# 10-minute Trading Window Stock Volatility Prediction with Neural Network and Gradient Boosting methods

Selene Bao
2021

## Contents

# 1 Abstract

In recent years machine learning methods have been more widely used in the financial modeling of capital markets. This paper explores four machine learning tools, LSTM Recurrent Network, FeedForward Neural Network, Light Gradient Boosting, and Extreme Gradient Boosting, as applied to financial realized volatility forecasting. We train the above algorithms on over 100 stocks' trading window of 10 minutes realized volatility and project the next trading interval's volatility. By conducting a time series validation, we find that tree-based models are overall outperforming the neural network models given the heterogeneity within the data features.

# 2 Keywords

Quant trading, realized volatility, realized correlation, high-frequency data, bid and ask, trade volume, trading window, time series, clustering, long short-term memory, recurrent neural network, feedforward neural network, light gradient boosting, weighted extreme gradient boosting, permutation importance.

# 3 Introduction

## 3.1 Background

In the context of economic globalization with the impact of COVID-19 and other crucial issues such as the Trade War, the capital markets, especially the stock market, have experienced unprecedented fluctuations in these three years. Given the importance of return volatility on making effective investment decisions, real-time estimates and forecasts of stock volatility become particularly significant and challenging.

Volatility, one of the most critical characteristics of financial markets, acts as a double-edged sword in the tradeoff between return and risk. Financial institutions and researchers have been making extensive efforts to accurately measure current and future volatility to reduce uncertainty within a controllable level. Unlike the raw return of a financial instrument, realizations of return volatility can not be directly observed or simply calculated. Realized volatility measures the annualized standard deviation in the daily price return of an underlying asset over a given period. When we trade, a valuable input to make decisions is the standard deviation of the stock log returns. The standard deviation will be different for log returns computed over longer or shorter intervals.

In this project, we explored a replicated stock market with a mindset of short-term quant trading, with 112 stocks with 3830 ticks within 10 minutes of book and trade data to predict the realized volatility in a certain trading window. A tick here refers to the change in the price of a financial security from one trade interval to the next 10-minute interval. There are many methods in the standard machine learning tool and deep learning including, regression, SVM, decision tree, random forecast, neural network which can be used to illustrate the realized volatility projection. In this pa-

per, we are going to focus on comprehensive models related to neural networks and ensemble gradient boosting models. Clustering tools and Time Series Analysis are incorporated into our models to reflect the data characteristics of high-frequency and time-stamped. Comparison among all models are also included.

## 3.2 Motivation

We have seen the ups and downs in the stock market. If we need to have some image of what's happening in the stock market today, what we can picture is different algorithms fighting each other with similar or entangled designs, which comprise 70 percent of the United States stock market. As we have always heard that Professionals try to harness the spikes and slumps, but most investors should stick with diversification. Hence, we would like to zoom in and analyze the shorter-term environment with the newly developed technology of machine learning in recent years, trying to figure out what might be the driving force of the volatility within minutes.

## 3.3 Related Work

Given the importance of return volatility on the practical financial investment decision-making, extensive efforts have been made to provide real-time estimate and forcasts for the volatility in the capital market. Anderson et. al(2001) studied frameworks in intergrating of high-frequency of daily trading data in exchange currency market into the measurement, modeling, and forecast process. Quadratic variation are introduced in their paper as an important index to measure realized volatility and methods of vetor autoregression and memory-based neural network are implemented. Luong(2018) analyzed realized volatility forecast for financial time series data with 15-s trading windows in Australia Stock Market and considered the inclusion of purified implied volatility as a factor in the modeling of heterogeneous autoregressive model (HAR). Audrino(2008) presented a new semi-parametric model for the prediction of implied volatility with boosting regression tree and cross validation.

We are inspired by the idea of portfolio construction that a multivariate timewise series would need a stochastic volatility model, but were not confident when considering problems of auto-dependency and cross-dependency. Previous models proposed by Bollerslev, Engle et. al (1990) in the time-series framework of the GARCH model might not be suitable for this case, although it could deal with heteroskedasticity. We have discovered a plausible idea given by Yang et. al (2021) predicting stock price based on LGBM and XGBoost, with their focus on high frequency trading. We consider LGBM as it optimizes features by including "Leaf-Wise based decision tree growth strategy, optimal segmentation of category eigenvalues, feature parallelism and data parallelism". To implement Gradient boosting, we would create statistics for different time ids, seconds in buckets besides all return and price information, as we have just discussed earlier. Also, We also get some motivations from Buhlmann (2007), whose paper is focused on designing advanced gradient boosting methods from the statistical perspective.

# 4 Data

## 4.1 Data Source

This project focuses on the forecast of short-term stock realized volatility and the raw data is collected from Optiver and Bloomberg with over a hundred stock's information on stock id, time id, and trading order book. The order book here refers to an electronic list of buy and sell orders for a financial security or instrument organized by price level. A typical stock order book contains the number of shares being bid on or offered at each price level. To execute a trade, it is important to find sellers and buyers with counter-interest to trade with based on information in the order book. The overall size of the data includes 112 stocks with 3830 ticks of 10-minute trading windows for each stock. Within each trading window, seconds-based information are provided in the order book. The data formats in our project include csv files and parquet files: train and test datasets are in csv format and all order books and trade data are recorded under parquet files. In the csv training dataset, we have 428,960 observations with 3 attributes - **stock id**, **time id**, and **target**, which refers to the realized volatility for the certain time point of a specific stock.

## 4.2 Variable Description

Target: In this project, target means realized volatility computed following the same stock id and time id, which leads to zero overlapping between the target and the volatility computed from the original dataset. Realized volatility, in financial term, is simply the $\sigma$ standard deviation , calculated as the square root of the sum of squared log returns:

$$\sigma = \sqrt{\sum_t tr^2_{t-1,t}} \text{ where } r_{t-1} = \log\left(\frac{s_{t-1}}{s_{t_1}}\right)$$

Stock returns: Returns can be calculated in various ways. In this project, we would take the stock price into log returns, a common practice of performance measure. Calling St the price of the stock S at time t, the log return between time $t_1$ and $t_2$ is defined as follows:

$$r_{t_1,t_2} = \log\left(\frac{S_{t_2}}{S_{t_1}}\right)$$

Based on the fixed time interval of 10 minutes, a 10-minute log return will be denoted as:

$$r_t = r_{t-10\,\min,t}$$

Stock Price: Since trades are happening every second during the trading window, we will first model out a relatively optimal way to measure the stock price such as weighted average price based on bid and ask information during the time window.

## 4.3 Data Preprocessing

Based on the fact that the raw data are with high frequency, we preprocess the data with second-based intervals within each trading window. This gives us more features to get considered when we implement the time series analysis. Regarding information in order book, several variables are created based on time windows including: bid-ask spread, trading volume, trading size, weighted average price, price spread, log return, etc.

As mentioned above, we are utilizing log return as the main input to calculate the realized volatility. When addressing log returns, we have to decide the prices within the trading window, here, we introduce the Weighted Average Price(WAP) into our model, the WAP are generally calculated as follows:

$$WAP = \frac{\text{BidPrice}_1 * \text{AskSize}_1 + \text{AskPrice}_1 * \text{BidSize}_1}{\text{BidSize}_1 + \text{AskSize}_1}$$

If two order books have both bid and ask offers with the same price respectively, the one with more offers will generate a lower stock valuation since there are more sellers intend to sell the stocks, indicating a higher supply in the market, which directly influences the stock price volatility.

Train-Test Split are conducted with a 8-2 split: For each stock, 20% of the trading window data will be used by time order as the test dataset for the algorithms.

## 4.4 Exploratory Data Analysis

The distribution of the target volatility has a log-normal distribution (right-skewed 2.827), with extra kurtosis compared to the normal stock market in the long-run. Outlier or high leveraging points are worth noticing from the right plot. Those extreme values can be spotted in the upper right corner. We are not going to remove the outliers in our dataset given that we are dealing with stock dataset, which might have observations deviate from other observations a lot.

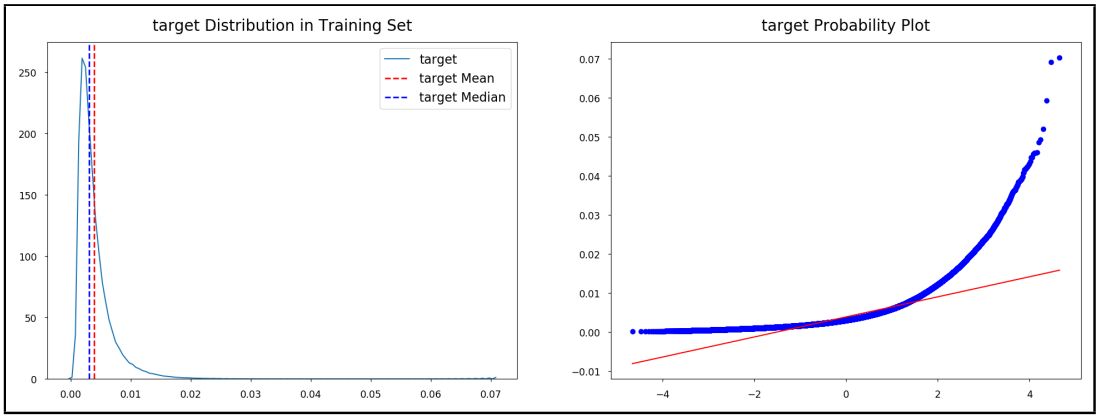


Figure 1: Distribution of Realized Volatility

Volatility reflects a combination of information, such as industrial financial status, organic growth of the institution, and bid-ask power. The most volatile stock does not secure a higher return on certain strategies. As shown in Fig. 2, wider chunks in each stock indicate a single stock could be fluctuating at a certain time and being deadly silent otherwise. Temporal relations in the market normally are hidden and granger causal.

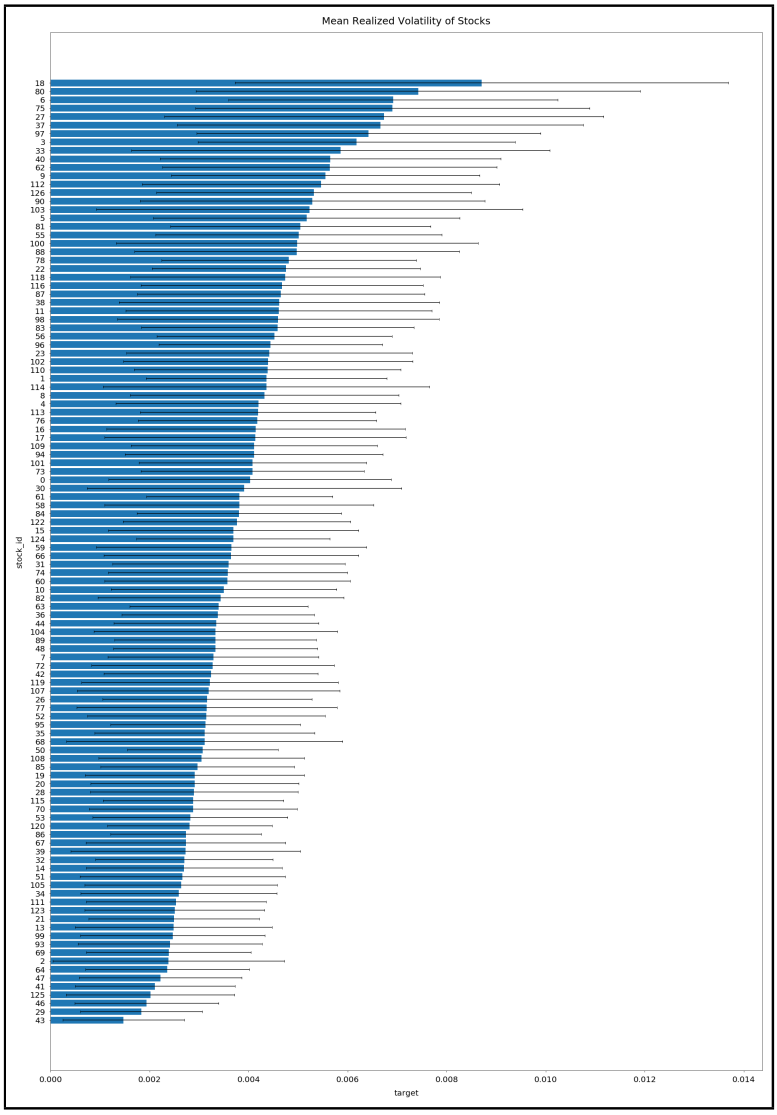


Figure 2: Realized Volatility Rank

That being said, the top volatile and stable stocks at a certain time id are illustrated in Fig.3 and Fig.4. We would pay much more attention to those stocks under some condition since they might be influenced by the same event, a time id that renders stocks to spike might be the time the FED makes announcements. Stock 77 at time 24600 ranks the highest volatility of all stocks at all time. The most volatile single stock, however, is Stock 18, with three top-10 positions on the list.

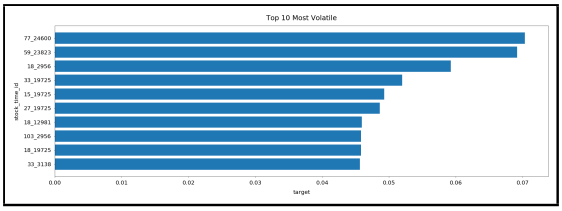


Figure 3: Most Volatile Stocks

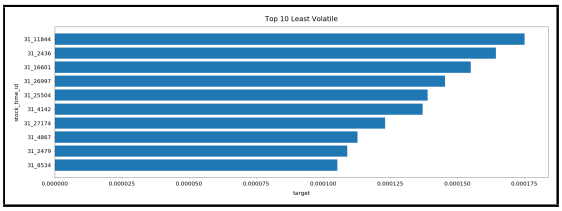


Figure 4: Least Volatile Stocks

Divergence is analyzed in Fig.5 and 6, two stocks are randomly chosen from the set of 112 stocks. The random two stocks behave divergently even within such a short time window, there seems to be no common shock in terms of return shapes and bid-ask price change.

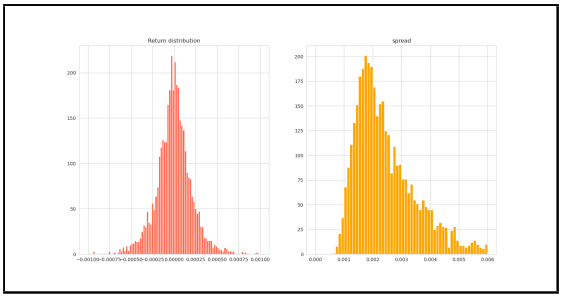


Figure 5: Stock 18 Statistics

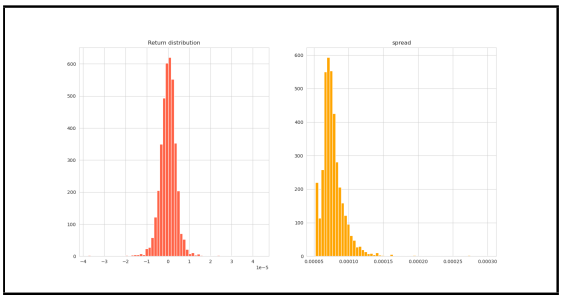


Figure 6: Stock 43 Statistics

Realized volatility for the current window will be construed as the baseline for modeling comparison. By taking a closer look at stock 77 at the top of target volatility, we could

witness that the volatility at time 264.00 is much greater than that of all times, showing that one stock could be extremely volatile in one time and stable in another. That could pose an obstacle in modeling since we might not be able to tell whether the current realized volatility would cause underestimation or this time point could be an outlier instead. The decrease in the bid-ask price chart is a natural way of illustrating that the size of bid and ask is a dominating factor of price moving. When I was a trader, I tended to sell chunks of shares at the most competitive price, hence making time id a good indicator of bid-ask power by frequency of updates.

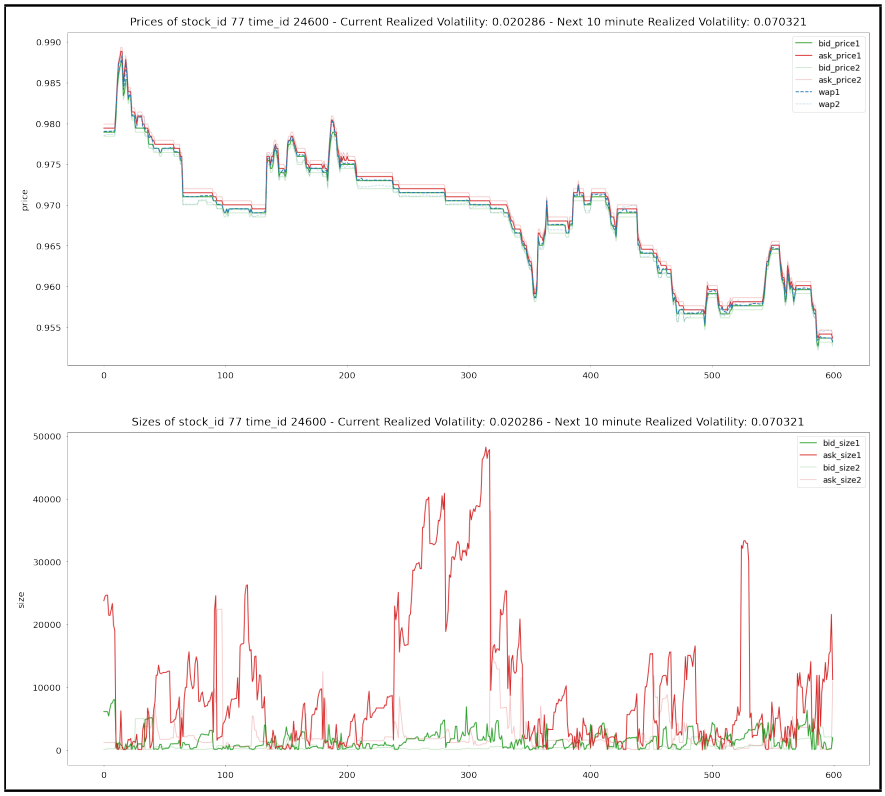


Figure 7: Zoom into one stock's trading window

Considering the potential inconsistency with target and stock price, we would need to cast a glance at all 112 stocks. The relation between target and bid-ask price ratio is not that strong, with an extreme skewed standard deviation. We will explore more combinations with time id in the modeling session.

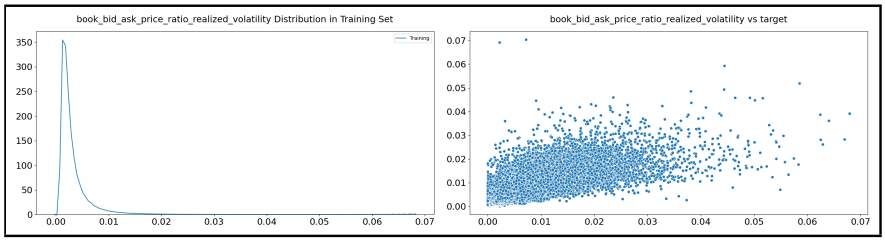


Figure 8: Distribution of Volatility based on Bid-Ask Price Ratio

# 5 Methodology

In this section, we are going to illustrate the main methodology we used in our project as well as some advanced modification to the traditional model. Given our data in stock market with timeliness, we first consider some neural networks for time series analysis and feature selection models. Gradient boosting methods are later introduced.

## 5.1 LSTM Recurrent Neural Network

Recurrent Neural Network has been widely used in time series analysis with predictions being dependent on previous predictions. Both inputs and stage outputs will be recurrently transformed as the new input sequence in the model. However, the learning process in standard RNN are highly dependent on memory and only small number of discrete time steps can be involved in the relevant input events and target outputs. In other words, when we are having long-range sequences, Standard RNNs are not able to store all previous inputs and the vanishing gradient problem will occur. The problem is illustrated in the following figure from Alex Graves' thesis. The shades of the nodes indicate the sensitivity of the network nodes to the input at a given time qindow. When the shade is darker, the input will be more sensitive. However we see that from time step 1 to 7, the sensitivity value is decaying deeply, which means that the network at time step 7 has already forgotten the first time step's input. When training the RNNs, the adjusted weights will be repeated to affect the gradient to the direction of too large or too small.

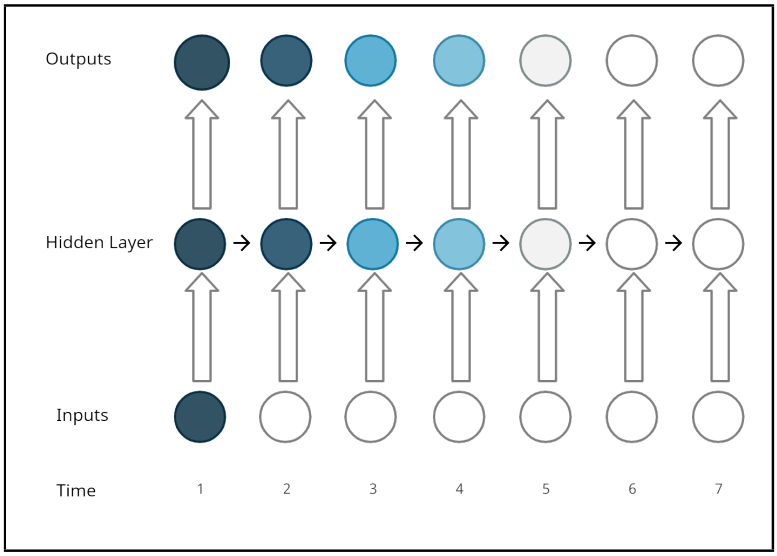


Figure 9: Standard RNN - vanishing gradient

Generally , RNNs will forget the input after 5 to 10 discrete time steps[10]. To slove the vanishing gradient problem, LSTM(Long Short-Term Memory) is introduced with the feature that it can "learn to bridge minimal time lags in excess of 1000 discrete time steps by enforcing constant error flow through "constant error carrousels" (CECs)

within special units, called cells." [10] A simple one timestep LSTM is illustrate below:

$$
\begin{aligned}
f_t &= \sigma_g\left(W_f \times x_t + U_f \times h_{t-1} + b_f\right) \\
o_t &= \sigma_g\left(W_o \times x_t + U_o \times h_{t-1} + b_o\right) \\
i_t &= \sigma_g\left(W_i \times x_t + U_i \times h_{t-1} + b_i\right) \\
c'_t &= \sigma_c\left(W_c \times x_t + U_c \times h_{t-1} + b_c\right) \\
c_t &= f_t \cdot c_{t-1} + i_t \cdot c'_t \\
h_t &= o_t \cdot \sigma_c\left(c_t\right)
\end{aligned}
$$

where, $\sigma_g$ is the sigmoid, $\sigma_c$ is the activation function of tanh,$f_t$ is the forget gate, $i_t$ is the input gate, $o_t$ is the output gate, $c_t$ is the cell state, $h_t$ is the hidden state, and $W_{f,o,i,c}$ are the weight matrices. Here, LSTM observes the inputs from the previous timestep LSTM and generates the memory of cell state and hidden state, which can be used for the next stage. The whole model can be demonstrated in the below image[11]:

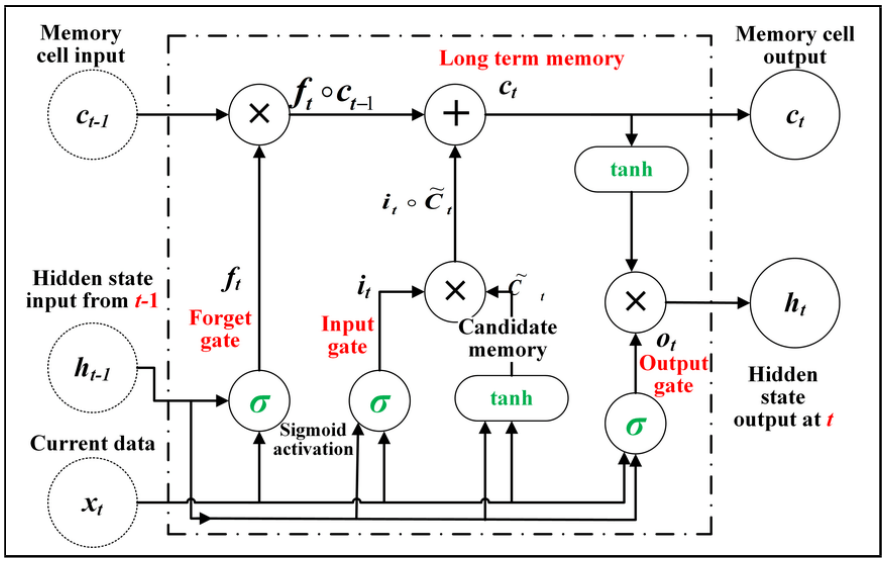


Figure 10: LSTM Neural Network Structure[11]

Using a series of gates to train and store each node with its own Recurrent Network, the LSTM is able to keep, forget or ignore data points based on a probabilistic model. In our project, we are going to set the number of hidden layers and the time step $t$ to experiment towards the prediction.

## 5.2 FeedForward Neural Network

Different from Recurrent Neural Network, FeedForward Neural Network does not feed back the outputs of the model from the previous state to the next one.

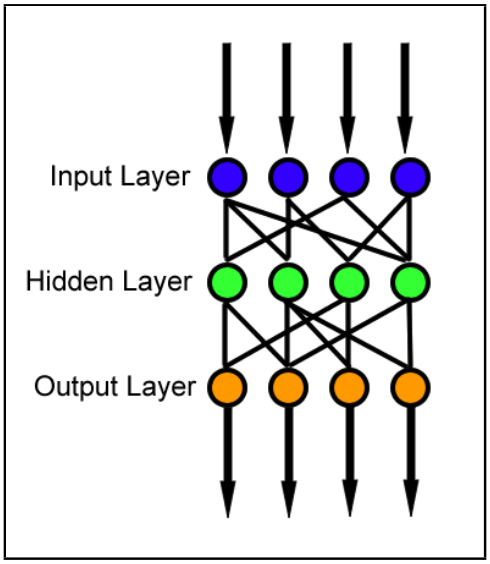


Figure 11: FeedForward Neural Network Structure[12]

In FFNN, we generally illustrate functions to map a set of variables and features to outputs under $y = f(x;\theta)$ with the learning of the parameters $\theta$ to utilize the best function approximation.

Given the fact that stocks in equity market can be highly correlated with each other, espeically when some stocks are within the same sector. We implement K-Means clustering method when choosing the features to be included in the FFNN model to demonstrate some possible relationship that might affect the prediction of the realized volatility of certain stocks. **Pearson Correlation** is used here as the distance measurement in the K-Means Clustering:

$$d_{cor}(x,y) = 1 - \frac{\sum_{i=1}^{n}(x_i - \bar{x})(y_i - \bar{y})}{\sqrt{\sum_{i=1}^{n}(x_i - \bar{x})^2 \sum_{i=1}^{n}(y_i - \bar{y})^2}}$$

## 5.3 Gradient Boosting

We implemented two methods within the category of ensemble gradient boosting: Light Gradient Boosting and Extreme Gradient Boosting.

Both LightGBM and xgboost are a hill-climbing algorithms. They are similar in the way of splitting trees and selecting optimal segmentation points, which is actually a search process.LightGBM reduces complexity of histogram building by down sampling data using one side sampling and bundling features together, basically retaining instances with large gradients while randomly sampling on instances with small gradients, which makes it fast. The process of the algorithm could be demonstrated as the following steps:

Let $f_0(x) = \arg\min_c \sum_{i=1}^{N} L(y_i, c)$, For t = 1 to T: we calculate pseudo residuals

$$g_{ti} = \frac{\partial L(y_i, f_{t-1}(x_i))}{\partial f_{t-1}(x_i)}$$

and fit decision tree and compute the step multiplier for each leaf and update. $F_t(x) = F_{t-1}(x) + \lambda_t * \gamma_t h_t(x)$ where $\lambda_t$ is the learning rate for iteration $t$, a detailed derivation mathmatically can be found in LightGBM: A Highly Efficient Gradient Boosting Decision Tree[9]
For XGBoost, the initialization is the same with lgbm that

$$f_0(x) = \arg\min_c \sum_{i=1}^{N} L(y_i, c)$$

For setting m trees, M = 1,2,. . . , M: For n samples i = 1,2,. . . , the first derivative is calculated as $t = \sum_{i=1}^{N} g_{Ti}$ and the second derivative is $H_t = \sum_{i=1}^{N} h_{Ti}$ and the second derivative is $H_t = \sum_{i=1}^{N} h_{Ti}$, where:

$$g_{ti} = \frac{\partial L(y_i, f_{t-1}(x_i))}{\partial f_{t-1}(x_i)}, h_{ti} = \frac{\partial^2 L(y_i, f_{t-1}(x_i))}{\partial {f_{t-1}}^2(x_i)}$$

The optimal J = 1,2,. . . , J region (characteristic) and region J threshold are solved according to $G_{tj} = \sum_{xi \in R_{tj}} g_{ti}, H_{tj} = \sum_{xi \in R_{tj}} h_{Ti}$, to solve the optimal solution of leaf node: $Omega_{tj} = -\frac{G_{tj}}{H_{TJ} + lambda}$.

The major differnce is that, instead of assigning a weak classifier used to fit the negative gradient, XGB uses the optimization function including penalty of complexity. The objective function on which the search to minimize is $Obj = Loss + \omega$, where Omega is to penalize on the complexity of the model, as demonstrated above. This complexity is not the complexity of cart in a certain lesson, but the total complexity of all carts in XGB. It is conceivable that for each additional cart, the complexity will increase his punishment. When the loss decreases less than the complexity increases, XGB stops.

# 6 Experiments and Results

## 6.1 Error Measurement: RMSPE

Given the small scale of realized volatility, we use the RMSPE(Root Mean Square Percentage Error) as the error measurement of the models:

$$\text{RMSPE} = \sqrt{\frac{1}{n} \sum_{i=1}^{n} \left( \frac{y_i - \hat{y}_i}{y_i} \right)^2}$$

where $y_i$ is the observed realized volatility and $\hat{y}_i$ is the predicted volatility of the stocks in the time window.

## 6.2 LSTM Recurrent Neural Network

We implement several time windows for LSTM model as the time step being trained in the encoder layer and utimately found that a time window of 30 performs best to predict the realized volatility.

| Time window | RMSPE |
|---|---|
| 10 | 0.43 |
| 20 | 0.40 |
| 30 | 0.38 |
| 50 | 0.39 |
| 100 | 0.42 |

Table 1: Performance across time windows in LSTM

Through experiments, we also found that although LSTM overall did a quite good job in predicting the stock realized volatility, it performs poorly in predicting a certain abnormal volatility, which is reasonable in common sense. Since we are solely dependent on previous data in the time series analysis to project the future time's volatility, it is common that the algorithm will fail to capture some abnormal or extreme events that happen in the future. Below is an example of prediction of stock 0' s volatility based on a 30 time step in LSTM:

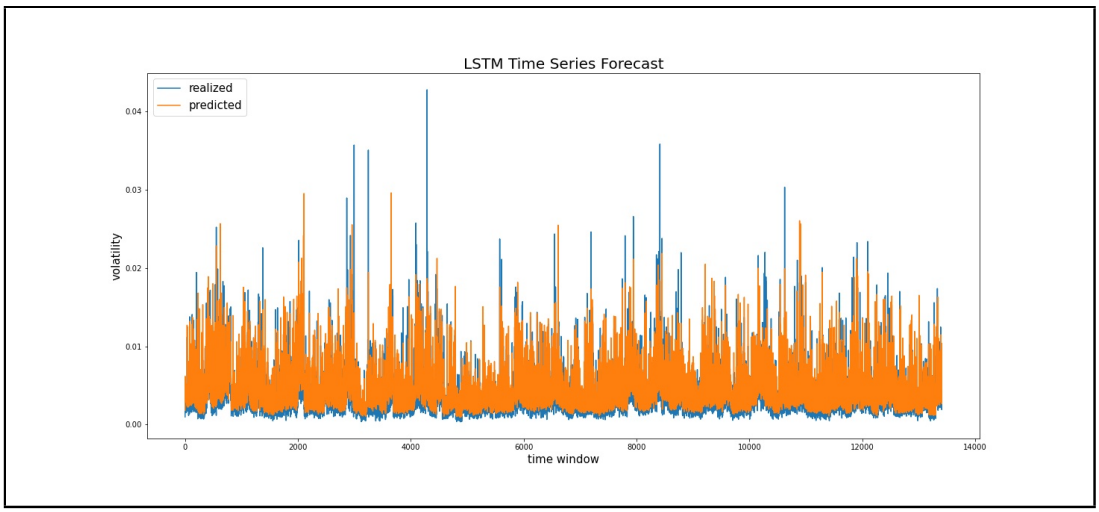


Figure 12: LSTM Volatility Prediction on Stock id of 0

Another factor we need to consider is that the algorithm training process for one stock might not work well for another stock, which can affect the overall RMSPE a lot. This is reasonable as well since not all stock will have same performance in volatility across timeline. Espeically when we are evaluating two stocks in two sectors that might hedge against each other, the overall data trend will be extremely different.

## 6.3 FeedForward Neural Network

By implementing K-Means Clustering for stocks based on the correlation between stock's volatility across timeline, we generate five groups in Table 2 and we can infer that stocks in the same cluster behave in similar way should be in the same stock sector in equity market.

| **Cluster** | 1 | 2 | 3 | 4 | 5 |
|---|---|---|---|---|---|
| **Stock IDs** | 4, 5, 10, 16, 22, 23, 26, 29, 36, 44, 48, 60, 66, 69, 72, 73, 87, 94, 95, 102, 109, 112, 113, 115, 116, 120, 122 | 3, 6, 9, 18, 61, 63, 8, 80, 81 | 0, 2, 7, 13, 14, 15, 17, 19, 20, 28, 30, 32, 34, 35, 39, 41, 42, 43, 46, 47, 51, 52, 53, 64, 67, 68, 70, 85, 93, 100, 103, 104, 105, 107, 108, 114, 118, 119, 123, 125 | 21, 27, 31, 33, 37, 38, 40, 58, 59, 74, 77, 82, 88, 98, 99, 110, 111 | 1, 11, 50, 55, 56, 62, 75, 76, 78, 83, 84, 86, 89, 90, 96, 97, 101, 124, 126 |

Table 2: K-Means Cluster in FFNN

After implementing clustering method, we add several features from stocks in different groups as input features in our FFNN. Regarding feature of second-based trading window, we choose 150s-trading windows as primary feature and divided the 10 minute trading interval into 4 features. When conducting the training process, we use permutation importance test to decide features being included in our final model and around 50 features are eventually left in our FFNN model on log return, time id, trading volume, trading size, price spread, bid-ask spread, etc.

The graph below shows the result for 112 stock's volatility prediction on test dataset across time. We can see that the problem of not successfully predicting abnormal volatility still exists. However, since we consider the correlation of stocks in same sector into features, it is more accurate to use within-group feature to predict those stocks have more similar previous volatility and deviates those with different behaviors.

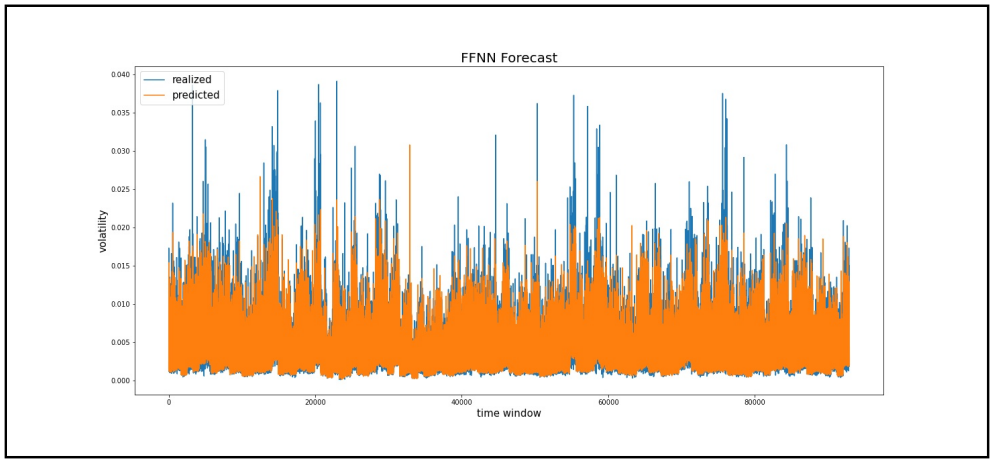


Figure 13: FFNN Volatility Prediction

## 6.4 Light Gradient Boosting

Gradient boosting is basically a hill-climbing algorithm that is fit using any arbitrary differentiable loss function and gradient descent optimization algorithm. It chooses the maximum delta value to grow and uses histogram-based algorithms. This gives the technique its name, "gradient boosting," as the loss gradient is minimized as the model is fit. In this case, the loss function is RMSPE.

The baseline model was trained with the following hyperparameters: number of leaf: 50, learning rate:0.1, feature fraction:0.8, bagging 0.8, boost round 10000, early stopping point 50, the evaluation function is RMSPE. With an 8/2 train-test time-window, The best output showed an RMSPE of 0.232. From the results, we found that it did capture most of the spikes but at the same time those spikes tend to amplify the predicted volatility through the following several seconds when environments are very unstable. That might be due to the nature that the model has got relatively large losses from the spikes and then was cascading downward when splitting the tree leaf-wise further. It has improved a lot by capturing the pattern compared to former techniques, while we would propose a fine-tune to tackle the problem of high variance and heteroskedasticity.

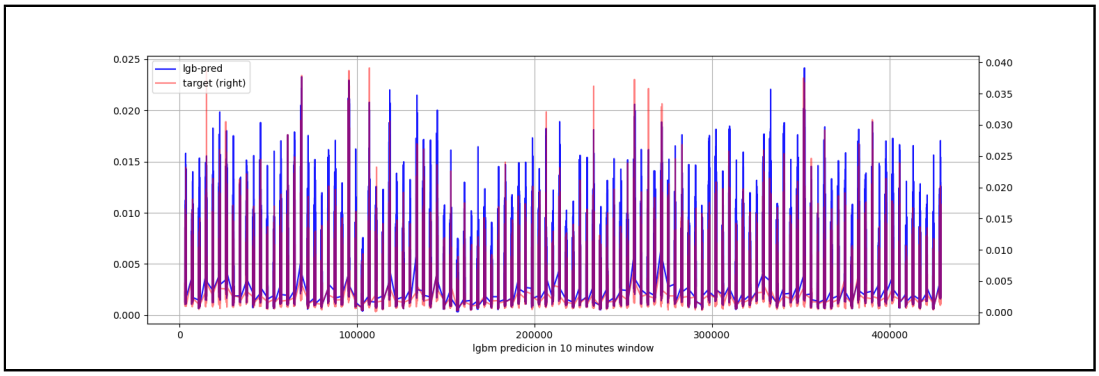


Figure 14: Light Gradient Boosting Prediction

## 6.5 Extreme Gradient Boosting with groups

This algorithm is an improved version of the Gradient Boosting Algorithm. The base algorithm is Gradient Boosting Decision Tree Algorithm. Its powerful predictive power and easy-to-implement approach has made it float throughout many machine learning notebooks. It does not build the full tree structure but builds it greedily. As compared to LightGBM, it splits level-wise rather than leaf-wise.

In Gradient Boosting, negative gradients are taken into account to optimize the loss function but here Taylor's expansion is taken into account. The regularization term penalizes complexity. We have grouped stocks into 28 different ones to deal with heterogeneity. More importantly, we have applied weights, taking the inverse of the square root of correspondent y as a scaling anchor, to tackle the amplification problem of high leveraging spikes to the grouped model. This could be effectively observed

in the figure below, compared to the light GBM baseline in the above section, Those tuning has improved our RMSPE score future to 0.2070.

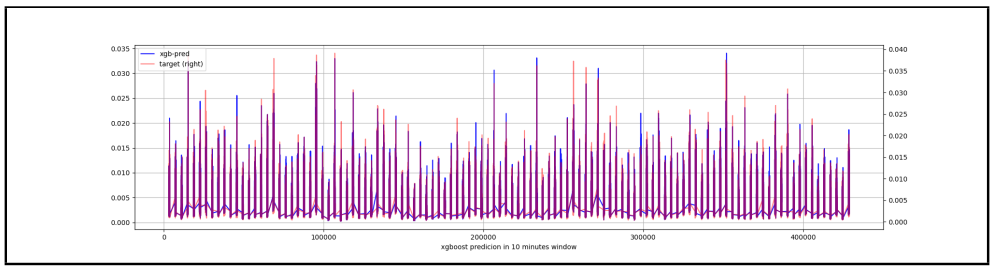


Figure 15: Extreme Gradient Boosting Prediction

## 6.6 Models Performance Comparison

Table 3 summarizes the performance comparison among four models. Looking into the overall results across models, we can find that LSTM, being highly dependent on time-series data and choice of time step in input layer, performs the worst among all models. FFNN and Light Gradient Boosting are having similar performance with one focusing more on neural network generating with clusters and one focusing on ensemble of tree model. Extreme Gradient Boosting is performing the best, which meets our expectation when we were modeling. Both in the context of implementation difficulty and interpretability, XGBoosting can overall outperforms neural network models since NN models usually need to address lots of questions on activitation function, loss function, layer amount selection, and feature slection. Further, when we have heterogeneous features in the dataset, which is true in our dataset, tree-based methods such as XGBoosting can treat features independently of each other and create the rules based on the values for those features, so these models are generally perform better than deep learning models such as FFNN.

| **Model** | **RMSPE** |
|---|---|
| LSTM | 0.38 |
| FFNN | 0.23 |
| LightGB | 0.23 |
| XGBoost | 0.21 |

Table 3: Model RMSPE Comparison

# 7 Conclusion and Future Work

The intraday data for all stocks within the 10-minutes' window are permitted by the hedge fund who provided the data and encrypted them by string ids. Our models have captured the majority of the volatility solely on prices and trade/order book records, which indicated that the rule of demand-supply has a dominant effect on the spot volatility. Nonetheless, despite that we have improved our model taking sampling weights and sector-wise differences into consideration, we would have to ask for

more information other than trades to figure out some of the momentum of spikes and slumps, as timing is crucial to HFTs. Because the algorithms nowadays are dependent on one quality above all else, which is speed. As we have seen in our data and models, the volatility expected or predicted would be a guideline of the strategy for algorithms to execute trades. Short-term strategies are diametrically opposite from traditional long-term, buy-and-hold investing, since the arbitrage and market-making activities generally occur within a small time window, before the price discrepancies or mismatches disappear. Expecting to collect more data with comprehensive information in stock market, which can reflect the real-time impact at a higher level, is still hard to reach.

There is another phenomenon that was observed in the process of modeling and worth mentioning. There seems to be a chain effect of bid and ask, which probably means that there is a noticeable portion of algorithms trading frequently in the market are designed under a similar mindset, which tends to trigger faster execution every time there is a big sell/buy order hitting the market, leading to an aftershock for our high-leveraging data points, just like an earthquake.

We would have to be careful, as investors, since the math that we use to break up the big thing into a million little things can be used to find a million little things and sew them back together. And if we were to replay the same sequence of the events identically, there is no guarantee that it would cause a similar pattern.

For further studies, factors related to market confidentiality and real-time reflection, along with market demand and supply, will pertain to the analysis. Regarding methodology, hybrid modeling with tree models and networks might be implemented in a comprehensive way.

# 8 References


[1] T. Andersen, T. Bollerslev, F. Diebold, and P. Labys, "Modeling and forecasting realized volatility," 2001.

[2] P. Bühlmann and T. Hothorn, "Boosting algorithms: Regularization, prediction and model fitting," *Statistical Science,* vol. 22, no. 4, 2007.

[3] B. D. LeBaron, "Forecasting realized volatility with kernel ridge regression," *SSRN Electronic Journal,* 2018.

[4] F. Audrino and D. Colangelo, "Semi-parametric forecasts of the implied volatility surface using regression trees," *Statistics and Computing,* vol. 20, no. 4, pp. 421–434, 2009.

[5] C. Luong and N. Dokuchaev, "Forecasting of realised volatility with the random forests algorithm," *Journal of Risk and Financial Management,* vol. 11, no. 4, p. 61, 2018.

[6] R. Bhowmik and S. Wang, "Stock market volatility and return analysis: A systematic literature review," *Entropy,* vol. 22, no. 5, p. 522, 2020.

[7] G. Zhang and G. Qiao, "Out-of-sample realized volatility forecasting: Does the support vector regression compete combination methods," *Applied Economics,* vol. 53, no. 19, pp. 2192–2205, 2021.

[8] Y. Yang, Y. Wu, P. Wang, and X. Jiali, "Stock price prediction based on XGBoost and lightgbm," *E3S Web of Conferences,* vol. 275, p. 01040, 2021.

[9] K. Guolin, M.Qi, F. Thomas, W. Taifeng, C. Wei, M. Weidong, Y. Qiwei, L. Tie-Yan, "LightGBM: A Highly Efficient Gradient Boosting Decision Tree," *Advances in Neural Information Processing Systems,* vol. 30, 2017

[10] F. A. Gers, "Learning to forget: Continual prediction with LSTM," *9th International Conference on Artificial Neural Networks: ICANN '99,* 1999.

[11] T. Zebin, M. Sperrin, N. Peek, and A. J. Casson, "Human activity recognition from inertial sensor time-series using batch normalized deep LSTM recurrent networks," *2018 40th Annual International Conference of the IEEE Engineering in Medicine and Biology Society (EMBC),* 2018.

[12] "Feedforward Neural Network," *Wikipedia,* 05-Dec-2021.

[13] P. Buhlmann and T. Hothorn, "Boosting algorithms: Regularization, prediction and model fitting," *Statistical Science,* vol. 22, no. 4, 2007.