths.

6. Discussion

6.1. Overall Investigation for Errors

We have taken the last three days of data (date_id = 478, 479, 480) for investigation and obtained the results as shown in the subsequent figures.

As shown in Figure 3, the distribution of the real target is flatter, meaning it encompasses more data points with extreme target values. However, the distribution of the predicted target price is more centralized around 0 and has a smaller range. This suggests that our model's performance is lacking when it comes to predicting data points with large absolute target values.

As depicted in Figure 4, there isn't an obvious relationship between the stock target price and stock_id. However, certain stock_ids exhibit a notably high frequency of large errors (errors greater than 30). This suggests potential undetected volatility among these stocks with frequent significant errors.

Figure 5 illustrates the relationship between the actual target price and the predicted target price. Most data points cluster around the origin, while outliers (data points with high

prediction errors) are evenly distributed in all directions, suggesting that the errors are not biased in any particular direction.

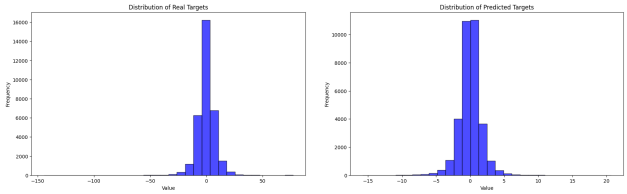

Figure 3. Distribution of real target and predicted target

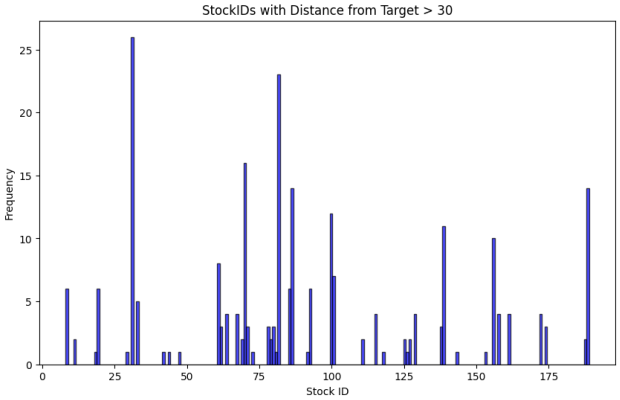

Figure 4. Frequency of stock_ids with large errors

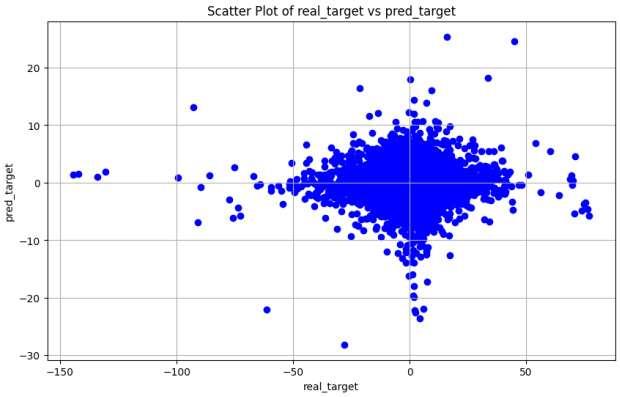

Figure 5. Scatter plot of real target versus predicted target

6.2. Specific Samples Analysis

Taking a closer look at specific data points with largest errors (shown in Appendix A), several patterns emerge which could provide further insights into the model's difficulties.

Repeated Stock_id
It's evident that the same stock_id appears multiple times among the data points with the highest error. Specifically, stock_id = 189 appears on multiple occasions. This phenomenon is also shown in Figure 3.

Imbalance Size
Several samples here have a large imbalance_size as high as 10,330,153. Such values suggest the model might not handle extremities or outliers well.

Magnitude of Real Target
The magnitudes of real_target for these points are notably large, reaching up to 144. This indicates that our model struggles with extreme real target values. One reason might be that these values represent outliers or infrequent occurrences in the training data. Another explanation, as elaborated in Section 6.1, is our model's narrower range for pred_target compared to real_target. This suggests an inadequacy in handling larger targets effectively.

Potential Solutions
To address these issues, we should delve deeper into the stock_ids with large errors and engage in extensive feature engineering to better capture potential volatility. A thorough exploration of their historical data, market influences, and other external factors might provide insight into the observed discrepancies. Instead of training the models on all stock_ids, we may also try to train seperate models to deal with stocks that are harder to predict.

Additionally, for the challenges posed by large real_targets, we can explore techniques like resampling to create a more balanced dataset. Implementing a custom loss function that places greater emphasis on samples with larger targets might also be advantageous. This tailored approach ensures the model is more attuned to and can handle extreme target values effectively.

7. Conclusion

Our final model, an ensemble of LightGBM and Catboost, enhanced with extensive data engineering, achieves a Mean Absolute Error (MAE) of 5.3297 on the test set. This notable performance has enabled us to secure a rank of 32nd out of 4,015 teams, with 10 days left before the competition closes. This ranking stands as a testament to the effectiveness of our methodology.

In this project, we employed a range of well-known machine learning algorithms, including two gradient-boosting models, LightGBM and Catboost, as well as a deep transformer model. Notably, despite the higher complexity and the significantly longer training time required by the deep transformer model compared to the gradient-boosting models, it did not achieve superior performance on the test set.

We also focused on extensive Feature Engineering and transformed 13 raw features into approximately 500 features. From capturing stock liquidity and directional movement with imbalance features to tracing temporal dependencies using lag features, our engineered features enabled our models to grasp intricate patterns within the stock data. The

use of nearest neighbors based on aggregate stock features added another dimension, helping our model capture the trend of similar stocks. Upon conducting a meticulous analysis of the features, we discovered that certain features had a significantly large impact on the predictions, while others were less useful. Consequently, further extracting and refining useful features from those important ones proved to have a considerable effect on the overall predictive performance. Therefore, understanding how to distinguish between useful and less relevant features was crucial in enhancing the predictive accuracy of our model.

This project aligns closely with the principles of Quantitative Research, particularly in its focus on extracting pivotal information that plays a significant role in enhancing our model. It represents a practical application of theoretical knowledge within a real-world context, highlighting the necessity of not only constructing sophisticated models but also comprehensively understanding the market dynamics they are designed to predict. Ultimately, this project not only advanced our ability to in Machine Learning and Quantitative Research, but also underscored the intrinsic unpredictability of certain aspects within stock market dynamics.

References

Tom Forbes, John Macgillivray, Matteo Pietrobon, Sohier Dane, Maggie Demkin. (2023). *Optiver - Trading at the Close.* Kaggle. https://kaggle.com/competitions/optiver-trading-at-the-close

Kaggle. (2021). *Optiver Realized Volatility Prediction*. Retrieved from https://www.kaggle.com/competitions/optiver-realized-volatility-prediction.

Nyanpn. (2022). *1st place (public 2nd place) solution*. Kaggle. Retrieved from https://www.kaggle.com/nyanpn/public-2nd-place-solution.

Roondiwala, M., Patel, H., & Varma, S. (2017). Predicting Stock Prices Using LSTM.

Wang, C., Chen, Y., Zhang, S., Zhang, Q. (2023). Stock market index prediction using deep Transformer model.

A. Appendix: Sample Data Points

Attribute	Data									
index	20631	20831	6031	26789	13082	27789	27589	27189	27389	26989
pred_target	2.234	2.600	1.606	2.743	1.261	-0.636	2.309	-7.157	-2.616	-1.966
real_target	-144.379	-142.210	-133.939	-130.450	-92.780	-99.310	-89.770	-85.720	-90.790	77.130
stock_id	189	189	189	189	31	189	82	189	31	31
imb_size	5380148	6080944	5737450	7590427	59352	10330153	977255	5232961	39046	485995
sec	240	260	250	270	540	280	100	230	300	490
wap	1.001172	1.001168	1.001247	1.001286	0.989185	1.001456	1.002091	1.000803	1.001216	0.998076
date_id	480	480	480	480	480	480	479	480	478	479
volume	39081.22	35713.81	104474.6	52734.52	320077.4	31376.56	220404.8	71759.70	225823.1	217021.7
abs_error	146.614	144.810	135.545	133.193	100.571	89.134	88.029	83.633	79.746	78.416

Table 5. 10 data points in date 478, 479, 480 with highest errors
(not all columns are shown)

sec: seconds_in_bucket, imb_size: imbalance_size

B. Appendix: Link to project files

Link: *https://drive.google.com/drive/folders/1YTs9UOgDtmlhySJn6cCVEl_1d6eOx4nv*
Shorten URL: *http://tinyurl.com/CSCI467OptiverFiles*
Note: please login as USC users to view the files.