Stock Market Volatility Prediction

Sai Sahana Bhargavi Byrapu sbyrapu@gmu.edu
Bantwal Shreyas Mallya bmallya@gmu.edu
Saichandana Yenna syenna@gmu.edu

Flow Diagram

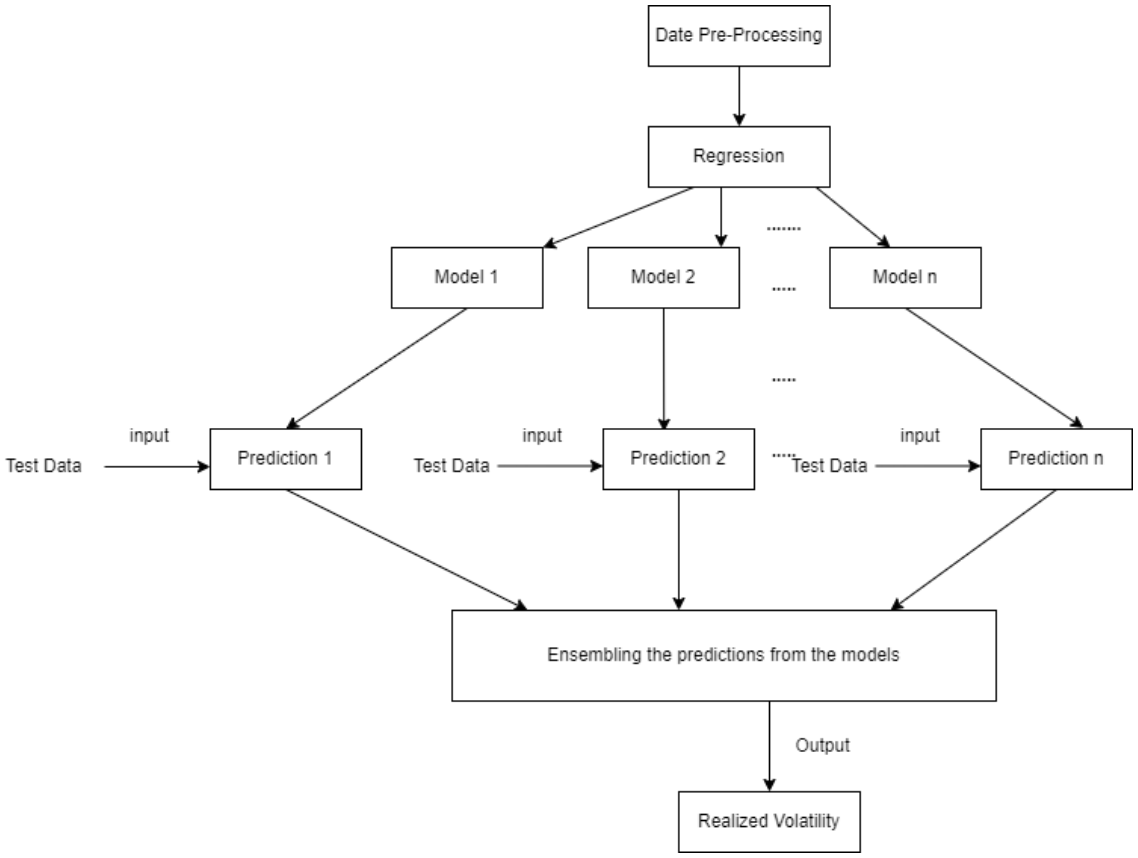

Figure 1: overall approach

The data pre-processing is done. This is followed by building regression models. Once the regression models are built, each model is evaluated. Ensembling is carried out by taking the weighted average of the predictions from each model, where outputs from each model are assigned a weight based on the R2 score and RMSPE of the respective output. This weighted average will give the realized volatility.

1 Introduction

In financial literature and in many other applications, volatility is a very central and critical topic. For any decisions to be made on the investment side, volatility is a very important factor. Volatility implies stability as well as loss. When volatility is high, there will be major losses. If the price of a stock fluctuates rapidly in a short period where it is hitting several high and low points, then it means that the price of the stock has high volatility.

There is a gateway to generate above average profit when the volatility spikes. On the other hand, there is also always a risk of losing a huge amount of capital in a short span of time. When the broader economy is considered, volatility has major consequences. In the market liquidity field, as the volatility increases, market liquidity decreases.

Predicting volatility can be a powerful instrument which can act as an insurance agent when the market fluctuates. People often think about volatility only when prices fall, however volatility can also refer to sudden price rises too.

Realized volatility: This is a non-parametric expost estimate of return realizations over a fixed time interval using all available high-frequency intraday data. This measure was built on the theory of continuous-time and arbitrage-free processes with the theory of quadratic variation. Realized volatility is important because it helps to quantify the inherent price risk arising out of volume fluctuations and external factors of a stock based on its historical performance. Many market makers are dedicated to continuously improving financial markets creating better access and prices for options, exchange-traded funds(ETFs), cash equities, bonds and foreign currencies on numerous exchanges around the world.

Summarization of Approaches:

The various models that we plan to use in our approaches are Gradient Boosting Decision Tree (GBDT), AdaBoost, CatBoost, XGBoost, LightGBM, TabNet, and Feed-Forward Neural Network (FFNN). Each model has its own advantages and disadvantages. The target variable is the realized volatility. Each model aims towards calculating the realized volatility after various of its parameters are set and the model is trained. The details of various approaches used in our project can be found in the "Details of Approach" section.

2 Related Work

In our reference paper, a prediction model is created using the Light Gradient Boosting Algorithm (Light GBM). LightGBM is a gradient boosting framework based on decision trees to increase the efficiency of the model. As opposed to other boosting algorithms that grow the decision tree level-wise, the LightGBM splits the tree leaf-wise.

The base paper mentions the LightGBM being optimized and used on the traditional Gradient Boosting Decision Tree (GBDT). This optimization is done in the following ways:

- Decision tree based on histogram
- Gradient based one side sampling which saves a lot of time and space overhead
- Exclusive Feature Bundling which achieves the purpose of dimensionality reduction
- A leaf wise algorithm with depth limitation
- Direct support for category features
- Supports efficient parallelism which includes splitting of the data into N partitions where each of the partitions can be used for training in different machines.
- The rate at which the cache hit is optimized.

Limitations of using LightGBM

- **Overfitting:** Since it splits the tree leaf wise, it can lead to overfitting due to the production of complicated trees.
- **Dataset Compatibility:** There is a high possibility it would overfit small datasets since it is very sensitive to overfitting.

XGBoost and Gradient Boosting Decision Tree are the other boosting algorithms that were used. But they have their own shortcomings.

XGBoost: It uses weak regression trees as weak learners. The algorithm also does cross-validation and computes the feature importance. Furthermore, it accepts sparse input data.

Limitations of using XGBoost

- Space consumption is large.
- Large overhead in time.
- Not friendly towards cache optimization.

GBDT (Gradient Boosting Decision Tree): is an additive ensemble learning approach that uses decision trees as weak learners. Additive means that the trees are added one after the other. Previous trees remain unchanged. When adding subsequent trees, gradient descent is used to minimize the loss.

Limitations of using GBDT

- GBDT needs to traverse the entire dataset multiple times.
- Loading the entire training data into memory limits the size of the training data, and repeatedly reading and writing training data can take a very large amount of time if you do not load it into memory.
- When the data size is at the industrial level, ordinary GBDT algorithm cannot meet the needs.

CatBoost: Uses a greedy approach to solve the issue of exponential feature set increase. CatBoost has common training parameters with XGBoost and LightGBM but provides a much flexible interface for parameter tuning.

Limitations of using CatBoost it's slightly difficult to tune the parameters which optimize the model for categorical features.

3 Data Sets

This dataset contains stock market data relevant to the practical execution of trades in the financial markets. To do the experiments, we have around 301k training data-set. And the test set contains data that can be used to construct features to predict approximately 128k target values.

- **Trade Parquet:** This file is partitioned by stock-id. It contains data on trades that were actually executed. It provides order book data on the most competitive buy and sell orders entered into the market.
- **Book Parquet:**This file is partitioned by stock-id. It provides the order book data on the most competitive buy and sell orders entered into the market. Contains data on trades that actually executed. Usually, in the market, there are more passive buy/sell intention updates (book updates) than actual trades.

  The various features in our datasets include the stock-id, time-id, seconds-in-bucket, bid-price, ask-price, bid-size, ask-size. The dataset that we plan to use is present at the link: https://www.kaggle.com/competitions/optiver-realized-volatility-prediction/data
- **seconds-in-bucket:** Number of seconds from the start of the bucket, always starting from 0.
- **bid-price:**Normalized prices of the most/second most competitive buy level.
- **ask-price:**Normalized prices of the most/second most competitive sell level.

- **bid-size:**The number of shares on the most/second most competitive buy level.
- **ask-size:**The number of shares on the most/second most competitive sell level.
- **price:**The average price of executed transactions happening in one second. Prices have been normalized and the average has been weighted by the number of shares traded in each transaction.
- **size:**The sum number of shares traded.
- **order-count:**TThe number of unique trade orders taking place

4 Details of the Approach

Proposing an ensemble model that aggregates the prediction of each base model and results in the final prediction of the unseen data. This is done so that it improves the accuracy of the predictive analysis and eliminates the cons of the individual models and gives a higher accuracy than the individual models.

4.1 Data-Preprocessing:

- **Weighted average price:** The weighted average price is abbreviated as WAP. It is used to calculate instantaneous stock valuation and calculate realized volatility as our target. The formula for the WAP can be given as follows:
  WAP = (BidPrice1 * AskSize1 + AskPrice1 * BidSize1) / (BidSize1 + AskSize1)
- **Log return:** For mathematical modeling purposes,Stock returns are computed in the form of Log returns where St is the price of the stock at time t,at various time intervals say t1 and t2 It is given by the formula
  Rt1, t2 = log (St1 / St2).
- **Realized volatility:**When we trade options, a valuable input to our models is the standard deviation of the stock log returns. The standard deviation will be different for log returns computed over longer or shorter intervals, for this reason it is usually normalized to a 1-year period and the annualized standard deviation is called volatility.log returns are computed over all consecutive book updates and realized volatility is defined as where, sigma is the squared root of the sum of squared log returns. $\sigma = \sqrt{\sum {r^2}_{t-1,t}}$

Book pre-processor and Trade pre-processor functions process the both train and test data after reading from respective book and trade parquet files for corresponding stock-ids where realized volatility is calculated using weighted average prices and log returns and and the statistics for each group for different windows(seconds-in-bucket -500,400,300,200,100) and the features are merged into a dataframe for the set of records of each stock-id and further all of the stock-id data-frames are concatenated into single dataframe.

- **K-fold Cross Validation:** KFold cross validation approach taken as 5 folds where the training set is split into k smaller sets.A model is trained using k-1 of the folds as training data; the resulting model is validated on the remaining part of the data (i.e., it is used as a test set to compute a performance measure -RMSPE).

4.2 Evaluation metrics:

- **Root Mean Square Prediction Error:(RMSPE)** It measures how far the predications fall from measured true values using Euclidian distance.
- **R2 score:** A statistical measure of fit that indicates how much variation of a dependent variable is explained by the independent variable(s) in a regression model.
- **Weighted Performance:**
  - Based on the evaluation metric, these models are assigned ranks.
  - The model which performs the best is given the highest rank and the one that performs the worst is given the lowest rank.

$$\text{RMSPE} = \sqrt{\frac{1}{n}\sum_{i=1}^{n}((y_i - \hat{y}_i)/y_i)^2} \qquad R^2 = 1 - \frac{\sum_{i=1}^{n}(\hat{y}_i - y_i)^2}{\sum_{i=1}^{n}(y_i - \bar{y}_i)^2}$$

(a) RMSPE (b) R2 score

Figure 2: RMSPE and R2 formulae

– Once the ranks are assigned, a weighted average is calculated and used to give a final output The weighted average is calculated as per the formula below:

$$\textbf{Weighted Performance=}[\frac{\Sigma(rank_n * prediction - of - model_n)}{\Sigma(rank_n)}] \quad (1)$$

– Here rank-n is the rank of model n and evaluation-metric-result-n is the evaluation metric result of model n.

– Once the weighted performance value is obtained, it can be compared with the evaluation metric result of each model.

4.3 Pseduocode:

– 1.Assign various parameters for the models.

– 2. Perform K-Fold Cross Validation.

– 3. Set up the model.

– 4. Fit the model.

– 5. Train the model.

– 6. Calculate the predictions on the test set.

– 7.Compute the evaluation metrics namely R2 score and RMSPE.

4.4 Algorithms:

– **LightGBM:**It is a gradient boosting framework that uses tree-based learning algorithms. For Gradient-based one-side sampling (GOSS), all the instances with large gradients contribute more towards the information gain. The data with large gradients can be targeted using pre-defined threshold to select large gradients or selecting the top percentile and randomly drop the ones that have small gradients. [1].
Exclusive Feature Bundling aims to eliminate ineffective sparse features by bundling them into one. In greedy bundling, a graph is first built using edges to represent total conflicts then sort the features by the degree in the graph, based on the knowledge, and assign the feature to its corresponding bundles. In the phase of Transforming the features into one, the new feature will be represented in bins and the critical thing is to ensure that the relationship between the original feature and new feature bin is 1:1.
Various parameters are set up for LightGBM and they are as follows: **boosting type:** Gradient boost decision tree **max-depth:** -1. **max-bin:** This is the maximum number of bins the feature values will be bucketed in. The value used here is 100. **min-data-in-leaf:** 500. **learning-rate:** 0.05. **sub-sample:** percentage of rows used in tree building. given a value of 0.72. **sub-sample-freq:**The frequency for bagging. The value used here is 4. **feature-fraction:**randomly select a subset of features on each iteration. The value given here is 0.5. **lambda-l1:** – It is given a value of 0.5. **lambda-l2:** It is given a value of 1.0. **categorical-column:**The value given to it is None. It is a parameter where the name of the categorical column if present is mentioned.

Figure 3: LightGBM

(a) Merging-Exclusive-Features

Input: $numData$: number of data
Input: F: One bundle of exclusive features
binRanges ← {0}, totalBin ← 0
for *f in* F **do**
  totalBin += f.numBin
  binRanges.append(totalBin)
newBin ← new Bin(numData)
for $i = 1$ ***to*** $numData$ **do**
  newBin[i] ← 0
  for $j = 1$ ***to*** *len(F)* **do**
    if *F[j].bin[i]* $\neq 0$ **then**
      newBin[i] ← F[j].bin[i] + binRanges[j]
Output: $newBin, binRanges$

(b) GOSS

Input: I: training data, d: iterations
Input: a: sampling ratio of large gradient data
Input: b: sampling ratio of small gradient data
Input: $loss$: loss function, L: weak learner
models ← {}, fact ← $\frac{1-a}{b}$
topN ← a × len(I) , randN ← b × len(I)
for $i = 1$ ***to*** d **do**
  preds ← models.predict(I)
  g ← $loss(I, \text{preds})$, w ← {1,1,...}
  sorted ← GetSortedIndices(abs(g))
  topSet ← sorted[1:topN]
  randSet ← RandomPick(sorted[topN:len(I)], randN)
  usedSet ← topSet + randSet
  w[randSet] × = fact ▷ Assign weight $fact$ to the small gradient data.
  newModel ← L(I[usedSet], − g[usedSet], w[usedSet])
  models.append(newModel)

(c) Greedy-Bundling

Input: F: features, K: max conflict count
Construct graph G
searchOrder ← G.sortByDegree()
bundles ← {}, bundlesConflict ← {}
for *i in searchOrder* **do**
  needNew ← True
  for $j = 1$ ***to*** *len(bundles)* **do**
    cnt ← ConflictCnt(bundles[j],F[i])
    if *cnt + bundlesConflict[i]* $\leq K$ **then**
      bundles[j].add(F[i]), needNew ← False
      break
  if *needNew* **then**
    Add $F[i]$ as a new bundle to $bundles$
Output: $bundles$

Baseline Approach: The base approach includes the following steps:

* By using GBDT, we combine a set of weak base learners into a strong one. GBDT has the advantages of strongly generalizing ability and not being easy to overfitting.
* LightBGM can reduce memory usage and improve the training speed by using the decision tree algorithm based on histogram.
* We ensemble Adaboost and GBDT with LightGBM to find the performance evaluation and better accuracy.
* AdaBoost, which uses the error of a previous weak learner to update the sample weight, then iterates round by round, where GBDT updates the sample data in the direction of the negative gradient to make the algorithm converge globally.
* **AdaboostRegressor:** This algorithm creates a model while assigning each data point an equal weight. Then, it gives points that were incorrectly categorized with higher weights. The next model now gives more weight to all the points with higher weights. it will continue to train the models until the error is lower. **Base-estimator:** The base estimator from which the boosted ensemble is built. **N-estimators:** The highest estimated size at which boosting is stopped. Perfect fit causes the learning process to end early.
  learning-rate: Each iteration of the boosting process adds weight to each regressor. Each regressor's contribution is increased as the learning rate increases.
  loss'linear', 'square', 'exponential': The loss function to use when updating the weights after each boosting iteration.
* **GradientBoostingRegressor:** This estimator allows for the optimization of any differentiable loss function and constructs an additive model in a forward stage-wise manner. To understand the Gradient boost below are the steps involved. In Gradient boosting weak learners are decision trees.
  Step1: Create a foundation tree that has just one root node. It represents the initial guess for all samples.

Step2: Build a tree from errors of the previous tree.
Step3: Modify the tree depending on learning rate. This learning rate determines the contribution of the tree in the prediction
Step4: Repeat step 2 until the maximum number of trees are used or the fit is not improved by adding the new tree to the previous trees in order to forecast the outcome. The final prediction model is the combination of all the trees.
Parameters are as follows:
loss('squared-error', 'absolute-error', 'huber', 'quantile'): Loss function to be optimized. 'squared-error' refers to the squared error for regression. 'absolute-error' refers to the absolute error of regression and is a robust loss function. **learning-rate**Learning rate shrinks the contribution of each tree by learning rate. **n-estimators:**The number of boosting stages to perform.A large number typically yields better performance since gradient boosting is fairly resistant to overfitting **criterion('friedman-mse', 'squared-error', 'mse'):**The function to measure the quality of a split. Supported criteria are "friedman-mse" for the mean squared error with improvement score by Friedman, "squared-error" for mean squared error. **min-samples-split:**The minimum number of samples required to split an internal node. **min-samples-leaf:**The minimum number of samples required to be at a leaf node. **Min-weight-fraction-leaf:**The minimum weighted fraction of the sum total of weights required to be at a leaf node. **max-depth:**Maximum depth of the individual regression estimators. The maximum depth limits the number of nodes in the tree. **Random-state:** Controls the random seed given to each Tree estimator at each boosting iteration. **max-features('auto', 'sqrt', 'log2'):**The number of features to consider when looking for the best split **max-leaf-nodes:**Grow trees with max-leaf-nodes in best-first fashion. Best nodes are defined as relative reduction in impurity. **N-iter-no-change:**It is used to decide if early stopping will be used to terminate training when validation score is not improving.
Approach 1:

* **CatBoost:** It supports numerical, categorical, and text features but also has a good handling technique for categorical data. Parameters are set up as follows:
  n-estimators : Here the n-estimators is the number of iterations to see how the model is improving on each iteration. The value is set to 100.
  loss-function: The loss function used here is the RMSE which is the root mean square error. This is the metric used in training. RMSE is the standard deviation of the residuals and a measure of how spread out these residuals are.
  eval-metric : The eval metric is the metric used for overfitting detection and best model. The eval-metric used here gain is the RMSE.
  cat-features : This is the method used for storing the categorical feature values and it is set to None here.
* **XGBoost:**It is a gradient boosting framework that uses tree-based learning algorithms.Parameters are set as follows:
  n-estimators: It is the number of runs XGBoost will try to learn. It's value is set to a list of values 100, 200, 300, 400.
  learning-rate:A list of values namely [0.001, 0.005, 0.01, 0.05] is given.
  max-depth: It indicates the maximum depth of the tree. It takes a list of values [8, 10, 12, 15].
  gamma: It is a pseudo-regularization hyperparameter in gradient boosting.
  random-state: It is given a value of 42 to get the same train and test sets across different executions.
* The prediction from each of the individual models namely CatBoost, XGBoost, and LightGBM would be taken.
* Once the predictions from these individual models are obtained, the evaluation metric for each of these individual models is calculated too.

* If the ensemble model results are higher than the results of all the individual models, then it means there is a better way to evaluate the performance.

Approach 2: To improve the performance of the ensembling technique and overcome the limitations of boosting algorithms, TabNet is proposed to implement along with combination of LightGBM and Feed Forward Neural Network (FFNN) models.

TabNet:

* TabNet is a deep tabular data learning architecture that uses sequential attention to choose which features to reason from at each decision step, enabling interpretability and better learning as the learning capacity is used for the most useful features.
* The main concepts Ghost Batch Normalization(GBN)allows to train large batches of data and also generalize better at same and Sparsemax enables the model to select relevant features at each decision step more effectively.
* The components Attention Transformer, models learns the relationship between relevant features and decides which features to pass on to the feature transformer of the current decision step where as in Feature Transformer all the selected features are processed to generate the final output using multiple Gated Linear Unit Blocks. Hyperparameters are set as **n-d, n-a:** 16 ,16 **batch size:** 1024 **virtual batch size:** 128 **max-epochs :**are 100 **optimizer-fn=**given as Adam, **optimizer-params=**dict(lr=(2e-2)), **sparsity regularization constant:** 0 to 0.00001 **number of shared GLU Blocks:** 2 **number of independent decision Blocks:**given as 2 **relaxation constant:** 1 to 2.5 **number of decision steps:** are taken as 2 **batch normalization momentum:** 0.5 to 0.98 **eval-metric=**[RMSPE], **loss-fn=**RMSPELoss

FFNN:

* It is an artificial neural network in which the connections between nodes do not form a cycle where information is only processed in one direction.
* While the data may pass through multiple hidden nodes, it always moves in one direction and never backwards. channel the inputs (to the input layer) through a set of computational nodes which are mathematical operators and activation functions arranged in layers to calculate the network outputs.
* The output layer is the final layer of the neural network and contains linear functions.
* The layers between the input layer and the output layer are called hidden layers and contains 'swish' activation functions Parameters: hidden-units = (128,64,32) stock-embedding-size = 24 optimiser:Adam(learning-rate=0.006) batch-size=2048, epochs=50
* In this ensemble model, the predictions from the mentioned individual models are evaluated using evaluation metric and ranks are assigned to each model and based upon the ranks and evaluation metric values, weights are averaged and computed as per the formula given in weighted performance and the performance is estimated and compared with the Approach 1 and Approach 2.

5 Results:

5.1 RMSPE and R2 Scores for each Model:

Model	RMSPE	R2-score
LightGBM	0.2050	0.8440
AdaBoost	1.91	0.724
GBDT	0.2315	0.704
CatBoost	0.2591	0.871
XGBoost	0.2591	0.824
Tabnet	0.2315	0.853
Feedforward	0.2103	0.864

5.2 Weighted performance for Ensemble Models:

Model	weighted performance: RMSPE	weighted performance: R2-score
LightGBM +CatBoost + XGBoost	0.5769	0.6930
LightGBM +AdaBoost + GBDT	2.578	0.7242
LightGBM+ Tabnet + Feedforward	0.632	0.765

5.3 Feature Importance

Feature importance graph is plotted for each model where the x-axis shows the importance value of the feature and the y-axis shows the name of the features. For the sake of simplicity, the top 20 most important features chosen for each model

LightGBM: In the feature-importance graph, it can be seen that the stock-id is the most important feature.
CatBoost: log-return1-relaized-volatility-200 is the most important feature. This is the stock return value in the time window value of 200.
AdaBoost: The x-axis shows the importance value of the feature and the y-axis shows the name of the features. The top 20 features are chosen and it is seen that total-volume-sum-1c1 is the most important feature
GradientBoost: The x-axis shows the importance value of the feature and the y-axis shows the name of the features. The top 20 features are chosen and it is seen that log3-realized-volatility-200 is the most important feature.

6 Discussion and conclusions:

– By ensembling the GradientBoost and AdaBoost, the RMSPE of the ensembled model was much higher with a value of 2.578 than the individual models. It was more than the RMSPE score of the individual models and R2 score of the ensembled model is almost similar to that of the individual models.

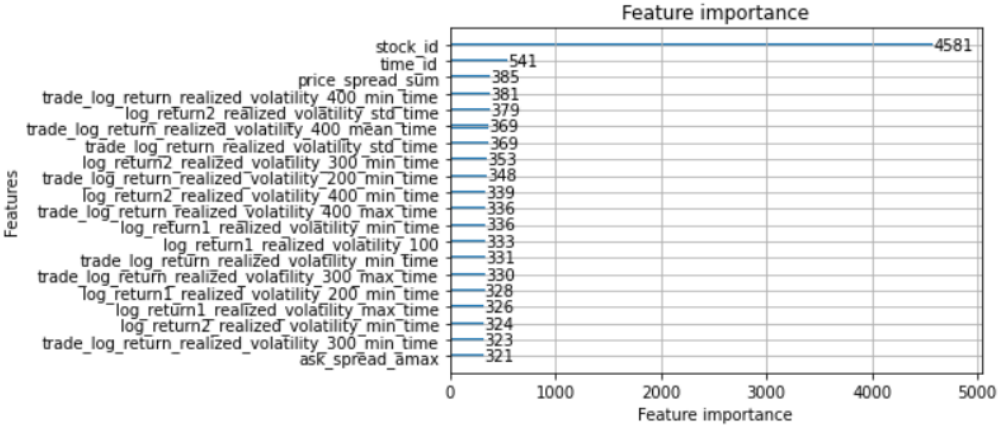

Figure 4: LightGBM

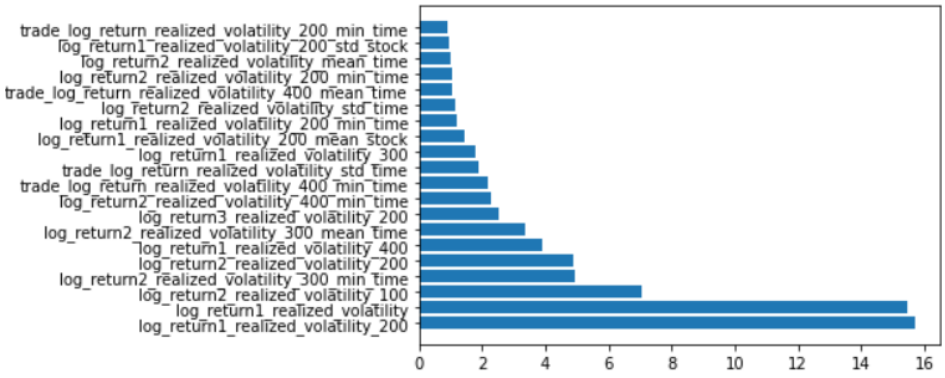

Figure 5: CatBoost

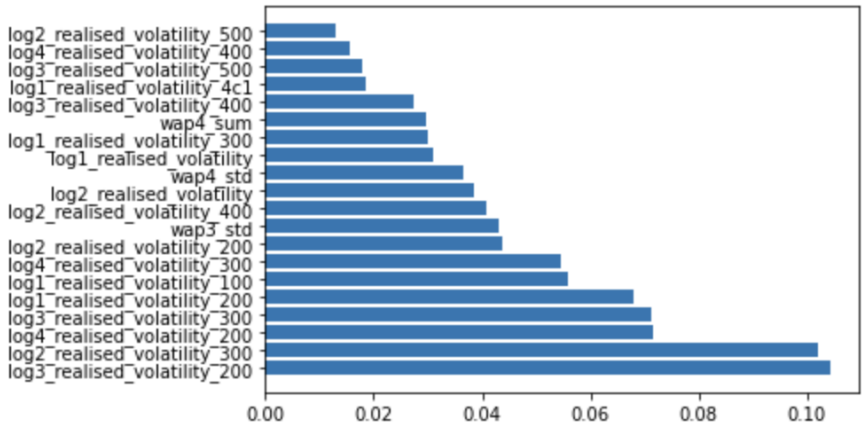

Figure 6: GradientBoost

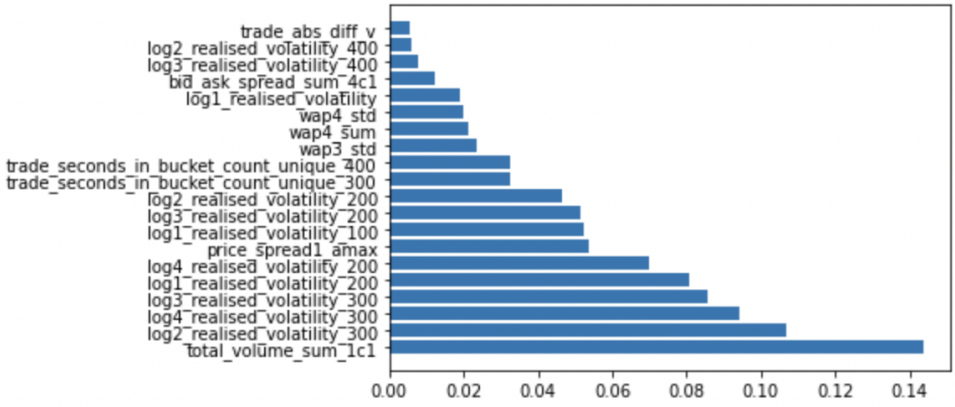

Figure 7: Adaboost

– When the LightGBM ,CatBoost and XGBoost were ensembled, the RMSPE of the ensembled model was much higher with a value of 0.5769 than the individual models. It was improved by a little more than twice the RMSPE score of the individual models. But the R2 score of the ensembled model was low when compared to the individual models.
– When the Tabnet and Feedforward were ensembled along with LightGBM, the RMSPE of the ensembled model was higher with a value of 0.632 than the individual models of Tabnet(0.2315) and Tabnet(0.2103).Also, the R2 score of the ensembled model was low having value as 0.765 when compared to the individual models of Tabnet(0.853) and Feedforward(0.864).

7 Statement of individual contribution:

Common statement for all 3 of us: We did the data pre-processing, built the ensemble models, calculated the evaluation metrics namely RMSPE, R2 score, and weighted performance for the ensemble model, and plotted the feature importance of our models.
Shreyas – Implemented approach lightGBM, CatBoost, and XGBoost
Chandana - Implemented approach lightGBM, AdaBoost, and GradientBoost
Sahana - Implemented approach lightGBM, TabNet and Feedforward Neural Network

8 References:

[1] https://ieeexplore.ieee.org/stamp/stamp.jsp?arnumber=9543438
[2] https://www.kaggle.com/competitions/optiver-realized-volatility-prediction
[3] https://link.springer.com/article/10.1007/s12541-019-00048-6
[4] https://towardsdatascience.com/boosting-algorithms-explained-d38f56ef3f30
[5] https://www.geeksforgeeks.org/catboost-ml/
[6] https://yanpuli.github.io/posts/2018/05/blog-post-13/
[7] https://towardsdatascience.com/implementing-tabnet-in-pytorch-fc977c383279
[8] https://towardsdatascience.com/implementing-tabnet-in-pytorch-fc977c383279
[9] https://github.com/ds-wook/optiver-prediction