Project: Optiver - Trading At The Close

Master's in Electronics and Computer Engineering
University of Porto - Faculty of Engineering
Professor: Jaime S. Cardoso
Andry M. G. Pinto
Students: Antonio Gouveia Ribeiro, *up202302879*
Miguel Gouveia, *up202302221*
Tiago Duarte Correia de Oliveira, *up202005936*

***Abstract*—This project involves diving into the *National Association of Securities Dealers Automated Quotations* (Nasdaq) closing dataset to create a model that can predict the closing price movements of numerous Nasdaq-listed stocks. Four different models will be tested to determine the most suitable one for the time-series forecasting employed. Additionally, a feature engineering method is introduced to enhance the organization of the dataset for more effective model training.**

***Index Terms*—Nasdaq, Closing Dataset, Time-Series Forecasting, Feature Engineering, Four Models**

I. INTRODUCTION

Stock exchanges operate in dynamic, high-pressure settings where time is of the essence. As trading unfolds daily, the intensity escalates notably during the last 10 minutes before the market closes. During this critical period, entities such as Optiver, serving as market makers, integrate conventional data books with action data books. This integration aims to gather insights from both sources to offer optimal prices to all participants in the market. This work aims to construct a model that can forecast the closing price movements of numerous stocks listed on Nasdaq. Real-world data for this endeavor is supplied through Optiver - Trading at the Close.

Nasdaq is a global electronic marketplace for buying and selling securities. It operates 29 markets, one clearinghouse, and five central securities depositories in the United States and Europe, as most of the world's giant technology companies are listed on the Nasdaq [1]. As a dynamic financial marketplace, Nasdaq exhibits distinctive characteristics, particularly during the last ten minutes of the trading day. As the market is coming to a close at the end of the day, traders and investors become more active, as they are trying to finalize their position, as there could be rapid changes in stock prices. Market makers like Optiver play a crucial role when it comes to this crucial period.

Time series is a time-dependent dataset, which means that the values are obtained in specific intervals of time. Usually, the values are taken at regular intervals, but the sampling could be irregular [2]. Time series models use previous values of the main variable to predict future outcomes, utilizing lag values of the target variable as predictors, whereas traditional models use other variables as predictors. Time series analysis involves extracting insights from data and utilizing different techniques. A notable mention of a type from a time series analysis is Time series forecasting.

II. DATASET DESCRIPTION

When building a time-series forecasting model, it's important to choose relevant features that capture the patterns and dynamics of the time series. In relation, the experiment was done solely using target values as the primary feature, eliminating the need for complex feature engineering. This method simplifies the modeling process, enabling a focused analysis of the inherent patterns and trends within the target data [3]. Additionally, the selection of the features ('stock_id', 'date_id', 'seconds_in_bucket') and the selection for the output was 'target'. The features chosen were because of their simplicity for feature engineering and their crucial role in capturing the contextual and temporal dimensions of the time series. As for the output, the 'target' variable serves as the focal point for the model's predictive training.

III. OBJECTIVES

Models that will be introduced to test this Time-Series Forecasting dataset:

- *Linear Regression* (LR)
- *Long-Short Term Memory* (LSTM)
- *K-Nearest Neighbors* (KNN)
- *Random Forest* (RF)

These models will be explained in the upcoming sections and compared at the end, seeing which model performs better in time-series forecasting.

IV. LINEAR REGRESSION

LR is a parametric model and a supervised machine-learning algorithm that analyzes two separate variables, independent and dependent, to define a single relationship. LR assumes a linear relationship and aims to find the best-fit line that minimizes the difference between the true and predicted values.

For LR the dataset was altered because the original dataset implemented that is used for every other model showed the results as 0 so for this model all of the data from the dataset was utilized as features, maintaining 'target' as the output. In addition, the LR approach utilized was through the sklearn library in Python, where the implementation of the train-test-split method was introduced to train the LR model. The dataset was split into 80% training and 20% testing. During training, the model used 80% of the data that was given at random, and validation/testing was performed on 20%. This process helped determine the model's performance to unseen data, to avoid overfitting. Additionally, through parameter tuning, these settings appeared as the best-fit parameters for the model.

As a result of the training process, the following graphs for training and testing were obtained:

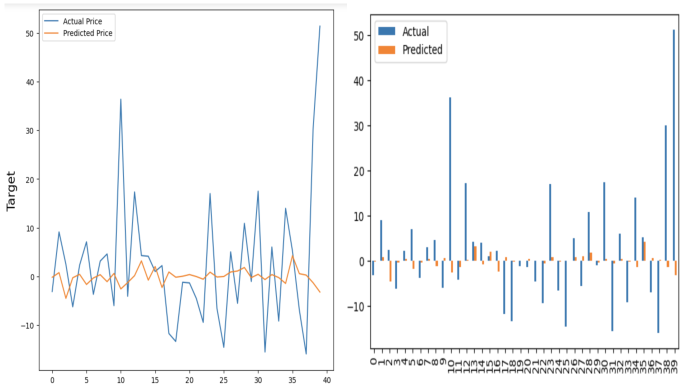

Figure 1: Train dataset results

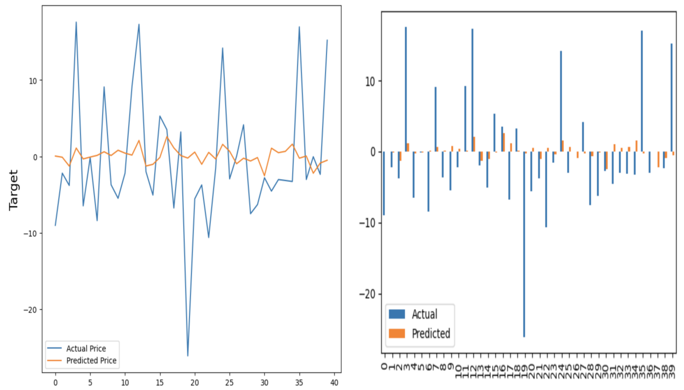

Figure 2: Test dataset results

Corresponding to the succeeding evaluation metrics:

	MAE	MSE	RMSE
Train	6.32	87.24	9.34
Test	6.31	87.24	9.34

Table I: Evaluation metrics for train and test datasets

As can be seen from the training metric table presented above, the obtained results for *Mean Absolute Error* (MAE), *Mean Squared Error* (MSE), and *Root Mean Squared Error* (RMSE) indicate that there was poor performance concerning the model which may signify that the model may be underfitting, but as the values of the training and testing are constant, this shows that the model is not overfitting. Overall, this model does not generalize well to the data provided as can be seen from the graphs in Figure 1 and Figure 2.

V. LONG-SHORT-TERM-MEMORY(LSTM)

LSTM is a parametric and supervised machine learning algorithm that performs well in capturing long-term dependencies and patterns in sequential data. It is a variety of recurrent neural networks (RNNs) that are capable of learning long-term dependencies, especially in sequence prediction problems. LSTM has feedback connections, it is capable of processing the entire sequence of data [4]. LSTM is particularly effective for time series forecasting tasks, as it processes input sequences step by step, updating its internal state and making predictions at each time step. The network learns to capture complex temporal relationships in the data, making it well-suited for applications where the order and timing of events matter.

The LSTM approach was implemented using the 'build_lstm_model' function, using the TensorFlow and Keras libraries in Python. The 'Sequential' model was employed to create a stacked LSTM neural network with multiple layers, utilizing 3 LSTM layers with 50 units each as well as 3 dropout layers to prevent overfitting.

Following the training, the model was evaluated on the testing dataset. Predictions were made for both the training and testing sets, presented in Figure 4 and Figure 5. With this, the training and validation loss over epochs were able to be captured, as presented in Figure 3.

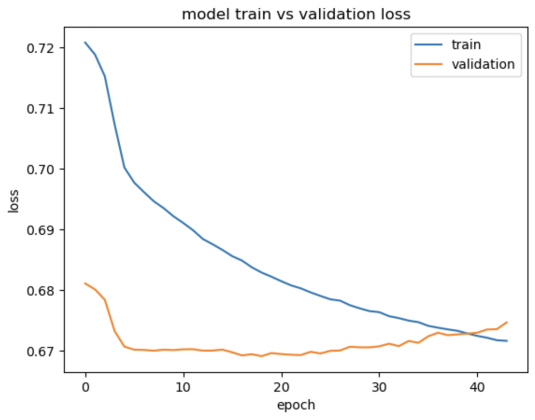

Figure 3: Model Train vs Validation Loss

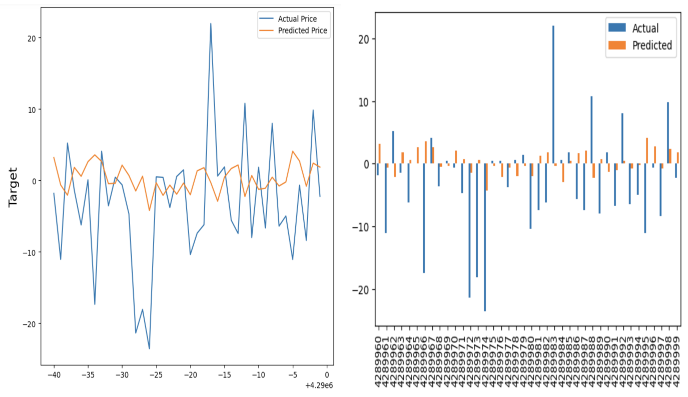

Figure 4: Train dataset results

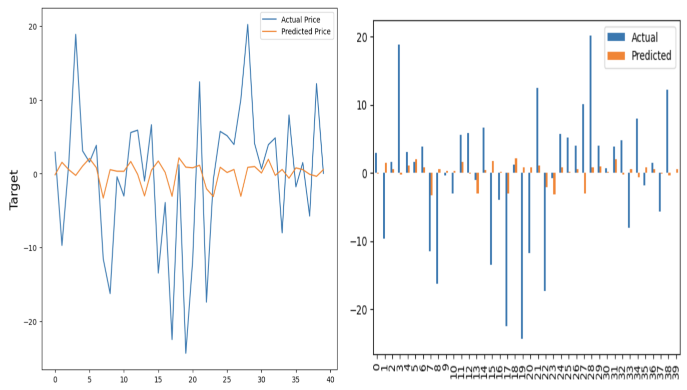

Figure 5: Test dataset results

Corresponding to the succeeding evaluation metrics:

	MAE	MSE	RMSE
Train	6.69	94.99	9.75
Test	5.93	78.54	8.86

Table II: Evaluation metrics for train and test datasets

The presented graphs indicate observable improvements in the performance of the LSTM model when contrasting training and testing data, presented in II. This presentation highlights the performance improvement observed from training to testing. Additionally, the graph depicted in Figure 3 illustrates that the training session finished early due to early stopping. This decision was prompted by the absence of tangible improvements in the model's performance, causing it to stop before reaching the initially planned 100 epochs.

Overall, LSTM effectively captures long-term dependencies in sequential data, making it suitable for tasks where the order and timing of events matter. As can be seen with the hyperparameters that were chosen to train this model it was successful in preventing overfitting from occurring and showing good predictions made on the testing data. Despite the advantages that LSTM brings, a disadvantage was the substantial trial-and-error process required for selecting optimal hyperparameters to achieve the best results in this specific time series forecasting scenario. Regarding this, LSTM showcases promising performance in time series forecasting.

VI. K-Nearest Neighbors (KNN)

KNN is a non-parametric and supervised machine learning algorithm that operates by storing all previous cases and uses that information to predict an end value based on a similarity measure. It employs 'feature similarity' to generate predictions for test data or new data points, assigning values to these by how closely they resemble other training data examples. KNN regression has two distinct strategies. The first consists of calculating the average of the target values of the K-nearest neighbors. The second is by determining an inverse distance weighted average of the K-nearest neighbors. Similar to KNN classification, KNN regression applies the Euclidean, Manhattan, and Minkowski distance functions [5].

The KNN regression approach that we utilized is the first one, through the scikit-learn library in Python.

Concerning the hyperparameter K, a cross-validation method was used to determine the optimal value. The pool of possible K values ranged from five to thirty-five, with increments of five, and the cross-validation split the training data into three partitions. This procedure resulted in an optimal value of thirty-five for K.

Following the training process, the subsequent graphs containing predictions for both training and testing data were obtained:

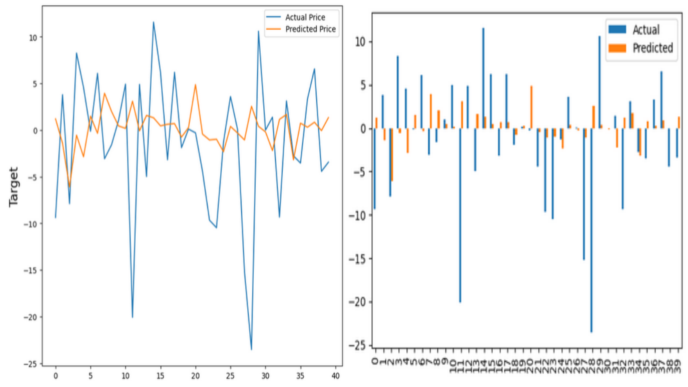

Figure 6: Train dataset results.

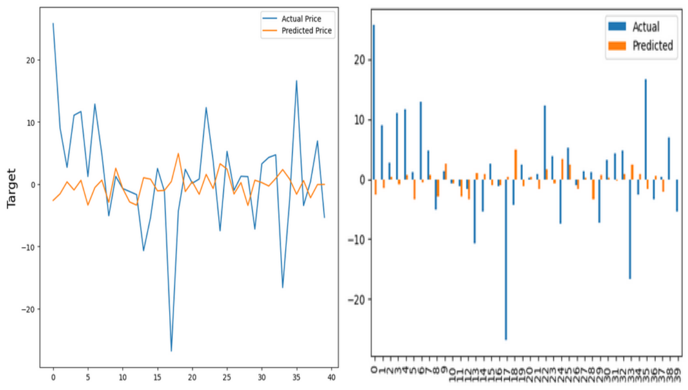

Figure 7: Test dataset results.

Corresponding to the succeeding evaluation metrics:

	MAE	MSE	RMSE
Train	6.29	85.46	9.24
Test	6.47	90.45	9.51

Table III: Evaluation metrics for train and test datasets

Even though it's observed that the training metrics show smaller error values across the different types calculated compared with the test metrics, the model overall presents high error values. This can imply that is an underfitting model, failing to extract the desired generalization by not extrapolating enough patterns and complexities present in the data.

Following that conclusion, it is pertinent to delve into the advantages and disadvantages of this model. KNN is fairly simple to implement. Its non-parametric techniques result in minimal assumptions about the underlying data distribution, emerging as a versatile model to be applied to various types of data. On the opposite side, is sensible to outliers. Since it emphasizes local patterns instead of global trends, the smaller the K value, the more the presence of noise can significantly impact predictions. Notably poor at scaling. As the dataset grows larger the computational costs of predicting new instances become high and the memory usage prominently heavy.

VII. RANDOM FOREST

RF is a Supervised learning algorithm that is based on the ensemble learning method and Many Decision Trees. RF is a Bagging technique, so all calculations are run in parallel and there is no interaction between the Decision Trees when building them.

The algorithm works by having K Decision Trees, N samples in each node, and F features that will be randomly selected for each node of the Decision Tree. The feature used to split the node is picked from one of these F features. With these values defined, the algorithm will create K subsets of the data from the original dataset. K trees are built using only one of the subsets. Each tree is built until there are fewer or equal to N samples in each node. K-trained models form an ensemble and the final result of the Regression task is produced by averaging the predictions of the individual trees [6].

The approach of the grid search cross-validation and the RF Regression was implemented also using the scikit-learn Python library.

To determine the optimal number of trees for the model, K, a grid search cross-validation method was used with a pool of 25, 50, and 75 trees. The optimal value found was for 75 trees. The rest of the parameters use the default values of the scikit-learn RandomForestRegressor class, except for the random_state value, which uses a fixed seed value of 221. It uses a fixed seed value to ensure the reproducibility of the results from training.

The following graphics present the predictions of the model for the training and testing data:

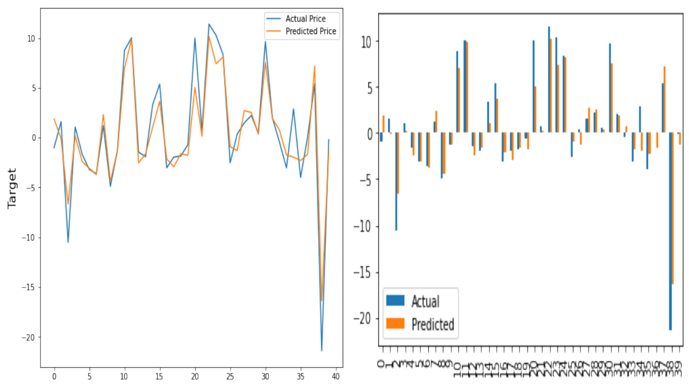

Figure 8: Random Forest Train dataset results.

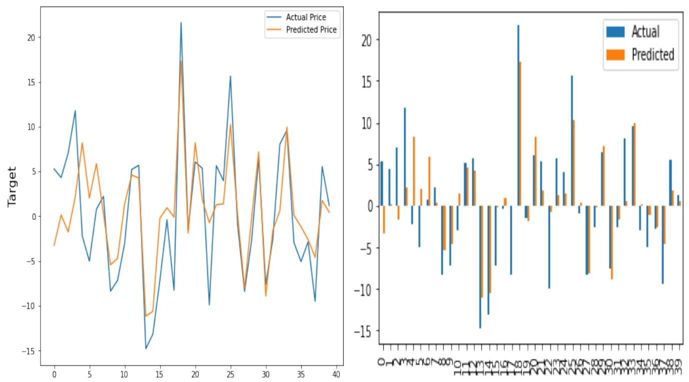

Figure 9: Random Forest Test dataset results.

Corresponding to the succeeding evaluation metrics:

	MAE	MSE	RMSE
Train	1.71	6.55	2.56
Test	4.55	44.57	6.68

Table IV: Evaluation metrics for train and test datasets

The values obtained for the RF Model were the best out of the techniques used for this project. However, the MAE, MSE, and RMSE have a substantial increase from the training to the test data, suggesting the model is suffering from overfitting.

Regarding the advantages of the model: it was easy to implement and handles a wide variety of data types, including numerical and categorical. It can also handle large datasets with high dimensionality, missing values, and outliers and provide information about the importance of each feature in the data. The model handles linear and non-linear relationships very well and doesn't require feature scaling, since it uses rule based approach instead of distance calculation. However, this model is very complex and computationally expensive, especially for cases like this with large datasets. The algorithm can change considerably by small changes in the data as well. Although it is resistant to overfitting, it can still occur in some cases, particularly when working with noisy data, which could be the case in this project.

VIII. CONCLUSION

In conclusion, the models assessed in this project showcased both advantages and disadvantages in the realm of time series forecasting, with varying performances among them. As LR shows it's unfit for this type of dataset, showing the worst results and that it is underfitting, KNN displayed underfitting as well and poor results, LSTM demonstrated that it is one of the more promising ones, having the second best error values, and showing good performance when testing, and RF showing the best error values but having bad performance demonstrating a clear sight of overfitting occurring.

Furthermore, the LSTM model emerged as the most promising, closely approaching the competition benchmark with an MAE of 5.667. The dataset organization played a crucial role in highlighting the relative strengths and weaknesses of each model. While KNN exhibited underfitting and random forest exhibited overfitting issues, as linear regression demonstrated poor and inadequate results.

Despite LSTM proving to be the optimal performer, there remains room for improvement. Exploring alternative models specifically tailored for time series forecasting or further hyperparameter tuning for LSTM could potentially enhance results. It's worth noting that the competitor's model, with different hyperparameters, achieved slightly better performance. This highlights the significance of ongoing improvement and experimentation to achieve time series forecasting models that are both robust and accurate.

IX. REFERENCE

REFERENCES

[1] A. HAYES, "What nasdaq is, history, and financial performance," Available at https://www.investopedia.com/terms/n/nasdaq.asp, 2023.

[2] A. M. Khulood Albeladi, Bassam Zafar, "Time series forecasting using lstm and arima," vol. 14, no. 1, pp. 1–8, 2023.

[3] J. A. RODRIGUEZ, "Lstm with simplified feature engineering," Available at https://www.kaggle.com/code/jorgearreserodriguez/lstm-with-simplified-feature-engineering#LSTM-with-Simplified-Feature-Engineering, 2023.

[4] Intellipaat, "What is lstm? introduction to long short term memory," Available at https://intellipaat.com/blog/what-is-lstm, 2023.

[5] S. Kohli, G. T. Godwin, and S. Urolagin, "Sales prediction using linear and knn regression," Singapore, pp. 321–329, 2021.

[6] V. Lyashenko, "How to use random forest for regression: notebook, examples and documentation," Available at https://cnvrg.io/random-forest-regression/.