


RICE ENGINEERING
Computer Science

# Trading At The Close

COMP-642
Machine Learning Project

***Students:***

Soham Ambre (sa188)
Diana Dutava (dd65)
Fernando Raverta (fr15)

# Contents

# 1 Introduction

This project, based on the "Trading At the Close" Challenge on Kaggle [5], explores stock closing price prediction in a focused scenario, specifically within the NASDAQ Market. The NASDAQ has a unique procedure for determining the closing price of stocks that includes releasing bid information 10 minutes before the trading day concludes [3]. The aim of this project is to assess if machine learning can leverage the information released to forecast the variation of a stock's price one minute after relative to a stock index (target). The composition of the stock index is not released. However, it is said to be a weighted average of the stock's prices.

We have explored the literature on market prediction topics. We found this is a controversial topic, with disputes about whether stock prices are random walks and, as a consequence, unpredictable or if there is a pattern, such as mean reversion, that can be predicted [2]. The weak stock market hypothesis, which is taken as an assumption by some asset pricing methods, suggests that any information about the asset prices is already included in the prices. Consequently, this hypothesis implies that no method can achieve superior returns on a consistent basis based on historical price, volume, and returns. However, whether prices can be predicted or not may not have a global answer and may depend on the efficiency of each market and the asset itself [1].

# 2 Problem and Data Description

This problem though being stock price prediction, it differs from the general setting which relies mostly on price and volume series. In this case, the aim is to predict the variations of stocks at $t+60$ but considering specific information that NASDAQ releases only in the last 10 minutes of the trading day.

To dive into the setting of the problem, it is mandatory to explore some generalities about the NASDAQ market and its closing procedure. As shown in figure 2, NASDAQ keeps 2 books for stock operation, the *continuous trading book* and the *auction trading book*. In the continues trading book, orders are executed when a seller agrees to sell at a specific price $p$ and a buyer is willing to purchase at a price $P$ such that $P >= p$. In contrast, in the auction trading book, orders are not processed until the end of the trading day. The auction trading book contains three types of orders: Market-on-Close (MOC), Limit-on-Close (LOC), and Imbalance-Only (IO). However, the distinctions between these orders are beyond the scope of this discussion.

Table 1: Continuous Order Book

| BID | PRICE | ASK |
|---|---|---|
| - | 10 | 1 |
| 2 | 9 | - |
| - | 8 | - |

Orders are executed once they are matched at an agreed-upon price.

Table 2: Auction Order Book

| BID | PRICE | ASK |
|---|---|---|
| - | 10 | 1 |
| 3 | 9 | 2 |
| 4 | 8 | 4 |

Orders are accepted but not executed until the market closes.

Table 3: Combined Order Book

| BID | PRICE | ASK |
|---|---|---|
| - | 10 | 2 |
| 5 | 9 | 2 |
| 4 | 8 | 4 |

At the end of the trading day, both books are combined.

Figure 2: Types of trading books at NASDAQ

As depicted in fig. 3, the closing procedures begins at 2:50 CDT. Since at that time, NASDAQ starts to release information about the continuous and auction book every 10 seconds.

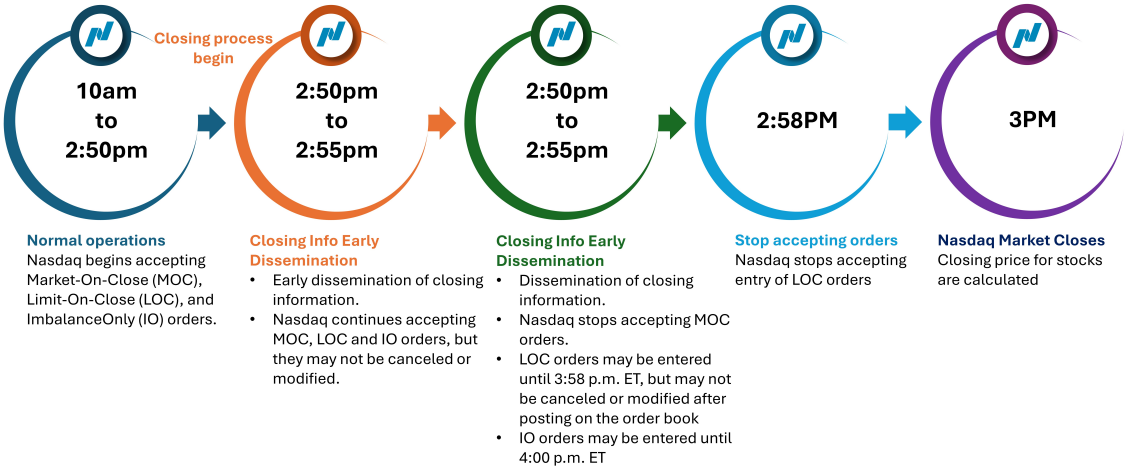


Figure 3: NASDAQ Closing Procedure timeline

The information released at 14:50 pm CDT is depicted in Fig. 4, this data is updated every 10 seconds. The values shown are based on the market state described by the trading books on Fig. 2 considering them belong to *"stock 3"* at 2:57:10 CDT of day *42* (bottom right corner). The bid and ask values are taken from the continuous trading book and represent the highest buy price offered (bid) and the lowest selling price offered (ask). The amount of the stock offered to either buy and sell is listed in the bid and ask sizes respectively. The last value taken from this table is the weighted average price which is calculated as a weighted average of the bid and ask prices as shown in the upper right corner of fig. 2.

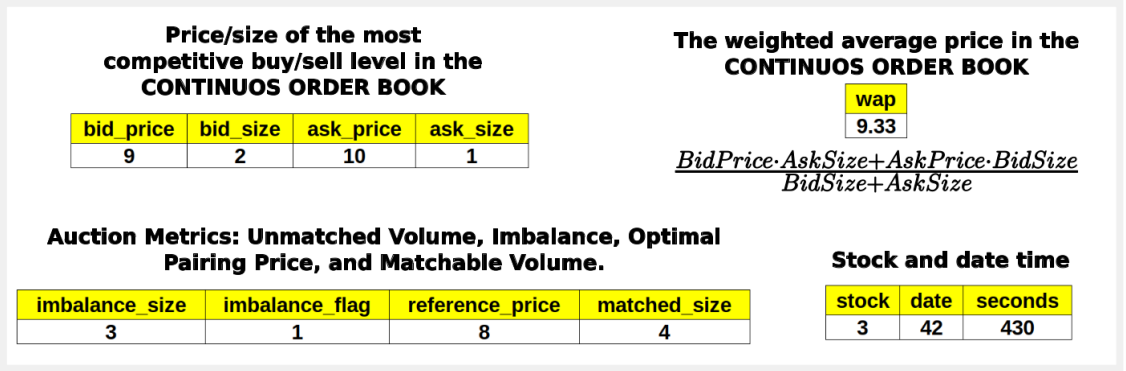


Figure 4: Data NASDAQ releases starting from 14:50 CDT until the end of the trading 10. Data is updated every 10 seconds

The auction metrics (bottom left corner 4), are calculated from the auction book table. Particularly, the reference price is the price that maximizes the matched orders. We have two candidates: 9 and 8. The reference price would be 8 since there are offers to buy 7 stocks at price 8 or higher and 4 to sell at that price. However, only 3 operations would be performed at price 9. As there are 3 buy orders at the reference price that did not match with a seller, there is an imbalance of size 3 on the buying side (i.e $imbalance\_flag = 1$ and $imbalance\_size = 3$).

Two additional metrics are released starting from 2:55 CDT: *near price* and *fair price*. These are also updated every 10 seconds. The *near price* is like the reference price but taken from the combined book (see fig. 2). In this case, its value would be 9 as it is the price that maximizes the matched operations. Notice that at this price, 5 orders would be executed meanwhile at price 8 only 4. The *fair price* is taken from the auction book only and its difference with the reference price is too subtle to be discussed here.

Based on the information available at time $t$, the objective is to predict the variation in the stock price at $t+60$, relative to the movement of a weighted average price of 200 NASDAQ stocks, known as the *Index*. The dataset does not provide the value or composition of the index. In Section 3.1, we will discuss this aspect in detail. Below is the equation for the prediction *target*:

$$\text{Target} = \left( \frac{\text{StockWAP}_{t+60}}{\text{StockWAP}_t} - \frac{\text{IndexWAP}_{t+60}}{\text{IndexWAP}_t} \right) \times 10,000 \tag{1}$$

Note that this expression is in basis points, a commonly used unit in finance, which is why the difference between the stock and index variations is multiplied by 10,000.

Finally, it's important to highlight that the dataset does not include actual dates or stock names. Additionally, all prices are expressed as variations relative to the

preceding value.

# 3 Approaches for predicting the target

## 3.1 Approach 1: Single Stock Price Movement Predictors

The idea here is to train a single model for each stock to predict its ($\frac{stockWAP_{t+60}}{stockWAP_t}$). Then, use these prediction to calculate the index variation ($\frac{indexWAP_{t+60}}{indexWAP_t}$) and, lastly, compute the target based on these values as indicated in equation 5.

The hypothesis here is that **by isolating the behavior of each stock, and training a specific model to make prediction for each stock, we will be able to be more accurate in predicting the index movement and consequently the target**.

This approach is based on [7] and involves several stages:

1. **Predict stock weights on the index**: Assuming that the index is a weighted average of stocks prices and that stock weights does not change over time as indicated in the formula below:

$$\text{index\_return} = \left( \sum_{i=0}^{199} \frac{w^i \cdot \text{stockWAP}^i_{t+60}}{\text{stockWAP}^i_t} \right) \times 10{,}000 \tag{2}$$

It is required to perform a linear regression to know the stock weights ($\{w_i \mid \forall_i 0 <= i < 200\}$) in the index. From the target value, we can derive the return for the indexWAP since we know the return for the stockWAP. In this context, "return" refers to the ratio between a variable at $t + 60$ and its corresponding value at $t$. Therefore, we can compute the index_return by simple algebra as follows.

$$\text{stock\_return} = \left( \frac{\text{stockWAP}_{t+60}}{\text{stockWAP}_t} \right) \times 10{,}000 \tag{3}$$

$$\text{index\_return} = \text{stock\_return} - \text{target} \tag{4}$$

Notice that index return is expressed in basic points. With the $stock\_return$ and $index\_return$ calculated at each time $t$ we can run a linear regression to

predict $\{w_i \mid \forall_i 0 <= i < 200\}$ such that it optimizes the following optimization problem:

$$\arg\min_{w^i} \sqrt{\left(\left(\sum_{i=0}^{i<200} w^i \ stock\_return\right) - index\_return\right)^2}$$

We performed the linear regression splitting data as 425 day of data for training and the rest for testing. Unsurprisingly, we found a set of coefficients $\{w_i \mid \forall_i 0 <= i < 200\}$ such that they accurately computed the index with a $R2 \approx 1$ in both training and test data.

2. **Predict stock prices variation**: For each stock, a model is trained with the aim of predicting the price return at $t+60$. We trained Prophet and XGBoost models.

3. **Calculate target at time t:** based on the variation computed on 2) and the weights computed in 1), a prediction for the target can be calculated as follows:

$$\hat{target}_t = \left(\frac{\hat{stockWAP}_{t+60}}{\text{stockWAP}_t} - \left(\sum_{i=0}^{199} \frac{w^i \cdot \hat{stockWAP}^i_{t+60}}{\text{stockWAP}^i_t}\right)\right) \times 10{,}000 \qquad (5)$$

## 3.2 Approach 2: General Model for Target Prediction

The idea here is to train one model to predict variations for all the stocks at once. The model predicts the target directly. The hypothesis here is that **by using one model for all the stocks might be less accurate than individual models, but using a powerful enough model and right feature representation may still give good accuracy**.

A model involving all the stocks may find patterns that arise from the collective behavior of the stocks in the market leading to accurate predictions for each stock. Moreover, this model can be less influenced for a change on the index weight or composition. Additionally, this model can be used to predict prices of new stocks more easily.

For this approach we tried working on models such as an 1D CNN. This model, initially proposed in [6], was trained with additional feature imputations. For efficient training, we created synthetic lag features based on certain original features like imbalance size, reference price,bid and ask prices and WAP. The synthetic lag was up to 10 time values for each feature. This was based on the assumption that

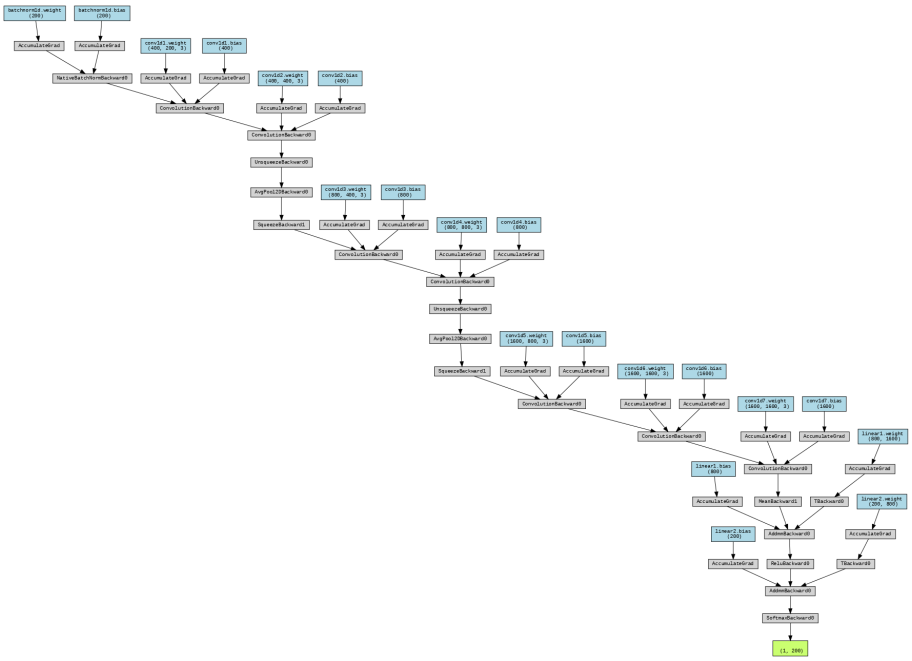

Figure 5: 1D CNN model representation

a stock value while considering individual variations would show similar behaviors for these features at time values t and t+100.

Other synthetic features that we created were delta scaled coefficients, which ranged between values of 0 and 1. For delta scaling of two values a and b, the delta scaled coefficient for a and b would be calculated as : (a-b)/(a+b). This scaling was applied to combinations of any 2 features from the following: bid price, ask price, reference price, WAP. We introduced a signed imbalance as well, which would consider the product of the imbalance flag and the imbalance value. We also considered the ratios between bid and ask prices for every stock value.

The model in consideration had an input layer of (200,400), as it was running a simultaneous prediction for targets for all the 200 stock values. The inputs were batch-normalized to prevent overfitting. Each of the convolutional layers were conv1d. Average pooling was used to condense the embeddings to prevent overfitting.

The embeddings from the convolutional layers were passed through a global average pooling layer, to make them readable to the following fully connected layer. Finally, the outputs were passed into another ReLU layer to obtain the finalized predictions for all 200 stocks.

The 1DCNN appears to give a training MAE that is very close to the baseline model (Fig. 9), i.e a model that simply predicts 0 for every target value. However, by using other metrics like "sign accuracy", i.e the accuracy of predicting the stock movement in the near future (t+60), we realized that the 1DCNN does not actually

seem to consider correlations with the target and WAP values of other stocks well enough. As a result, we have decided to pursue the usage of Large Language models (LLMs) like TimeGPT1[4], which greatly reduces the training pipelines to inferential training and skips out on a lot of complex steps that are required in appropriate data pre-processing.

We explored other models beyond neural networks, including random forest, as well as gradient boosting techniques like XGBoost and CatBoost. Our predictive methodologies diverged into two main strategies. Firstly, we employed a two-step prediction approach, initially forecasting individual stock returns, and subsequently leveraging linear regression coefficients to predict index returns. This process, described in detail in approach 3.1, allowed us to derive the target variable.

Alternatively, we pursued a direct prediction of the target variable. The direct prediction method is based solely on utilizing the features of an individual stock at time t to predict the target variable. However, it's noteworthy that the target variable is dependent on data from all stocks at the same time point (1).

The implementation of target prediction using ensemble models can be found in the *decision_trees.ipynb* notebook. In Figure 13, the superiority of ensemble models with direct prediction, such as random forest and gradient boosting over the baseline and 1D CNN is shown, prompting us to delve deeper into these models. We attribute this success to their robustness against overfitting compared to neural networks (NN). Our exploration involved fine-tuning these models, focusing on key hyperparameters such as the number of decision trees, maximum depth, and learning rate, alongside comprehensive feature engineering encompassing various combinations of lagged features. An interesting observation was the consistent selection of the largest number of decision trees (Fig. 6), lending support to our hypothesis of an overfitting issue. This finding underscores the importance of judiciously balancing model complexity with dataset characteristics to mitigate overfitting tendencies effectively.

Our efforts encountered challenges stemming from resource constraints. Firstly, the lack of native GPU acceleration support in the scikit-learn library for random forest meant that training a single set of parameters consumed approximately 2 hours on our dataset. To mitigate this, we turned to the cuML library, offering the capability to train random forests on GPU, thus accelerating our experimentation. Secondly, fine-tuning XGBoost posed persistent challenges due to consistent out-of-memory errors arising from our dataset's substantial size (Fig. 7). Nevertheless, leveraging native GPU utilization significantly reduced the training time for XGBoost in contrast to random forest taking now around 20 seconds. Consequently, we achieved further performance enhancements using XGBoost with optimal hyperparameter selection.

Our final model was implemented using CatBoost library and demonstrated the best results withing this approach. CatBoost provides a gradient boosting frame-

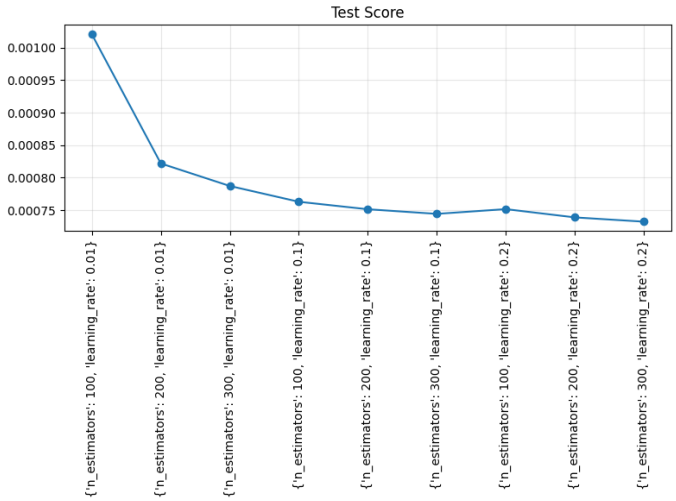


Figure 6: Fine-tuning XGBoost

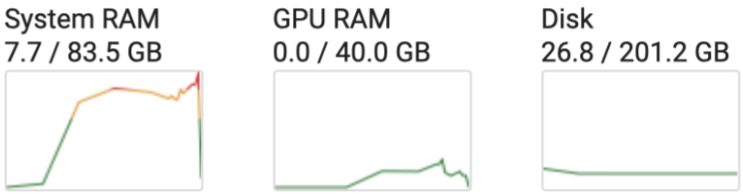


Figure 7: XGBoost Memory Usage

work similar to XGBoost but stands out with several distinct features, notably its native ability to handle categorical features and missing values. This capability proves particularly invaluable in our dataset, which demands intricate preprocessing due to a significant number of missing values, for instance those concerning missing far and near prices for the initial 5 minutes of each trading day. Additionally, CatBoost has its own internal method for handling categorical features efficiently. Stock ID being a categorical feature, can be naturally handled by CatBoost's Ordered Boosting algorithm. CatBoost automatically sorts and splits categorical features based on their values during the training process. This allows the model to learn from the categorical nature of the data directly, without the need for explicit encoding or preprocessing steps like one-hot encoding. In this model we also leveraged lagged features such as $imbalance\_size\_lag$, $reference\_price\_lag$, $matched\_size\_lag$, $bid\_price\_lag$, $ask\_price\_lag$, and $wap$ for up to 50 seconds in the past. Additionally, we utilized $stock\_id$ and $imbalance\_buy\_sell\_flag$ as categorical features to achieve better performance.

## 3.3 Approach 3: Ensemble of models for Target prediction

The concept centers on leveraging the strengths of the models crafted in approaches 1 and 2 to create a comprehensive ensemble framework for making predictions. The underlying hypothesis posits that an amalgamation of models, incorporating both individual stock analysis and a holistic market-wide view, will enhance the accuracy of predictions. To achieve this, various methods for integrating the outputs of each model are explored, including simple averaging, weighted averaging, and blending techniques. This strategic approach aims to leverage the unique insights provided by each model, thereby potentially offering a more robust and reliable prediction mechanism.

Due to the superior performance of gradient boosting decision trees (GBDT) in predicting the target variable, we chose to explore stacking ensemble techniques centered around them. We evaluated this approach for both two-step and direct target prediction methods. However, as of now, it has not resulted in noticeable improvements in prediction accuracy.

# 4 Experiment Results

The dataset comprise 5,237,892 records, each detailing information on 200 NASDAQ-listed stocks over a period of 481 days. For every trading day and each stock, there are 55 distinct entries, capturing market data at 10-second intervals from 14:50 to 14:59 CDT.

The dataset was spitted into 70% for training and 30% for testing. From day 0 till day 424 data will be used for training. Data from day 425 will be used for testing.

The models will be compared against its Median Average Error. The baseline considered for this project is a simple model that predicts the asset will increase 1 basis point if there is a buying imbalance, -1 basis point if there is a selling imbalance or 0 if there is no imbalance at all. This model reaches an Median Average Error of 5.85 on testing data.

Although the baseline heuristic is straightforward, predicting stock market prices is notoriously challenging—even when feasible. Achieving a model that consistently outperforms this baseline, even by a small margin, should be seen as a significant accomplishment.

## 4.1 Approach 1: Results

For this approach, first it was carried on a **validation analysis** to check if the general approach was feasible. We run a LR to find coefficients for each stock in the index and then we use these coefficients to calculate the $index\_return$ as described in formula 2. Then, we use the $wap$ values at $t+60$ to predict the target variable as indicated in eq. 5. The results are shown on fig. 8. As expected, we found a very tiny error $MAE = 0.008$ on training data and $MAE = 0.007$ on testing data. This minor error, likely due to the arithmetic accuracy of the predicted stock coefficients, confirms that this approach is worth pursuing.

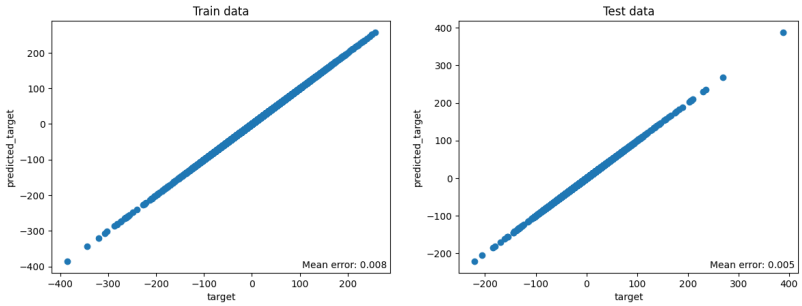


Figure 8: Target vs predicted target using approach described in section 3.1 when using the actual $WAP_{t+60}$.

We train models using Prophet and XGBoost to predict each stock price. They were trained with the records belonging to each stock and its goal was given the data at $t$ predict $wap_{t+60}$. Regarding the pre-processing pipeline before training we did scalling using *scikit-learn StandardScaler* for the numerical features ($imbalance\_size$, $reference\_price$, $seconds\_in\_bucket$, $matched\_size$, $far\_price$, $near\_price$, $bid\_price$, $bid\_size$, $ask\_price$, $ask\_size$, $wap$) and one-hot-encoding for $imbalance\_buy\_sell\_flag$). We did not use $date\_id$ nor $stock\_id$ for this approach.

Besides, for Prophet we had to provided dummy real dates to the data: we decided to use January 1st 2023 at 14:50 CDT as the starting datetime.

The predictions for $WAP$ using the XGBoost model, which achieved the lowest MAE, are illustrated in Figure 9. Despite the low MAE, the plot reveals clear evidence of overfitting. A similar pattern of overfitting was also observed with the Prophet model. Attempts to resolve this issue were unsuccessful. As a consequence, when this data was utilized to estimate the target at $t + 60$, the error on the test data was considerably higher than the baseline, with an $MAE \approx 15$ for the best XGBost model and $MAE \approx 15$ for the best Prophet model.

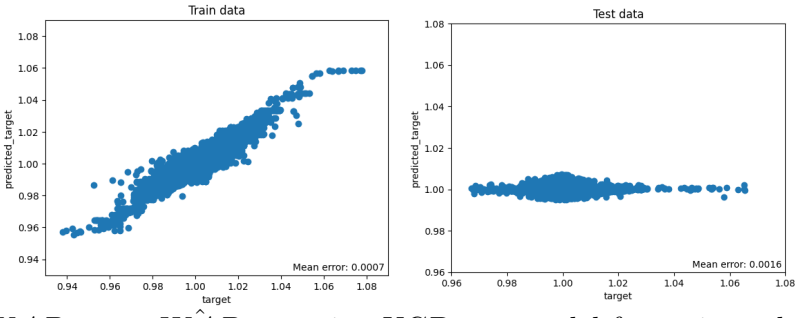


Figure 9: $WAP_{t+60}$ vs $\hat{WAP}_{t+60}$ using XGBoost model for train and test data. It shows overfitting.

## 4.2 Approach 2: Results

As stated earlier, the 1DCNN did give a training MAE that was very close to the baseline we obtain from predicting 0 for all target values. We used the 5 fold cross validation approach for efficient training of the 1D CNN model. This enabled us to track the MAE movement across all the folds, as well as the mean MAE and its comparison to the baseline.

The mean MAE for the 1D CNN approaches the baseline fairly closely. However, we observed a pattern in the training and validation losses that clearly suggested underfitting. The training loss remains constant throughout the entire training, as the validation loss drops across each fold. Additionally, the individual losses, as well

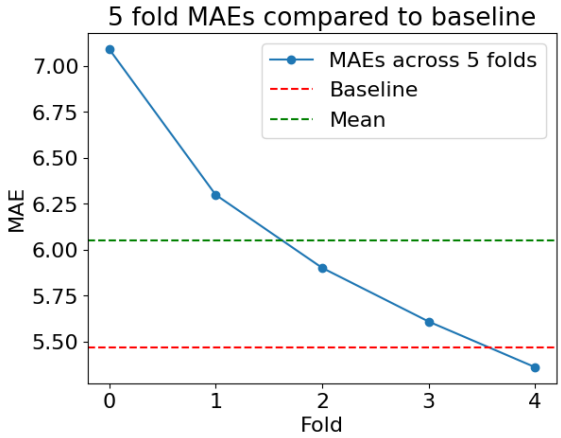


Figure 10: Training MAE movements against baseline for 1D CNN

as the MAEs remain constant throughout each epoch within each fold, triggering early stopping rather quickly due to negligible improvements.

We introduced an additional metric called "sign accuracy" in our evaluations. To leverage this metric, we compared the predicted target values with expected target values for sign similarity. The predictor was awarded 1 point for a correct prediction, and 0 points for an incorrect prediction. We observed that the sign accuracy for prediction with the 1D CNN was only 50.2%.

We show the results of the models explored in this approach in Fig. 13. We observe that ensemble models show promising results, despite their lack of utilization of data from all stocks to predict the target. Additionally, we were able to gradually increase the accuracy by careful feature engineering. Remarkably, the CatBoost model, augmented with lagged and categorical features, exhibited the highest accuracy among the models examined. With a resulting MAE of 5.783, it outperformed the baseline model by 1%.

In Fig. 11 and Fig. 12 we show that our prediction of individual stock returns (WAP at time t + 60) shows relatively good performance. However, our analysis revealed that the direct prediction of the target so far outperformed our initial expectations, demonstrating superior performance in terms of Mean Absolute Error (MAE) and Mean Squared Error (MSE) (Fig. 13). Despit that, we maintain our belief that, with careful model selection and feature engineering, the two-step approach holds promise for achieving optimal performance. To support this belief, we attempted to predict the target using the true value of the Weighted Average Price (WAP) at time t+60, yielding an MAE of 0.004 on our test set and $R2$ close to 1 (Fig. 14). This suggests that achieving accurate predictions for WAP at time t+60 could substantially enhance the performance of the two-step approach.

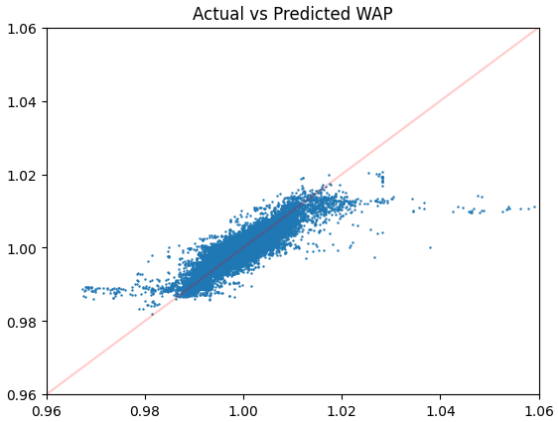


Figure 11: Actual vs Predicted WAP at time t+60

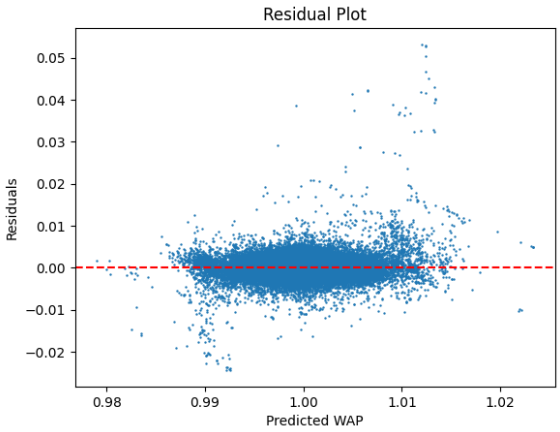


Figure 12: Residual plot for predicted WAP at time t+60

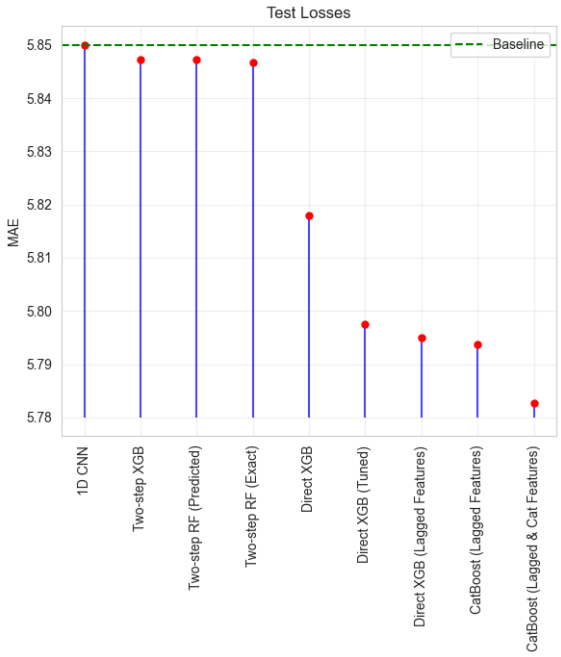


Figure 13: Models Comparison

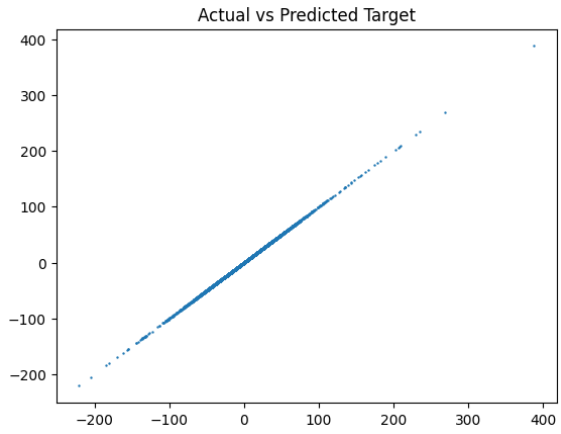


Figure 14: Actual vs Predicted Target using True WAP

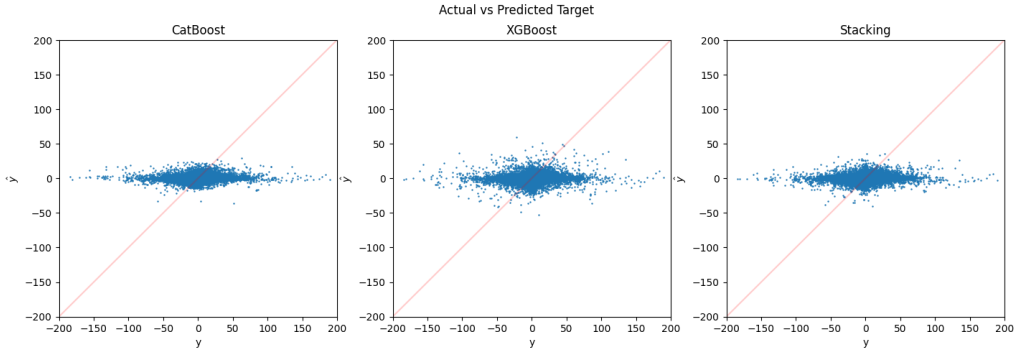


Figure 15: Direct Prediction: Actual vs Predicted Target

## 4.3 Approach 3: Results

Stacking GBDT models failed to enhance accuracy in either direct or two-step target prediction, according to our experimental findings. However, it remains a viable technique, and exploring alternative models beyond GBDT could be beneficial. Additionally, one limitation of stacking with the scikit-learn library is the requirement to use the same features for all models involved. Diversifying the feature sets to allow each model to leverage its unique strengths could potentially enhance performance. In Figure 15, we present a comparison between stacking CatBoost and XGBoost versus using CatBoost and XGBoost individually for direct target prediction.

# 5 Conclusion

We explored the problem of predicting stock price variation based on information NASDAQ releases during the last 10 minutes of the trading day. We proposed and tested 3 different approaches and a plethora of models included Prophet, XGBoost, Random Forest and Convolutional Neural Networks.

Ensemble models, such as Gradient Boosted Decision Trees (GBDT), currently stand out as the most promising. Through careful data preprocessing and feature engineering, we achieved gradual accuracy improvements for direct target prediction. Particularly noteworthy is the CatBoost model, enhanced lagged and categorical features and built-in missing values handling, which exhibited the highest accuracy among the models assessed. With an MAE of 5.783, it surpassed the baseline model by 1%. However, we recognize that this approach is suboptimal as it fails to leverage crucial data from all stocks to predict the target. Therefore, we advocate for dismissing this approach.

The two-step approach for target prediction holds the potential to achieve the optimal performance. Consequently, the next steps would be focused on improving the accuracy of predicting the WAP at time t+60. We anticipate that this improvement will lead to significant enhancements in predicting the target variable.

During the execution of this project, we faced several challenges, many of them due to the high size of the dataset (.6 GB dumped on disk) and the lack of enough computational power which many times exhausted our memory and prevent us from working on high-complexity models.

To conclude, the best model we developed outperformed the baseline by approximately 1%. While this improvement might seem modest, it represents a significant achievement in the context of stock market prediction. Moving forward, we are committed to delving deeper into this challenge, utilizing the valuable insights gained to enhance our approach and develop a more robust model.

# References

[1] Kausik Chaudhuri and Yangru Wu. Random walk versus breaking trend in stock prices: Evidence from emerging markets. *Journal of Banking & Finance*, 27(4):75–592, 2003.

[2] Dilek Durusu-Ciftci, M. Serdar Ispir, and Dundar Kok. Do stock markets follow a random walk? new evidence for an old question. *International Review of Economics & Finance*, 64:165–175, 2019.

[3] The Nasdaq Exchange. Closing cross faq. Technical report, Nasdaq, 2024. `https://nasdaqtrader.com/content/ETFs/closing_cross_faqs.pdf`.

[4] Azul Garza and Max Mergenthaler-Canseco. Timegpt-1, 2023.

[5] Optiver. Trading at the close. `https://www.kaggle.com/competitions/optiver-trading-at-the-close/data`. Kaggle Challenge, 2024.

[6] Optiver. Charles' trading at the close introduction. `https://www.kaggle.com/code/charlesndemo/charles-trading-at-the-close-introduction`, 2023.

[7] Optiver. Weights of the synthetic index. `https://www.kaggle.com/competitions/optiver-trading-at-the-close/discussion/442851`, 2024.