Optiver - Trading at the Close Predict US stocks closing movements

Raghu Ram Sattanapalle[1,*] <sattanapalle.r@northeastern.edu>
Eshan Arora[1,*] <arora.es@northeastern.edu>
Pazin Tarasansombat[1,*] <tarasansombat.p@northeastern.edu>
Authors contributed equally to this work.

a) Guidance

This project is not under the guidance of a Northeastern University faculty member.

b) Objectives and Significance

In this project, we are participating in a Kaggle competition with the objective of developing a predictive model for the closing price of NASDAQ-listed stocks. Utilizing data from the order book and the closing auction, we focus on the NASDAQ Closing Cross auction, a crucial event that transpires in the last ten minutes of each trading day. This auction represents nearly 10% of NASDAQ's average daily volume, highlighting its significant role in the financial markets[1]. During this period, NASDAQ determines the official closing price of stocks, aiming to maximize the execution of On-Close orders.

The target of our prediction model is the movement of the Weighted Average Price (WAP) of the stock 60 seconds into the future, within the Closing Cross period. The WAP provides a balanced reflection of the market's supply and demand, calculated as:

$$WAP = \frac{BidPrice \times AskSize + AskPrice \times BidSize}{BidSize + AskSize} \tag{1}$$

where the Bid Price and Ask Price represent the most competitive buy and sell levels, respectively, and the Bid Size and Ask Size denote the dollar notation amounts at these prices.

The target 60-second movement of the WAP is defined as:

$$Target = \left(\frac{StockWap_{t+60}}{StockWap_t} - \frac{IndexWap_{t+60}}{IndexWap_t} \right) \times 10000 \tag{2}$$

where t represents the current time, $t+60$ represents the time 60 seconds into the future, and the synthetic index is a custom-weighted index of NASDAQ-listed stocks created by the competition host, Optiver.

Accurately predicting the closing price is a complex task, influenced by a myriad of factors including market liquidity, trading volumes, and global market movements. The closing auction plays a pivotal role in this, serving as a convergence point for buyers and sellers, and thus is a focal point of liquidity. This underscores the importance of our project's objective to develop a model capable of predicting the WAP movement with high precision.

We evaluate the performance of our model using the Mean Absolute Error (MAE), calculated as:

$$MAE = \frac{1}{n} \sum_{i=1}^{n} |y_i - x_i| \tag{3}$$

where n is the total number of data points, y_i is the predicted value, and x_i is the observed value. A lower MAE indicates higher accuracy of the model.

Success in this project is defined as placing within the top 25% percentile of the Kaggle Competition, corresponding to a MAE significantly lower than that of baseline models. Achieving this does not only demonstrate the efficacy of our model but also contribute to the broader understanding of market dynamics during the closing auction, a period of heightened activity and significance in global financial markets.

Given the global implications and the intricate nature of the closing auction, as highlighted in the BlackRock whitepaper[1], our work in this project aims to provide insights and strategies that can be applied universally, regardless of geographical boundaries. This aligns with the global perspective on market-on-close activity, ensuring that our contributions are relevant and impactful.

To fully grasp the complexity of predicting the closing price and to navigate the challenges posed by the closing auction, the next section provides a detailed background on Market on Close orders, the NASDAQ Closing Cross auction, and the structure of order books. This knowledge is essential the development of our predictive model and for understanding the intricate mechanisms that influence the closing price.

c) Background

What is Market on close (MOC)?

Market on Close (MOC) refers to a type of trade order where an investor wants to buy or sell a given stock at the prevailing market price, with the execution of this order taking place at the close of the stock market trading session. Essentially, it's a directive to transact at whatever the stock's closing price ends up being. This type of order is used when investors are looking to enter or exit a position at the end of the trading day, which is often seen as a significant price point due to the finality of closing prices and their use in performance calculations.

Why is MOC important?

MOC orders are significant for several reasons:

- Price Reference: They provide a clear reference price that is used for accounting and performance evaluation.
- Market Impact: Large MOC orders can influence the closing price of a stock due to the volume of shares being bought or sold.
- Liquidity: MOC orders often contribute to increased liquidity at the end of the trading day.
- Benchmarking: Many funds and investment strategies use the closing price as a benchmark, making MOC orders crucial for aligning with these benchmarks.
- Reduced Slippage: By executing at the closing price, investors can avoid slippage that may occur with other types of orders.

Calculation and Distribution of Market on Close (MOC) Orders

Calculation of MOC

The calculation of the Market on Close (MOC) price is based on a procedure designed to reflect a fair closing price based on actual trades and order flow near the end of the trading day. This process is generally governed by the rules of the specific stock exchange and involves the aggregation of all MOC orders for a particular stock.

In the minutes leading up to the market close, exchanges like the New York Stock Exchange (NYSE) or NASDAQ collect and match MOC orders in a process that aims to balance buy and sell orders at a single price point. This price is determined by the exchange's matching algorithm, which seeks to execute the largest number of shares by

adjusting the price until the buy and sell orders are as close to equal as possible. The final MOC price is then reported as the official closing price for the stock.

Who Receives MOC Information

The information regarding MOC orders is available to market participants, but the detailed order book data might be accessible primarily to the members of the exchange or professional traders who subscribe to a higher level of market data service. This information can include the size and direction of MOC order imbalances, which are often published before the market close, providing insights into potential closing prices.

NASDAQ's Role

NASDAQ, like other exchanges, operates its own closing auction for stocks to determine the closing price. NASDAQ's Closing Cross is a process that determines the NASDAQ Official Closing Price (NOCP). Orders submitted for the close will be executed at the NOCP, provided there is sufficient opposing liquidity. NASDAQ collects MOC orders throughout the trading day and then, a few minutes before the market closes, begins its closing auction process to match buy and sell orders.

The NASDAQ Closing Cross involves several key steps:

- Order Collection: Traders submit their MOC orders throughout the trading session.
- Imbalance Information: NASDAQ disseminates information about order imbalances to the market, indicating whether there is excess buy or sell pressure.
- Auction Run: At a designated time before the closing bell, NASDAQ runs an auction to determine the closing price based on the MOC orders.
- Price Determination: The closing price is set where the maximum volume of shares can trade, ensuring an orderly end-of-day price discovery process.

Auction

Financial markets utilize auctions as a crucial tool for determining asset prices, enabling a direct interaction among numerous buyers and sellers within a controlled and regulated framework. In this competition, we concentrate on the closing auction, a specific type of auction [3].

Closing auctions aggregate orders within a defined timeframe, ultimately executing them at a single price that mirrors the buy and sell intentions of the participants. The NASDAQ Stock Exchange initiates order acceptance from the beginning of the trading day, with the auction book's status being made available from 3:50pm ET. This continues

for a 10-minute period leading up to the market's closure at 4pm ET, when all gathered orders are executed at a unified price. The closing auction plays a significant role in the financial markets, serving as a key mechanism for price discovery, particularly at the end of the trading day [1].

Timeline for NASDAQ Closing Auctions [2]

Key Times	**Key Actions**
Prior to 3:50 p.m. ET	NASDAQ begins accepting Market-On-Close (MOC), Limit-On-Close (LOC), and Imbalance-Only (IO) orders.
3:50 p.m. ET	Early dissemination of closing information begins. NASDAQ continues accepting MOC, LOC and IO orders, but they may not be canceled or modified.
3:55 p.m. ET	Dissemination of closing information begins. NASDAQ stops accepting MOC orders. LOC orders may be entered until 3:58 p.m. ET, but may not be canceled or modified after posting on the order book. IO orders may be entered until 4:00 p.m. ET.
3:58 p.m. ET	NASDAQ stops accepting entry of LOC orders.
4:00 p.m. ET	Closing process begins.

The closing price is determined as the price at which the maximum number of shares can be matched. In the event the auction would have an equal number of matched lots at different levels, NASDAQ uses a proprietary algorithm which takes into account the last traded price, the price-time priority of orders, and the available liquidity at different price levels. It is quite rare that the price at which the maximum shares can be matched is not unique.

Order Book

An order book is a digital record listing buy (bid) and sell (ask) orders for a given financial instrument or security, systematically arranged by price levels [3].

Continuous Trading Order Book Example

Bid	Price	Ask
	10	1
2	9	
0	8	

In continuous trading, the highest bid price will always be less than the lowest ask price. When an ask of 10 shares at a price of 9 is placed, 2 shares would be matched, and the new best ask would be 8 shares at a price of 9.

Updated Continuous Trading Order Book Example

Bid	Price	Ask
	10	1
	9	8
0	8	

Auction Order Book

Bid	Price	Ask
	10	1
3	9	2
4	8	4

The auction order book operates distinctly from its continuous trading counterpart. In this scenario, orders accumulate over a period, awaiting execution until the auction concludes. It is possible for the highest bid and lowest ask prices to overlap, placing the book in a 'crossed' state, as depicted in the given example [3]. The 'uncross' price is the price at which these overlapping orders are executed.

For the auction book example above:

- At a price of 10, 0 lots would be matched since there as no bids ≥ 10.
- At a price of 9, 3 lots would be matched, as there are 3 bids ≥ 9 and 6 asks ≤ 9.
- At a price of 8, 4 lots would be matched, since there are 7 bids ≥ 8, and there are 4 asks ≤ 8.

We would therefore describe the auction order book in the following way:

- The uncross price is 8.
- The matched size would be 4.
- The imbalance would be 3 lots in the buy direction.

The term imbalance refers to the number of unmatched shares. The term far price refers to the hypothetical uncross price of the auction book if it were to uncross at the reporting time.

Combined Order Book

Bid	Price	Ask
5	10	2
5	9	2
4	8	4

Merging the order books from both continuous trading and auction scenarios provides a fuller picture of the market's buying and selling interests across various price levels [3]. For the combined book:

- The uncross price is 9.
- The matched size is 5.
- The imbalance would be 1 lot, in the sell direction.

The hypothetical uncross price of the combined book is called the *near price*. NASDAQ provides near price information 5 minutes before the closing cross.

Reference Price:

The NASDAQ provides a 'reference price', serving as an indicator of the fair market price [3]. This price is determined based on the following conditions:

- If the near price is between the best bid and ask, then the reference price is equal to the near price.
- If the near price $>$ best ask, then the reference price = best ask.
- If the near price $<$ best bid, then the reference price = best bid.

So the reference price is the near price bounded between the best bid and ask.

Understanding the dynamics of Market on Close (MOC) orders, the NASDAQ Closing Cross auction, and order book structures is pivotal for our project. The predictive model we aim to develop hinges on accurately interpreting the order book data and the auction process, as these elements directly influence the closing price of NASDAQ-listed stocks. By delving into the intricacies of these financial mechanisms, we equip ourselves with the knowledge required to make informed predictions, ultimately striving to enhance market efficiency and contribute to more stable and predictable closing auctions.

The detailed exploration of MOC orders and the NASDAQ Closing Cross auction in this section lays the groundwork for our project, directly linking to our objectives of developing a high-precision predictive model for the closing price. The understanding of order book structures and the auction process is crucial, as it influences our approach to modeling and impacts the accuracy of our predictions. This background knowledge ensures that we are well-equipped to tackle the challenges of predicting the closing price, bringing us a step closer to achieving our goal of placing within the top 25% percentile in the Kaggle competition and contributing to the broader understanding of market dynamics during the closing auction.

Literature Review

Statistical Tools Used in Stock Market Analysis

In-depth analysis of selected studies reveals a wide array of statistical tools utilized for forecasting in the stock market. These tools encompass not only basic descriptive statistics for initial data interpretation but also incorporate advanced predictive models such as ARIMA, Regression, and Clustering.

- ARIMA: ARIMA, which stands for Autoregressive Integrated Moving Average, is a class of statistical models used for analyzing and forecasting time series data. It is a tool that captures various forms of data patterns to help forecast future points in the series. ARIMA models are widely used in finance, economics, and other fields for their robustness in time series prediction. It is praised for its utility in predicting stock market trends and understanding complex data sequences, providing a structured approach to capturing market dynamics[5].

- Clustering: Clustering methods efficiently categorize stocks into homogenous groups, simplifying the asset selection process for portfolio management and aiding in risk diversification strategies[6].

Machine Learning Algorithms for Stock Market Prediction

The inclination towards machine learning (ML) and deep learning algorithms is evident in stock market prediction research. These advanced techniques are favored for their ability to handle complex, nonlinear patterns that traditional statistical methods may not effectively address.

- SVM (Support Vector Machine): Support Vector Machines (SVM) have established their efficacy in financial markets, where they're used to discern patterns and predict future values based on historical data, thus aiding in both risk assessment and opportunity identification[7].

- NN (Neural Networks): Neural Networks (NN) are designed to replicate the neural structure of the human brain, making them exceptionally good at recognizing patterns in data, a capability that is instrumental in predicting stock market fluctuations[8].
- ANN (Artificial Neural Networks): Artificial Neural Networks (ANNs) go beyond simple prediction; they are capable of identifying complex nonlinear relationships within vast datasets, often outperforming other forecasting methods in accuracy[9].
- CNN (Convolutional Neural Networks): Convolutional Neural Networks (CNNs) have revolutionized the field of pattern recognition within the stock market, providing deep insights into data through their intricate network structure[10].
- RNN (Recurrent Neural Networks): Recurrent Neural Networks (RNNs) excel in analyzing time-series data, which is critical in understanding and predicting the sequential nature of stock prices[11].
- SVR (Support Vector Regression): Support Vector Regression (SVR) extends the capabilities of SVMs to regression problems, offering a robust approach to predict continuous outcomes such as stock prices with precision[12].
- GAN (Generative Adversarial Networks): Generative Adversarial Networks (GANs) have emerged as a powerful tool for generating synthetic data that is remarkably similar to real-world data, useful for simulating market scenarios and testing trading
- Naïve Bayes (NB): The Naïve Bayes classifier, with its foundation in probability, provides a simple yet effective approach to classifying financial data, making it an invaluable tool for initial market analysis[14].

d) Methods

(a) Data

Dataset Overview:

For our class project, we have chosen to participate in a Kaggle competition. This competition revolves around predicting the closing price movements of hundreds of stocks listed on the NASDAQ Stock Exchange, using data from both the order book and the closing auction of each stock. The challenge is not just theoretical; it mirrors the real-world scenarios that traders, quantitative researchers, and engineers at companies like Optiver face on a daily basis.

Data Acquisition:

The dataset for this project is sourced from the NASDAQ Closing Cross auction, a daily event that determines the official closing prices for securities listed on the exchange. This data is crucial as it accounts for almost 10% of NASDAQ's average daily volume and plays a significant role in setting benchmark prices for various investment strategies.

Dataset Description:

The dataset comprises several key components, each providing unique insights [4]:

- `[train].csv` - Auction data, including:
  - *stock_id*: Unique identifier for each stock.
  - *date_id*: Unique identifier for the date, consistent across all stocks.
  - *imbalance_size*: Amount unmatched at the current reference price (in USD).
  - *imbalance_buy_sell_flag*: Indicator of auction imbalance direction.
  - *reference_price*: Price optimizing various criteria.
  - *matched_size*: Amount matched at the current reference price (in USD).
  - *far_price* and *near_price*: Prices maximizing the number of shares matched based on different criteria.
  - *bid/ask_price*: Price of the most competitive buy/sell level in the non-auction book.
  - *bid/ask_size*: Dollar amount on the most competitive buy/sell level in the non-auction book.
  - *wap*: Weighted average price in the non-auction book.
  - *seconds_in_bucket*: Seconds elapsed since the beginning of the day's closing auction.
  - *target*: Main predictive objective.

We evaluate our model based on the Mean Absolute Error (MAE) between the predicted and observed stock price movements.

Acquisition Methodology:

The data for this competition is directly acquired from Kaggle [4], ensuring a reliable and consistent dataset. Given the time-sensitive nature of the competition and its focus on predicting future events, we adhere strictly to the provided time series API for submissions. This approach guarantees that our model does not inadvertently use future data in its predictions, maintaining the integrity of our results.

(b) Method and Implementation

Exploratory Data Analysis

Before delving into preprocessing and modeling, we conducted an extensive exploratory data analysis (EDA) to acquire a profound understanding of our dataset.

Utilizing custom functions, we delved into each feature of our dataset, scrutinizing aspects such as uniqueness, cardinality, null values, data types, and sample values. Our dataset encompasses 5,237,980 rows and 17 columns, capturing the final 10 minutes of trading for 200 stocks across 481 trading days, with each minute subdivided into smaller time buckets.

We identified that some stocks are devoid of data on specific days, totaling 964 instances of missing data across 11 unique stocks. Nevertheless, all existing time series maintain a consistent length of 55 steps, facilitating our preprocessing and modeling endeavors.

In the course of our missing values analysis, we calculated the number of missing values per feature, summarized in Table 1.

Table 1: Summary of Missing Values in the Dataset

Feature	**Number of Missing Values**
stock_id	0
date_id	0
seconds_in_bucket	0
imbalance_size	220
imbalance_buy_sell_flag	0
reference_price	220
matched_size	220
far_price	2,894,342
near_price	2,857,180
bid_price	220
bid_size	0
ask_price	220
ask_size	0
wap	220
target	88
time_id	0
row_id	0

Upon inspection, we discerned that the extensive number of missing values in far_price and near_price is attributable to these features being exclusively available during the last 5 minutes of the trading day, coinciding with the NASDAQ Closing Cross auction. This insight is pivotal as it contextualizes the missing values and guides our strategy for addressing them.

This thorough analysis ensures our cognizance of potential challenges and unique characteristics of our dataset, empowering us to make enlightened decisions in our subsequent modeling endeavors.

Data Preprocessing

Ensuring data quality and consistency is crucial for the success of any machine learning project. Our preprocessing steps comprise:

1. **Handling Missing Values:** We conducted a meticulous examination of missing values in our dataset. The significance of comprehending the financial context and maintaining temporal consistency when addressing missing data is paramount. For features with a smaller number of missing values (such as 'imbalance_size', 'reference_price', 'matched_size', 'bid_price', 'ask_price', and 'wap'), we will assess the impact of these missing values and consider imputation methods, while considering the time-series nature of our data. For 'far_price' and 'near_price', given their predictable absence in the first 5 minutes of each trading day, we can address the missing values by creating a binary feature indicating the availability of the price or by imputing the values based on available data, depending on the relevance of these features to our predictive models. The 'target' feature, being our dependent variable, has 88 missing values. Rows with missing target values is excluded from our training data, as they are not utilizable for training our supervised learning models. This intricate analysis of missing values ensures that our data preprocessing steps are informed and tailored to the specific characteristics of our dataset, leading to more reliable and accurate predictive models.

2. **Memory Optimization:** We implemented memory optimization techniques, notably through a custom function `reduce_mem_usage`, which systematically reduced the memory footprint of our DataFrame. By downcasting numerical columns to more memory-efficient types, we significantly enhanced computation performance, a critical step for managing large datasets and ensuring efficiency during model training and evaluation.

Feature Engineering

In our approach to develop a predictive model for NASDAQ Closing Cross, we have undertaken extensive feature engineering to extract meaningful insights from the data. Our feature engineering strategy focuses on capturing the multifaceted nature of financial time series data and includes the following key aspects:

- **Lagged Features:** Understanding stock market trends requires insight into its past. Here, we implemented lagged features for key variables like `imbalance_size`,

reference_price, matched_size, bid_price, and ask_price. These features provide a glimpse into the past, enabling our model to discern patterns over time. For instance, the imbalance_size lagged by one minute may offer crucial insights into the immediate market response.

- **Rolling Window Features:** The stock market is a blend of volatility and stability. To capture this, we compute the rolling mean and standard deviation for various window sizes (3, 5, 10 minutes). This approach helps us grasp both the rapid fluctuations and the more gradual trends in the market.

- **Interaction and Ratio Features:** The dynamics between different market variables are complex. We create features representing interactions and ratios, like the imbalance between buying and selling or the ratio of bid-to-ask prices. These features are pivotal in decoding the supply-demand balance and liquidity in the market.

- **Trigonometric and Time-Based Features:** Market activities exhibit cyclical patterns within the trading day. Applying trigonometric transformations to the seconds_in_bucket feature allows us to model these cycles and understand the urgency of trading activity as the auction time approaches.

- **Advanced Custom Features:** Based on the data, we compute a range of sophisticated features like imbalance_momentum, price_spread, spread_intensity, price_pressure, market_urgency, and depth_pressure. These features are designed to capture nuanced aspects of market dynamics such as the momentum of buying/selling interest, the intensity of price spreads, and the pressure exerted by market depth on price movements.

- **Statistical Aggregation Features:** To get a holistic view of the market, we calculate various aggregated statistics like mean, standard deviation, skewness, and kurtosis for different sets of features (e.g., prices and sizes). This gives a comprehensive view of the market conditions and helps in identifying underlying patterns.

- **Lagged Return Features:** For features such as ask_price, bid_price, ask_size, bid_size, we compute lagged returns over different time windows. This helps in understanding how past price and size movements influence future market behavior.

- **Parallel Computing for Efficient Calculation:** The computational intensity of generating features like triplet imbalance calculation is managed efficiently through parallel computing, utilizing Numba's JIT compilation. This significantly speeds up our feature generation process, a critical factor when dealing with large datasets.

- **Rolling and Moving Averages:** We employ rolling averages for different price features to smooth out short-term fluctuations, thereby uncovering underlying market trends. This is particularly useful in identifying persistent patterns amidst the market's inherent noise.

- **Data Transformation and Integration:** Our dataset is enriched by integrating additional features such as day of the week (`dow`), finer time granularity from `seconds_in_bucket`, and global stock information derived from aggregated market statistics. These features not only add depth to our model but also provide a more comprehensive perspective of the market conditions and stock behavior.

Through this extensive feature engineering process, we have transformed our data into rich insights, laying a solid foundation for our predictive modeling. Each feature brings us closer to understanding the complex and dynamic nature of the stock market, particularly during the critical period of the NASDAQ Closing Cross.

Implementation and Results of Linear Regression Analysis for the Synthetic Index

Our implementation of linear regression to determine the influence of individual stocks on the synthetic index involved a series of carefully planned steps, executed using Python's data analysis libraries. We began by calculating stock and index returns from our dataset, ensuring that all data anomalies were addressed to maintain data integrity.

Once the dataset was prepared, we employed the Linear Regression model from the `sklearn` library. This model was chosen for its efficiency and effectiveness in handling large datasets like ours. We fitted the model with stock returns as independent variables and the index return as the dependent variable. This setup was critical to understand the weightage of each stock in the synthetic index.

The results of our linear regression analysis were quite revealing:

- **Coefficients:** The coefficients obtained from the model, representing the weights of individual stocks in the synthetic index, varied significantly across stocks. These coefficients are key inputs in our feature engineering process, particularly in the `imbalance_features` function, where they are used to weigh the impact of each stock's movement.

- **R-squared Value:** The R-squared value from the model was significantly high, indicating a strong correlation between our model's predictions and the actual index returns. This high R-squared value reaffirms the reliability of our model in accurately determining the weights of individual stocks.

Through this analysis, we successfully quantified the contribution of each stock to the movement of the synthetic index. These quantified weights are a crucial component of our feature engineering process, allowing us to create more nuanced and informed features that reflect the complexities of the stock market.

In summary, our linear regression analysis not only provided us with valuable insights into market dynamics but also equipped us with a data-driven method to enhance our predictive model's accuracy and robustness.

Application to feature engineering: We map the calculated weights to each stock using the `stock_id` as a key. This mapping is critical in creating the `weighted_wap` feature, where each stock's Weighted Average Price (WAP) is multiplied by its respective weight, resulting in a feature that reflects the weighted influence of each stock's price movement.
The utilization of these weights in our feature engineering process allows us to create features that are more aligned with real-time market dynamics. It enables the model to account for the varying degrees of influence that different stocks have on the overall market, making our predictions more robust and reflective of actual market conditions.

Model Selection

We employed a variety of machine learning models, each with its own strengths, to capture different patterns in the data:

1. **LightGBM:**

   One of the primary models we have chosen for our stock market prediction task is the Light Gradient Boosting Machine (LightGBM) [16, 17]. LightGBM is a fast, distributed, high-performance gradient boosting (GBDT, GBRT, GBM) framework, based on decision tree algorithms. It is designed for efficiency, scalability, and higher performance, especially on large volumes of data.

   Key Features of LightGBM [15, 17]:

   - *Efficiency in Large Datasets:* LightGBM is optimized for performance and can handle large datasets with ease. This makes it particularly suitable for our dataset, which contains detailed information on stock prices and various market indicators.
   - *Handling of Sparse Data:* Financial time series data often contains many sparse features. LightGBM is efficient in handling such sparse data, ensuring that even subtle patterns in less dense features are not overlooked.

- *Faster Training:* Compared to other gradient boosting frameworks, LightGBM accelerates the training process without compromising accuracy, making it an ideal choice for iterative modeling and rapid experimentation.
- *Support for Categorical Features:* LightGBM natively supports categorical features, which means there's no need for extensive preprocessing to convert categorical data into numerical formats.
- *Leaf-wise Growth Strategy:* Unlike other boosting algorithms that grow trees level-wise, LightGBM grows trees leaf-wise, which can result in reduced loss and improved accuracy on complex datasets.

When implementing XGBoost for stock market prediction, our focus is not only on key hyperparameters but also on model training strategies and feature engineering. This ensures a model that is both accurate and robust against overfitting.

Implementation Strategy: For implementing LightGBM, we have carefully configured various hyperparameters to optimize its performance for our specific use case. Key parameters include the number of leaves in the decision tree, the learning rate, and the maximum depth of the trees. Regularization parameters are also fine-tuned to prevent overfitting, ensuring that the model generalizes well to unseen data.

2. **XGBoost - Extreme Gradient Boosting:**

Our second model was based on XGBoost regressor. XGBoost, short for Extreme Gradient Boosting, is a highly efficient and flexible gradient boosting library [18]. It's a decision-tree-based ensemble Machine Learning algorithm that uses a gradient boosting framework. Initially developed at the University of Washington, XGBoost has gained immense popularity in data science competitions for its performance and speed [19].

Key Features of XGBoost:

- *Efficiency and Scalability:* XGBoost is designed to be highly efficient, scalable, and portable. It effectively handles large-scale data and has been engineered to exploit every bit of available computational resources.
- *Regularization:* XGBoost includes L1 (Lasso Regression) and L2 (Ridge Regression) regularization, which prevents overfitting and improves model performance.

- *Handling of Missing Values:* XGBoost has an in-built routine to handle missing data. This is particularly useful in messy datasets common in real-world scenarios.
- *Built-in Cross-Validation:* XGBoost allows users to run a cross-validation at each iteration of the boosting process and is thus easy to get the exact optimum number of boosting rounds for a given training dataset.
- *Customizable Optimization Objectives and Evaluation Criteria:* XGBoost allows users to define custom optimization objectives and evaluation criteria, adding a level of flexibility for various data science problems.
- *Tree Pruning:* Unlike GBM where tree pruning stops once a negative loss is encountered, XGBoost grows the tree to its maximum depth and then prunes it back to improve efficiency.

When implementing XGBoost for stock market prediction, our focus is not only on key hyperparameters but also on model training strategies and feature engineering. This ensures a model that is both accurate and robust against overfitting.

Regularization in XGBoost:

XGBoost includes L1 (Lasso Regression) and L2 (Ridge Regression) regularization in its objective function. The regularization terms are added to control overfitting. The objective function of XGBoost with regularization is represented as:

$$\text{Obj} = \sum_{i=1}^{n} l(y_i, \hat{y}_i) + \lambda \sum_{k} w_k^2 + \alpha \sum_{k} |w_k| \tag{4}$$

Where:

- $l(y_i, \hat{y}_i)$ is the loss function evaluated on the predicted value y; and the true value y_i.
- λ and α are the L2 and L1 regularization terms, respectively.
- w_k represents the weights of the model.

Gradient and Hessian in XGBoost (Second-order Approximation):

XGBoost uses a second-order approximation for optimizing the objective function. The gradients (first-order derivative) and Hessians (second-order derivative) are computed for each instance. The generic update rule in XGBoost can be represented as:

$$\hat{y}_i^{(t+1)} = \hat{y}_i^{(t)} + \eta \cdot \sum_{k=1}^{K} f_k(x_i) \tag{5}$$

$$\text{Objective} = \sum_{i=1}^{n} l(y_i, \hat{y}_i) + \lambda \sum_{k} w_k^2 + \alpha \sum_{k} |w_k| \tag{6}$$

Where:

- $\hat{y}_i^{(t)}$ is the prediction at iteration t.
- η is the learning rate.
- g_i and h_i are the first and second-order gradients of the loss function, respectively.
- $f_k(x_i)$ represents the output of the k-th tree for the i-th instance.

Functional gradient Boosting

- *Update Rule:* The model is updated by adding a new decision tree at each iteration. The function is chosen to minimize the loss when added to the existing model. The update rule can be written as:

$$F_{t+1}(x) = F_t(x) + \rho_t \cdot h_t(x) \tag{7}$$

  Where:

  - $F_t(x)$ is the model at iteration t.
  - $h_t(x)$ is the function (tree) added at iteration t.
  - ρ_t is the learning rate at iteration t.

- *Gradient and Hessian Calculation:* At each iteration, the gradient and Hessian (second-order derivative) of the loss (residual) function with respect to the predictions are calculated for each instance. These are used to determine the best direction to update the model.

$$g_i = \frac{\partial \text{Loss}(y_i, F(x_i))}{\partial F(x_i)} \tag{8}$$

$$h_i = \frac{\partial^2 \text{Loss}(y_i, F(x_i))}{\partial F(x_i)^2} \tag{9}$$

  Where:

  - g_i is the gradient of the loss function with respect to the prediction for the i-th instance.

- h_i is the Hessian (second-order derivative) of the loss function for the i-th instance.
- y_i is the true value for the i-th instance.
- $F(x_i)$ is the current prediction of the model for the i-th instance.

- Optimization Objective: The objective function for finding the best function $h_t(x)$ at each iteration involves minimizing the following expression:

$$\text{Objective}(t) = \sum_{i=1}^{n} \left[g_i \cdot h_t(x_i) + \frac{1}{2} h_i \cdot h_t^2(x_i) \right] + \text{Regularization}(h_t) \tag{10}$$

  Where:

  - The first term in the summation is the linear approximation of the loss function's change.
  - The second term provides a quadratic approximation, making the optimization more accurate.
  - Regularization is applied to the function h_t to control its complexity.

Functional Gradient Boosting is an iterative algorithm where each step involves building a tree that best reduces the residual loss from the previous step. By using both the first and second-order derivatives (gradient and Hessian), the algorithm can make more informed updates, leading to faster convergence and potentially more accurate models.

XGBoost vs LightGBM

We are employing two tree best methods, to understand the reader better we will distinguish their workings in this section

LightGBM - Leaf-wise (Best-first) Tree Growth: LightGBM grows trees leaf-wise, choosing the leaf that will yield the largest decrease in the loss function. This strategy can be described as follows:

- **Leaf Selection for Splitting:** For each leaf, calculate the reduction in the loss function that would result from making a split at that leaf. This is given by:

$$\text{Gain} = \text{Loss}_{\text{before split}} - \text{Loss}_{\text{after split}}$$

- **Selecting the Best Leaf:** Among all leaves, the one with the highest gain is chosen for splitting. Mathematically, this is represented as:

$$\text{Best Leaf} = \arg\max_{\text{leaf}} \text{Gain}_{\text{leaf}}$$

XGBoost - Level-wise (Depth-first) Tree Growth: XGBoost grows trees level-wise, adding a new level to all leaves of the tree before moving deeper, which can be more balanced and less prone to overfitting. The steps include:

- **Level-wise Expansion:** At each level, all leaves are considered for splitting. The potential gain for each split is calculated similarly to LightGBM.
- **Regularization:** XGBoost incorporates a regularization term in its objective function, which is expressed as:

$$\Omega(f) = \gamma T + \frac{1}{2}\lambda \sum_{j=1}^{T} w_j^2$$

where $\Omega(f)$ is the regularization term, γ is the complexity control parameter, T is the number of leaves, w_j is the score on the j-th leaf, and λ is the regularization coefficient.

Comparison:

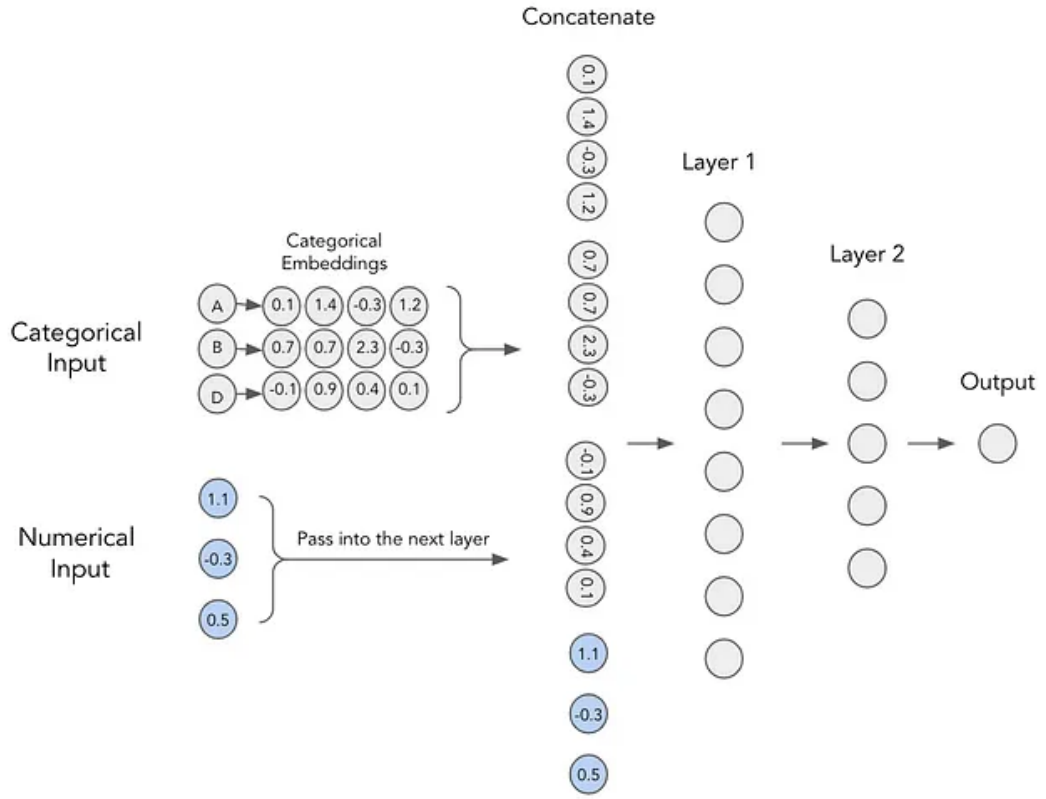
- LightGBM's leaf-wise approach can result in deeper, more complex trees, which may yield better performance on large datasets with complex patterns but can also lead to overfitting on smaller datasets.
- XGBoost's level-wise approach ensures a more balanced tree structure, reducing the risk of overfitting, which is beneficial for smaller or less complex datasets.

3. **Neural Networks:**

One of the model we selected to implement for this project is a Neural Network model. We shall compare the performance of the Neural Network with that of the LightGBM and XGBoost model. The nature of the data for this project is such that the data is tabular with mixture of both categorical data and numerical data. To enhance the performance of Neural Network for such data structure, we will make use of Transformer to transform categorical embedding into contextual embeddings, allowing the model to learn the relationship between categorical input. Two models provides the functionalities to use Transformer on categorical data: TabTransformer Model and FT-Transformer Model.

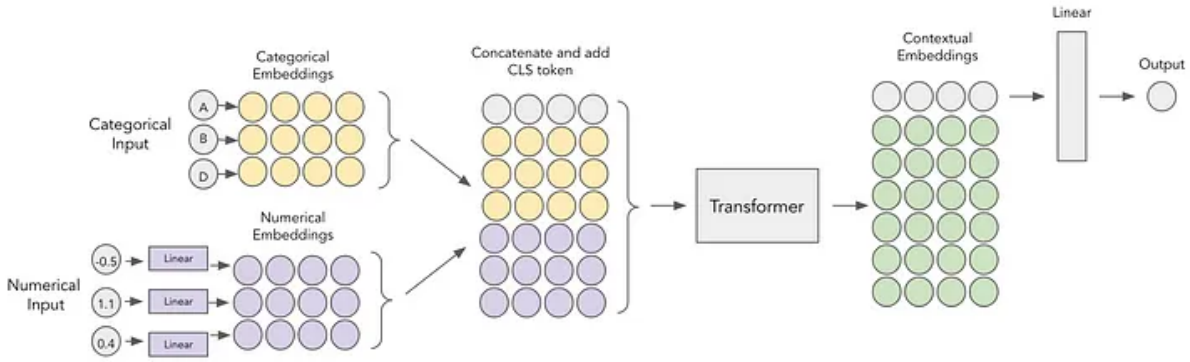
TabTransformer works by feeding the categorical embeddings of categorical input into a Transformer, transforming it into contextual embeddings. The model then feeds the contextual embedding alongside the numerical input into the traditional

MLP neural network to learn the pattern.



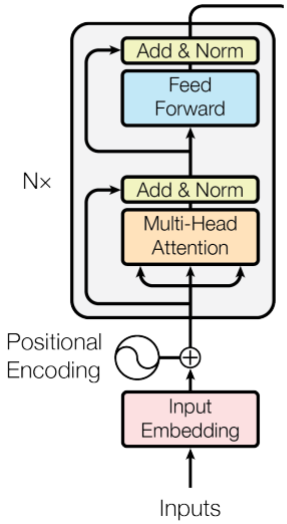
(Picture from: https://towardsdatascience.com/transformers-for-tabular-data-tabtransformer-deep-dive-5fb2438da820)

FT-Transformer, which we use for the project, is an improvement over TabTransformer model. FT-Transformer differs from TabTransformer in that rather than only passing the categorical embeddings into the Transformer, FT-Transformer also pass along the numerical data by transforming them into linear embeddings, allowing the model to learn the context from both the categorical and numerical input. We can then pass contextual embeddings into a traditional MLP neural network.



(Picture from: https://medium.com/p/dbc3be3b5bb5)

The key component of both TabTransformer and the FT-Transformer Model is the Transformer itself, which allows the model to contextualize and learn the relationship between inputs. Transformer is a popular tool in the NLP domain to contextualize embeddings, and the FT-Transformer model applies it to tabular data to learn relationship between categorial and numerical inputs. The Transformer block itself is an Encoder with a Feed-Forward MLP trained to learn relationship between inputs and contextualize using Multi-Headed Attention.



(Picture from: https://arxiv.org/abs/1706.03762)

Overall, after testing various parameters, Transformer Neural Network fails to match up to LightGBM in performance for our specific scenario. The model struggles to learn with with the computational resources available and as we explore more feature-sets, the runtime and resource usage to train the neural network model becomes prohibitive.

(c) Evaluation Strategy

Overview

Our evaluation strategy is meticulously designed to ensure a comprehensive assessment of our predictive models, aligning with the competition's guidelines and providing us with reliable performance metrics to guide our model development and selection process.

Primary Metric: Mean Absolute Error (MAE)

The primary metric for evaluating our models is the Mean Absolute Error (MAE), as dictated by the Kaggle competition rules. The MAE is calculated as follows:

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

where:

- n is the total number of data points in the dataset.
- y_i represents the predicted value for the i^{th} data point.
- x_i denotes the observed (actual) value for the i^{th} data point.

This formula calculates the average of the absolute differences between predicted and actual values across all data points, providing a straightforward and intuitive measure of model performance. The absolute value ensures that all errors, whether they are overestimates or underestimates, are treated equally, thus offering a balanced view of the model's prediction accuracy.

The simplicity and clarity of MAE make it an excellent choice for our evaluation purposes. It allows us to directly interpret the results as the average prediction error in the same units as the predicted variable, thus making it highly intuitive and practical for assessing the effectiveness of our models in the Kaggle competition.

By focusing on MAE, we aim to minimize these average errors, thereby enhancing the reliability and accuracy of our predictions. This metric's alignment with the competition's evaluation criteria also ensures that our model development is directly targeted towards the competition's objectives.

Time Series Cross-Validation

To evaluate our models, we implement a robust validation strategy tailored for time series data, known as Time Series Cross-Validation. This method is crucial for our project since our data is inherently sequential, with each data point representing a specific moment in time.

In Time Series Cross-Validation, the data is split based on time. Unlike traditional cross-validation methods that randomly partition data, this approach respects the temporal order of observations. It ensures that the model is only validated on data that occurs after the training data, thereby avoiding the leakage of information from the future into the training process. This temporal integrity is vital for our model's ability to make accurate predictions on unseen future data.

Our implementation of Time Series Cross-Validation involves dividing the dataset into a number of folds. Each fold serves as a unique validation set, while the preceding data

forms the training set. Specifically, we adopt a 'purged' approach to avoid look-ahead bias:

- We define a total of five folds, with each fold covering a specific range of days within our 480-day dataset.
- For each fold, we determine a 'purged' period—a small gap between the training and validation sets. This gap is crucial to ensure that information from the validation set does not inadvertently 'leak' into the training set, which could happen due to market impacts or other temporal effects.
- The model is trained on data before the purged period and validated on data after this period. This separation simulates a more realistic scenario where the model makes predictions based on past information without any knowledge of future events.
- This process is repeated for each fold, allowing the model to be validated on different segments of the dataset, thereby ensuring a comprehensive assessment of its performance over time.

By employing Time Series Cross-Validation, we adhere to the sequential nature of our financial data, ensuring that our model is rigorously evaluated under realistic conditions. This method not only enhances the validity of our model's performance metrics but also strengthens its capability to generalize to new, unseen data.

Early Stopping: To further enhance the XGBoost model performance and avoid overfitting, early stopping is implemented. The training process stops if the validation score does not improve for a specified number of rounds (100). This approach ensures that the model doesn't over-learn from the training data.

Feature Engineering and Selection

Feature engineering and selection are pivotal aspects of our project, significantly influencing the performance of our LightGBM model. LightGBM inherently handles features efficiently, making it a suitable choice for datasets with a large number of features and complex relationships.

In our approach, we initially conduct extensive feature engineering to transform our raw data into a format that can be effectively utilized by the LightGBM model. This process includes creating new features that encapsulate important aspects of the financial data, such as price movements, volume changes, and time-based indicators. We meticulously

design these features to capture the nuances of stock market dynamics, particularly during the closing auction period.

Once the features are engineered, we proceed with the selection process. LightGBM has a built-in mechanism for handling feature selection through its gradient boosting framework. This model automatically identifies and gives more weight to features that have a more significant impact on the prediction task. This inherent feature handling capability of LightGBM is beneficial for our model's ability to focus on the most relevant information in our dataset.

To further refine our feature selection process, we employ a post-modeling technique known as feature importance analysis. After training our LightGBM models on the time series cross-validated data, we assess the importance of each feature. This assessment is based on the gain importance, which quantifies the contribution of each feature to the model's performance.

- We load the models saved during the cross-validation process, ensuring that we consider the feature importance across all folds.
- For each model, we extract the feature importance scores and aggregate them to get an average importance score for each feature.
- We then visualize the top features using a bar plot, which provides us with a clear understanding of which features are most influential in predicting the closing price movements.
- This visualization not only validates our feature engineering efforts but also guides us in refining our feature set. We can identify and potentially remove features that have little to no impact on the model's performance, thereby simplifying the model and reducing the risk of overfitting.

The combination of LightGBM's inherent feature handling and our post-modeling feature importance analysis forms a comprehensive approach to feature engineering and selection. This method ensures that our model is trained on the most relevant and impactful features, leading to more accurate and reliable predictions.

Finally, the results of the feature importance analysis are saved into a CSV file for documentation and further analysis. This file serves as a record of the most influential features in our model and can be used for future reference.

Grid Search for Hyperparameter Optimization

In our project, the hyperparameters of the LightGBM model play a crucial role in determining its performance and efficiency. To find the optimal set of hyperparameters, we employed a grid search methodology, a systematic approach to parameter tuning that searches through a specified subset of hyperparameters to find the combination that yields the best model performance.

The grid search process involves the following steps:

- **Parameter Grid Definition:** We first defined a grid of hyperparameters to explore. This grid included a range of values for key parameters like the number of leaves in the decision tree (num_leaves), the learning rate, the number of estimators (n_estimators), and others like subsample and colsample_bytree. These parameters were chosen based on their potential impact on the model's ability to learn from the data and generalize to unseen data.
- **Cross-Validation Setup:** Given the time-series nature of our data, we used TimeSeriesSplit for cross-validation. This method ensures that the validation process honors the temporal order of the data, preventing data leakage and providing a realistic evaluation of the model's performance on unseen data.
- **Grid Search Execution:** We then ran the grid search over the defined parameter space, using the Mean Absolute Error (MAE) as the performance metric. The grid search systematically trained and evaluated LightGBM models on all combinations of the parameters in the grid, identifying the set of parameters that resulted in the lowest MAE.
- **Hyperparameter Selection:** Upon completion of the grid search, we extracted the best-performing hyperparameters. These parameters were then used to configure our final LightGBM model.

Notably, during our model training, even with a high number of iterations set to 6000, our models in each fold consistently ran through all iterations without early stopping. This observation suggested that our models were learning effectively from the data without overfitting, as overfitting typically results in early stopping due to no further improvement in validation error.

Hyperparameter Tuning for XGBoost:

In our XGBoost, after extensive engineering of features we employed these parameters:

- Number of Trees (n_estimators): A significant parameter in XGBoost is n_estimators, which defines the number of trees in the model. In the code, this parameter is dynamically optimized using a cross-validation setup, ensuring that the model neither underfits nor overfits. The use of 550 trees indicates a preference for a complex model, capturing subtle patterns in the stock market data.

- Learning Rate: The learning rate controls the step size at each iteration while moving towards a minimum of a loss function. A smaller learning rate (0.01 in the code) is chosen for gradual learning and better generalization.

- Max Depth: The maximum depth of a tree, set to 5, controls the depth of the individual trees. Restricting tree depth helps in preventing the model from becoming overly complex and overfitting the training data.

- Regularization Terms (alpha, lambda): Regularization is key in preventing overfitting. In the code, alpha (L1 regularization term) and lambda (L2 regularization term) are fine-tuned. This inclusion helps in reducing the model's complexity by penalizing the weights.

- Subsample and Colsample_bytree: These parameters control the fraction of samples and features used for training each tree. Subsampling (set to 0.6) and column sampling (set to 0.8) introduce randomness into the model, making it more robust and preventing overfitting.

- Subsample and Colsample_bytree: These parameters control the fraction of samples and features used for training each tree. Subsampling (set to 0.6) and column sampling (set to 0.8) introduce randomness into the model, making it more robust and preventing overfitting.

Final Model Parameters for LightGBM: After the grid search and additional fine-tuning through trial and error, we finalized the following parameters for our LightGBM model:

- *Objective*: Mean Absolute Error (MAE)

- *Number of Estimators (n_estimators)*: 6000

- *Number of Leaves (num_leaves)*: 256

- *Learning Rate*: 0.01

- *Subsample*: 0.6

- *Colsample_bytree*: 0.8

- *Max Depth*: 11
- *Regularization Alpha (reg_alpha)*: 0.2
- *Regularization Lambda (reg_lambda)*: 3.25
- *Other Parameters*: Optimized for GPU acceleration, minimal verbosity, and feature importance based on gain.

The process of grid search for hyperparameter optimization was instrumental in enhancing the performance of our LightGBM model, ensuring it was well-tuned to the specifics of our dataset and the task at hand.

Submission and Validation Using the Time-Series API

For the Kaggle competition, submissions and validations of our predictions are conducted using the provided Python time-series API. This approach is critical to ensure that our models adhere to the competition's rules, particularly concerning the chronological order of the data. The API facilitates a realistic simulation of a live trading environment, where future data is not accessible at the time of prediction, thus preventing any look-ahead bias.

Implementation Strategy: Our implementation strategy for using the API involves the following steps:

- *Environment Setup:* We initialize the Kaggle environment using the provided API. This environment mimics the real-time data flow in a trading scenario, providing new test data in each iteration.
- *Iterative Prediction:* The API provides a function to iterate over the test set. In each iteration, we receive a batch of new test data representing the current state of the market. This iterative process mimics real-time data streaming, where predictions are made as new data arrives.
- *Data Preparation and Feature Generation:* For each batch of test data, we perform the same feature engineering steps as we did for our training data. It's crucial to maintain consistency in feature generation to ensure that the model's input during inference matches its training.
- *Model Inference:* We use the trained LightGBM models to generate predictions for the current batch of test data. Since our model ensemble consists of multiple folds, we calculate a weighted average of predictions from all fold models. This ensemble approach helps in reducing variance and improving the robustness of our predictions.

- *Post-Processing and Submission:* After generating predictions using our LightGBM models, we apply crucial post-processing steps before submission:
  - *Mean Adjustment:* Instead of enforcing a zero-sum constraint, we adjust the predictions by subtracting their mean. This centers our predictions around zero, which is a more natural adjustment considering the nature of the target variable in the dataset.
  - *Clipping:* We clip the adjusted predictions to a predefined range, ensuring they remain within reasonable and realistic bounds. For our model, this range is set between -64 and 64.

Initial Zero-Sum Approach

We initially implemented a zero-sum constraint in our inference pipeline, aiming to balance the total sum of our predictions to zero. This method, commonly used in financial modeling, is based on the assumption that the market is zero-sum - for every winner, there is a loser, and vice versa. The zero-sum approach was implemented by adjusting the predictions by considering the standard error of the volumes, ensuring that the sum of all adjustments equaled zero. However, we observed that while this approach has its merits, it may not always reflect the true dynamics of market closing prices, which do not necessarily sum to zero due to market trends and directional biases.

Transition to Mean Adjustments

To address this limitation, we shifted our strategy towards subtracting the mean of the predictions from each prediction. This approach aims to center the distribution of our predictions around zero, without enforcing a strict zero-sum constraint. This simple yet effective change resulted in our model's improved accuracy and ranking on the Kaggle leaderboard. The mean adjustment technique allows for individual predictions to be more reflective of the underlying data and market sentiment, rather than being artificially adjusted to meet the zero-sum condition.

- *Performance Tracking:* Throughout the competition, we closely monitor the performance of our submissions. The API provides feedback on our model's performance, allowing us to make necessary adjustments to our strategy.

Challenges and Considerations: Working with a time-series API presents unique challenges:

- *Latency and Efficiency:* The API simulates a real-time environment where latency can be a critical factor. Our implementation ensures that the feature generation and prediction steps are efficient, minimizing the time taken per iteration.
- *Data Handling:* Since the API provides data in batches, careful handling of incoming data is necessary to maintain the temporal integrity of the dataset. We implement data caching strategies to efficiently manage and process incoming data.
- *Model Stability:* The real-time nature of the competition requires our models to be stable and reliable under varying market conditions. This is achieved through rigorous validation and the use of ensemble techniques.
- *Compliance with Competition Rules:* It is imperative to strictly adhere to the rules set by the competition, especially regarding the use of external data and prevention of look-ahead bias.

Through meticulous planning and implementation of the above strategy, we ensure that our submissions via the time-series API are both efficient and compliant with the competition guidelines. This approach not only helps in maintaining the integrity of our predictions but also provides us with valuable insights into their real-world applicability.

Conclusion

Our evaluation strategy provides a robust and comprehensive framework for assessing the performance of our predictive models, guiding our development process, and ensuring that our submissions to the Kaggle competition are based on well-vetted and reliable models. Through meticulous validation, error analysis, and adherence to competition guidelines, we achieve a competitive standing in the competition.

d) Results

Introduction to Results

The primary objective of our project was to develop a robust predictive models capable of accurately forecasting the closing price movements of NASDAQ-listed stocks. This section presents the outcomes of our modeling efforts, evaluated primarily through the lens of the Mean Absolute Error (MAE), a critical metric in the Kaggle competition we participated in.

The Mean Absolute Error, representing the average magnitude of the errors in a set of predictions, without considering their direction, served as the benchmark for assessing our model's performance. It quantifies the difference between the predicted values and

the observed data points, thus providing a straightforward and universally comparable measure of prediction accuracy.

Performance on Kaggle Leaderboard

In this section, we present our model's current standing on the Kaggle competition leaderboard, highlighting its achievement in reaching the top 25% percentile. We delve into the feature importance analysis, showcasing the most influential factors driving the model's predictions. Additionally, we provide a comprehensive evaluation of the model's performance, including insights gleaned from our rigorous hyperparameter tuning and optimization processes.

Our LightGBM model achieved a notable standing in the Kaggle competition, securing a place within the top 25% percentile on the leaderboard. This achievement is a testament to the effectiveness of our model in predicting the closing price movements of NASDAQ-listed stocks.

Our XGBoost model while initially in the top 25%, fell within the top 28% on the leaderboard by the time of writing this report. This isn't a remarkable position - however its a testament to tree based models to perform generally better without the contributions done by other experienced people on the leaderboard.

The Neural Network model struggles to perform on this competition dataset, achieving MAE of 5.4056, below our baseline LightGBM model. Due to the nature of the data, datapoints from early in the closing window is expected to be missing features such as the Near Price and Far Price, and well as gaps in the time-based comparison, the Neural Network struggles to learn and make prediction on the sparsely populated data. Moreover, as our number of feature increases, Neural Network demands intense resources and becomes increasingly difficult to implement. The final performance of the Neural Network model with MAE of 5.4056 ranks in the 30th percentile on the leaderboard.

Final Leaderboard Ranking and MAE Score: As of December 11th,2023, our best model based on LightGBM ranked 248th out of 4064 participating teams, with a Mean Absolute Error (MAE) of 5.3364. This performance underscores our model's high predictive accuracy and aligns with our objective of achieving a MAE significantly lower than baseline models.

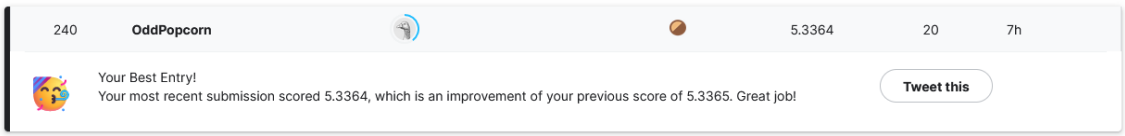


Figure 1: Present ranking of our best LightGBM model on the kaggle public leaderboard.

Baseline Model and Comparison: Since the competition did not provide a baseline model, we established our own simple baseline model for initial comparison. Our advanced LightGBM model significantly outperformed this LightGBM baseline, with an improved MAE score of 5.3364, compared to the baseline's MAE of 5.3620. While our XGBoost and Neural Network methods fell short of improving over the baseline set by LightGBM with scores of 5.465 and 5.4056 respectively.

Strategic Insights: The success of our model on the leaderboard can be attributed to several key strategies:

- Rigorous feature engineering, which allowed us to capture the complex dynamics of the stock market during the NASDAQ Closing Cross auction.
- Effective handling of the time-series nature of the data, using Time Series Cross-Validation to prevent data leakage and ensure model robustness.
- Systematic hyperparameter tuning through grid search and trial-and-error, which optimized our LightGBM model for better performance and generalization.

Comparison of Performance between Models

Our LightGBM-based model achieves the best performance in this competition, achieving MAE of 5.3364, placing 223rd in the competition. We compare the performance of LightGBM to the other models we tested:

Model	MAE	Difference (from Baseline)	Features count
LightGBM Baseline	5.3620	-	-
XGBoost	5.465	0.103	136
LightGBM	5.3364	0.0256	-
Neural Network	5.4056	0.0436	112

Compared to the baseline model, Neural Network struggles to learn the data of this competition, achieving an MAE below the baseline model. The sparse nature of the data due to the time-dependent features make Neural Network unsuitable to the task compared to LightGBM. Although, we expected XGBoost to perform closer to LightGBM, our results indicate differently. Suggesting that we need to spend more time on engineering features; as several runs on tuned parameters showed no improvement.

Feature Importance Analysis

A comprehensive feature importance analysis was conducted to identify the drivers of the predictive model in the LightGBM model. This analysis is essential to understanding

which variables most significantly impact the model's predictions. The following figure illustrates the top 40 features ranked by their importance, measured by the gain when using each feature in the splits of the model.

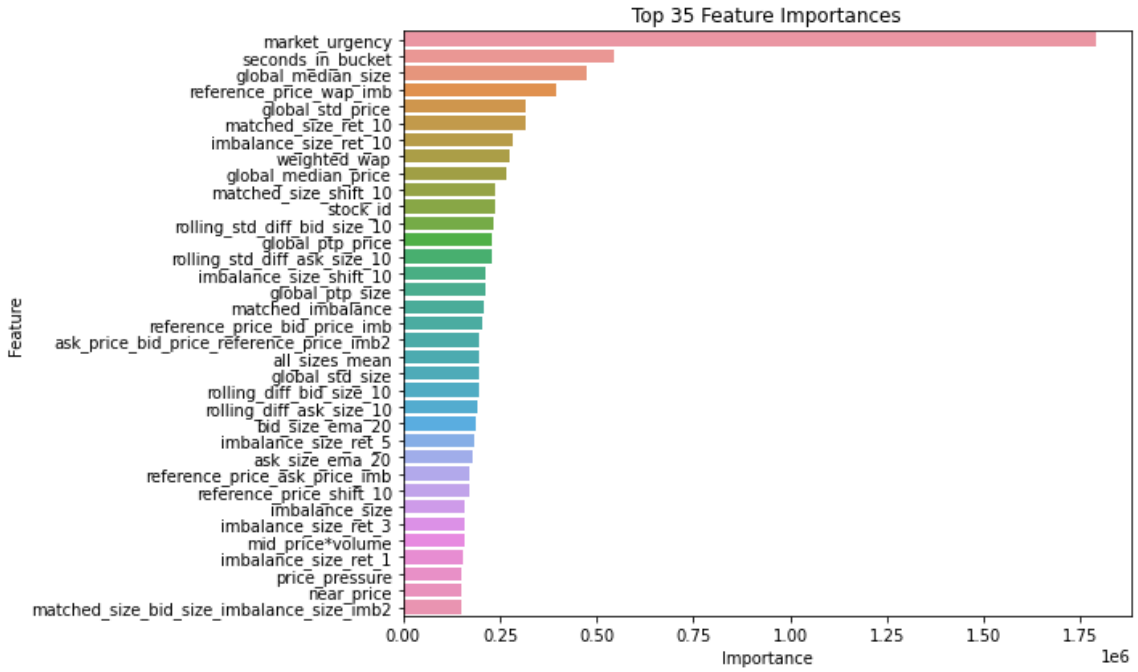


Figure 2: Top 35 Features Importance for final LightGBM model. The features are ranked by the average gain across all splits.

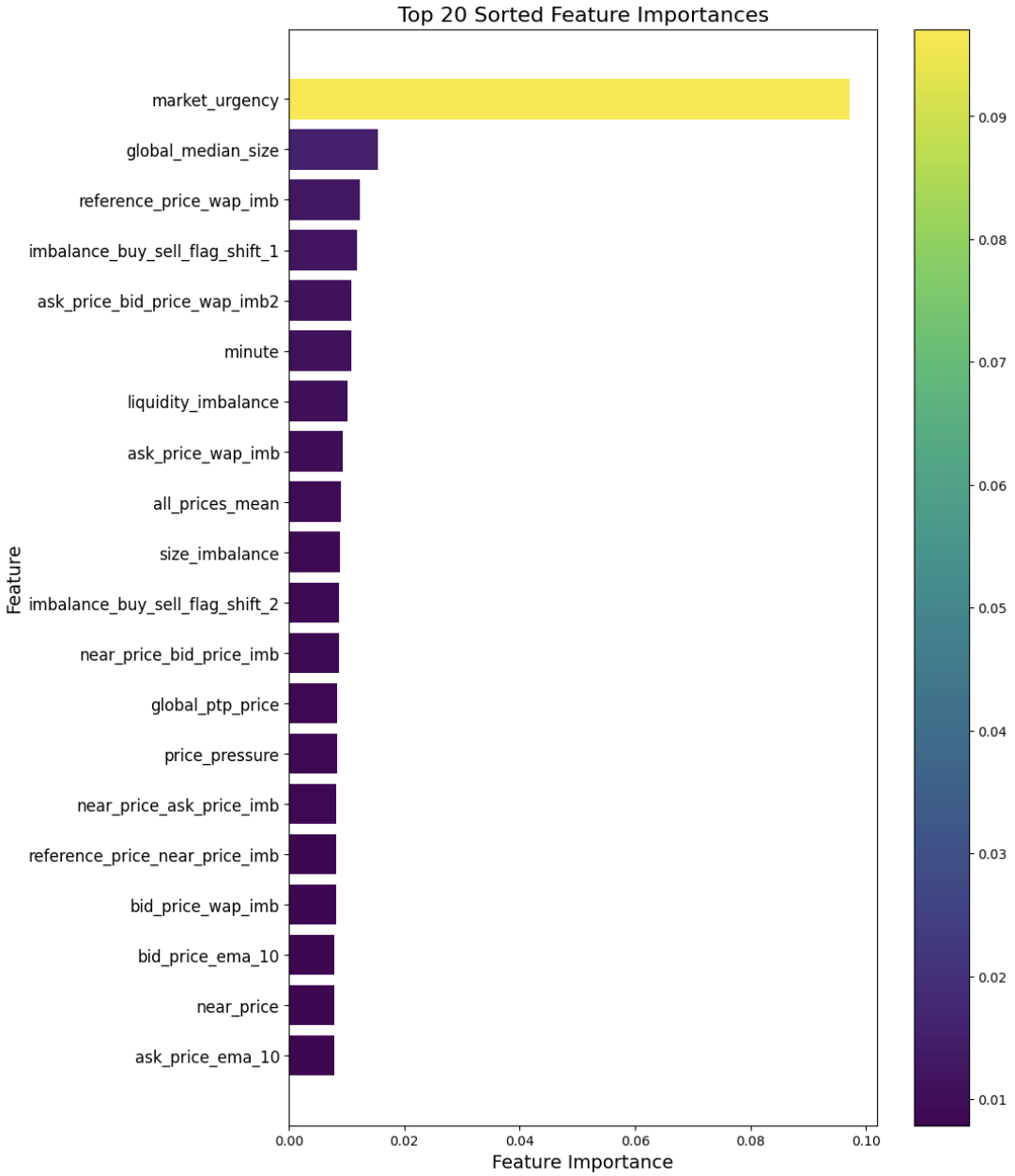


Figure 3: Top 20 Features Importance for XGBoost. The features are ranked by the average gain across all splits.

Significance of Top Features

Our models' feature importance analysis illuminated the most critical predictors in forecasting the closing price movements. This subsection discusses the relevance and significance of the top features identified in the analysis, drawing from our comprehensive feature generation process.

Market Urgency The 'market urgency' feature emerged as the most significant predictor, reflecting the immediate pressure in the market to execute trades. This feature

likely captures the last-minute surges in trading activity as the market nears the closing bell, where the urgency to match orders have a substantial impact on the final closing price.

Seconds in Bucket Closely related to market urgency, 'seconds in bucket' represents the temporal dimension within the closing auction, marking the countdown to the market close. This feature indicates how time, as a finite resource towards the end of the trading day, influences the order book dynamics and price settlement process.

Global Median Size and Reference Price WAP Imbalance These features encapsulate the broader market context, with 'global median size' reflecting the median order size across the market and 'reference price wap imb' highlighting the deviation of the weighted average price from the reference price. They underscore the influence of market-wide liquidity conditions and the prevailing market sentiment on individual stock prices.

Matched Size and Imbalance 'Matched size ret 10' and 'imbalance size ret 10' denote the recent changes in matched orders and the order imbalance size, respectively. These features are indicative of how recent trading activity and the supply-demand imbalance can foreshadow imminent price movements.

Weighted WAP By incorporating stock-specific weights into the weighted average price calculation, 'weighted wap' provides a nuanced view that accounts for the relative importance of different stocks in the index. It reflects how individual stock movements can collectively sway the index's direction.

Liquidity and Price Dynamics The model incorporates several features that capture the dynamics of liquidity and price movements, such as 'liquidity imbalance', 'price spread', and 'micro price'. These features reflect the bid-ask spread's tightness, the balance between buy and sell orders, and a liquidity-weighted price, respectively, all of which are crucial for understanding minute-to-minute price changes.

Statistical Aggregations Our model leverages the power of statistical aggregations, such as mean and standard deviation across all prices and sizes ('all_prices_mean', 'all_sizes_std', etc.), to capture the market's general state and identify patterns that may not be evident at the individual order level.

Temporal Shifts and Returns The inclusion of lagged and return features, like 'matched_size_shift_10' and 'imbalance_size_ret_5', allows the model to consider historical

context in its predictions, acknowledging the temporal continuity inherent in financial time series data.

Price Change Differences By calculating the differences in price changes across various windows ('price_change_diff_window'), the model assesses the momentum and volatility, providing insights into the aggressiveness of buyers versus sellers.
These top features, derived from our feature generation process, reflect the multifaceted nature of the financial markets, where both micro-level order book details and macro-level market conditions interplay to determine stock prices. Each feature contributes a piece to the complex puzzle of predicting closing prices, and together, they form the backbone of our model's predictive power.

Computational Efficiency

A key aspect of our project's success is its computational efficiency, particularly given the time-sensitive nature of stock market predictions. Our models were designed not only for accuracy but also for their ability to process and analyze data within a reasonable timeframe

The total execution time of our final Kaggle submission, encompassing all processes from data preparation to model inference, was approximately 5.7 hours. This timeframe includes the 3.231 hours estimated for reasoning about the data, training, and generating the single LightLGB model, and 2.3385 hours for the submission process via the Kaggle API. The submission process involved not only generating predictions using our LightGBM model but also handling data features, implementing our hyperparameter-tuned model, and complying with the API's sequential data processing requirements.

Importantly, our model adheres to the Kaggle competition's requirement of maintaining a total runtime of less than 9 hours. This compliance is critical in ensuring not only the validity of our submission but also its practical applicability in real-world scenarios, where timeliness is as crucial as accuracy.

The efficiency of our model is crucial in a real-world scenario where financial decisions need to be made rapidly based on the latest market data. Our model's ability to operate within these time constraints, without compromising on the accuracy of predictions, is a testament to the effectiveness of our optimization and implementation strategies.

It's important to note that computational efficiency can vary with different hardware and computational resources. Our results were obtained in the Kaggle environment, which offers a specific set of computational capabilities. However, the model's architecture and

our optimization efforts aim to ensure that it remains efficient across various platforms, aligning with the practical needs of real-time financial analysis and decision-making.

In summary, our project demonstrates a balanced approach to computational efficiency and predictive accuracy, making it a viable tool for real-time stock market analysis and prediction, while adhering to the stringent time constraints set by Kaggle.

e) Conclusions

1. Our venture into the domain of stock market prediction and participation in a Kaggle competition was both challenging and enlightening. As novices in both areas, we achieved a reasonable rank in the competition, which speaks volumes about our model's efficacy in predicting the closing prices of NASDAQ-listed stocks and our team's dedication to overcoming steep learning curves.

2. The learning curve for this project was steep, but immensely educational. Our achievement in ranking within the top 25% percentile in the Kaggle competition is not just a testament to our model's performance but also to our team's adaptability and commitment to learning. The process of transforming from novices in stock market analytics to developers of a competitive predictive model is a significant part of our project's success story.

3. A critical component of our learning journey was the Kaggle competition discussion board. This platform proved to be an invaluable resource, offering insights, advice, and a collaborative environment that greatly facilitated our understanding of both the Kaggle competition framework and the nuances of stock market prediction. The community's willingness to share knowledge and experiences played a crucial role in guiding our modeling choices, feature engineering strategies, and overall approach to the problem statement.

4. One of the most formidable challenges we faced was understanding the intricacies of feature engineering specific to financial data. The complex nature of stock market data, with its inherent volatility and the multitude of factors influencing stock prices, required us to delve deeply into financial theories and market behaviors. Developing features that could capture the nuanced dynamics of the market was both challenging and crucial for our model's performance.

5. Tuning the models for this specific application was another significant hurdle. We had to balance the precision of predictions with the computational efficiency required for processing large volumes of market data. Experimenting with various algorithms and hyperparameters to find the optimal combination was a time-intensive

and intricate process, which however, was crucial for enhancing the predictive power of our models.

6. The steep learning curve was not limited to technical aspects but also extended to understanding the Kaggle competition environment. This was our team's first experience in such a competitive and collaborative platform. The Kaggle discussion board was a lifeline, providing insights and guidance that were instrumental in shaping our strategies and approaches.

7. Addressing the challenges in optimizing our Neural Network model remains an open area for future exploration. Improving this model could involve experimenting with different neural network architectures and advanced techniques in feature engineering. Currently, training a Neural Network requires significantly more resources to construct a deeply connected layers especially in terms of computer memory, and fails to match up to LightGBM performance on tabular time-series data.

To summarize, our maiden journey into stock market prediction and Kaggle competitions was rich in learning and achievements. The project's success was shaped by overcoming the complexities of feature engineering, model tuning, and adapting to the Kaggle environment, all amidst the supportive backdrop of the Kaggle community. This formative experience has equipped us with valuable insights and skills for future ventures in financial technology.

f) Individual Tasks

Our team, comprising three members, carefully distributed tasks to optimize both individual learning and collective progress. With our limited prior experience in financial data analysis, we embraced a collaborative approach, emphasizing knowledge sharing and resource utilization, notably through the Kaggle discussion board.

- **Raghu Ram Sattanapalle (LightGBM Model):** Led the LightGBM model development, focused on feature engineering and hyperparameter tuning. Responsibility extended beyond just applying the algorithm; it involved crafting a robust set of features and optimizing the model for accuracy and efficiency. A significant part of the contribution was developing a comprehensive feature generation strategy, time series based cross validation strategy, zero-sum, and mean optimization model inference for submission via Kaggle's API.

- **Eshan Arora (XGBoost Model):** Leading the XGBoost model development for our project, I initiated with in-depth exploratory data analysis and comprehensive research on statistical models like ARIMA, transitioning to a focus on tree-based

models and hyperparameter tuning. My primary contribution lay in the implementation of a finely-tuned XGBoost tree model, which involved several runs and feature engineering. Throughout the project, I developed a profound understanding of financial data attributes, decision tree-based regression models, and the nuances of functional gradient boosting methods.

- **Pazin Tarasansombat (Neural Network Model):** Responsible for the Neural Network implementation. Focus on finding a Neural Network model that can learn on tabular data with time-based feature, exploring various Neural Network implementations. Implemented FT-Transformer based Neural Network for the project, experimenting with Transformer parameters to contextualize tabular data, attempting to produce a Neural Network model that can perform competively with Tree-Based model.

  The Kaggle discussion board was a linchpin in our learning and development process. It was an invaluable resource for gathering insights and strategies, significantly shaping our feature engineering and model tuning efforts. Regular interaction with this vibrant community deepened our project's analytical scope and effectiveness.

  In summary, our project epitomized the essence of teamwork, shared learning, and proactive engagement with external resources.

References

[1] BlackRock. *A Global Perspective on Market-On-Close Activity*. July 2020. URL: `https://www.blackrock.com/corporate/literature/whitepaper/viewpoint-a-global-perspective-on-market-on-close-activity-july-2020.pdf`. Accessed: 2023-12-10.

[2] NASDAQ. *Closing Cross Frequently Asked Questions*. 2023. URL: `https://nasdaqtrader.com/content/ETFs/closing_cross_faqs.pdf`. Accessed: 2023-12-10.

[3] Tom Forbes. *Optiver: Trading at the Close - Introduction*. 2023. URL: `https://www.kaggle.com/code/tomforbes/optiver-trading-at-the-close-introduction/notebook`. Accessed: 2023-12-10.

[4] Kaggle. *Optiver - Trading at the Close*. 2023. URL: `https://www.kaggle.com/competitions/optiver-trading-at-the-close/data`. Accessed: 2023-12-10.

[5] K.J. Kim, W.B. Lee, *Stock market prediction using artificial neural networks with optimal feature transformation*, Neural Comput. Appl., 13 (3) (2004), pp. 255-260.

[6] S.K. Chandar, M. Sumathi, S.N. Sivanandam, *Prediction of the stock market price using a hybrid of wavelet transform and artificial neural network*, Indian J. Sci. Technol., 9 (8) (2016), pp. 1-5.

[7] A. Sharma, D. Bhuriya, U. Singh, *Survey of stock market prediction using machine learning approach*, In 2017 International conference of Electronics, Communication and Aerospace Technology (ICECA), Vol. 2, (2017), pp. 506-509. IEEE.

[8] D. Enke, M. Grauer, N. Mehdiyev, *Stock market prediction with multiple regression, fuzzy type-2 clustering, and neural networks*, Procedia Comput. Sci., 1 (6) (2011), pp. 201-206.

[9] X. Li, H. Xie, R. Wang, Y. Cai, J. Cao, F. Wang, X. Deng, *Empirical analysis: stock market prediction via extreme learning machine*, Neural Comput. Appl., 27 (1) (2016), pp. 67-78.

[10] E. Chong, C. Han, F.C. Park, *Deep learning networks for stock market analysis and prediction: Methodology, data representations, and case studies*, Expert Syst. Appl., 83 (2017), pp. 187-205.

[11] E. Chong, C. Han, F.C. Park, *Deep learning networks for stock market analysis and prediction: Methodology, data representations, and case studies*, Expert Syst. Appl., 83 (2017), pp. 187-205.

[12] J. Patel, S. Shah, P. Thakkar, K. Kotecha, *Predicting stock and stock price index movement using trend deterministic data preparation and machine learning techniques*, Expert Syst. Appl., 42 (1) (2015), pp. 259-268.

[13] K. Zhang, G. Zhong, J. Dong, S. Wang, Y. Wang, *Stock market prediction based on generative adversarial network*, Procedia Comput. Sci., 147 (2019), pp. 400-406.

[14] J. Li, H. Bu, J. Wu, *Sentiment-aware stock market prediction: A deep learning method*, In 2017 International Conference on Service Systems and Service Management, (2017), pp. 1-6. IEEE.

[15] LightGBM Documentation. *Features — LightGBM 3.3.1.99 documentation.* URL: `https://lightgbm.readthedocs.io/en/latest/Features.html`. Accessed: 2023-12-10.

[16] LightGBM Documentation. *Welcome to LightGBM's documentation! — LightGBM 3.3.1.99 documentation.* URL: `https://lightgbm.readthedocs.io/en/latest/index.html`. Accessed: 2023-12-10.

[17] Ke, Guolin, et al. *LightGBM: A Highly Efficient Gradient Boosting Decision Tree.* NeurIPS 2017. URL: https://proceedings.neurips.cc/paper_files/paper/2017/file/6449f44a102fde848669bdd9eb6b76fa-Paper.pdf. Accessed: 2023-12-10.

[18] *XGBoost Documentation.* URL: https://xgboost.readthedocs.io/en/stable/ Accessed: 2023-12-10.

[19] Tianqi Chen, Carlos Guestrin *XGBoost: A Scalable Tree Boosting System.* URL: XGBoost:AScalableTreeBoostingSystem Accessed: 2023-12-10.