# Predicting Market Volatility using Hybrid Models

25HR06

| | |
|---|---|
| Abhinav Kumar | 2301AI50 |
| Swagatam Pati | 2301AI28 |
| Soumyakanta Panda | 2301CS86 |
| Kalp Alpesh Shah | 2301CS93 |
| Anurag Nath | 2301CS07 |

# volatility ?

## understanding market volatility dynamics

> *realized volatility* quantifies the degree of actual variability in asset prices over a specified period ... how much prices move up or down over time

> big swings  = high volatility
  small moves = low volatility ... tells you how unpredictable the market is

> usually calculated from historical intraday or daily returns

> typically computed using squared returns and often annualized

> differs from *implied volatility*
  ... derived from option prices and reflects market expectations

## applications ?

understanding applications of realized volatility

> realized volatility provides a backward-looking, data-driven view of market uncertainty ... helps measure how unstable a stock or market has been

> serves as a vital tool in quantitative finance for risk assessment, portfolio optimization, and pricing of derivatives

> valuable in high-frequency trading and market micro-structure analysis, where precise measurement of short-term price dynamics is crucial

> accurate estimation of this metric enhances model reliability and financial decision-making

## 🌀 problem formulation

> objective

predict short-term realized volatility using high-frequency limit order book [LOB] data

> challenges

- market micro-structure noise in high-frequency data
- need for fine-grained feature engineering [e.g. WAP, order flow]
- temporal dynamics and stock-specific behavior variation

> goal

design a model pipeline leveraging LightGBM, MLP, and CNN models to capture spatial and temporal dependencies

> output

a numerical prediction of volatility for each stock-time pair in the test set.

## 🌀 traditional models

> GARCH

good at modeling time-varying volatility but inadequate for high-frequency granularity

> ARIMA

effective for trend/seasonality but lacks predictive power for nonlinear volatility dynamics

## 🌀 modern approaches

> LSTM (long short-term memory)

captures sequential dependencies but suffers from long training times and over-fitting in small datasets

> CNNs for time series

shown to capture short-term patterns efficiently

> Gradient Boosting (LightGBM)

strong tabular learner for engineered features

# dataset

> source

*Optiver Realized Volatility Prediction* [@kaggle]
https://www.kaggle.com/competitions/optiver-realized-volatility-prediction

> content

- book data (limit order book)
  - “bid/ask prices” and “sizes” for top 10 levels timestamped within each time bucket
- trade data
  - □ - executed trades: “price”, “size”, “number_of_orders”

> target variable

- realized volatility per “stock_id” and “time_id”.
- calculated from log returns of WAP [weighted average price]

> challenges

- high dimensionality due to LOB levels
- missing values and irregular time intervals
- stock-specific patterns vary drastically

44 csv files ; 1 file for each stock
1 file

ask_price, bid_price, ask_size, bid_size, time_id

(1 cent)

# preprocessing

> data cleaning

- removed rows with missing / invalid values in key columns [“prices”, “sizes”]

> timestamp alignment

- converted all relative timestamps to a consistent scale
- grouped events by “time_id” buckets [10-minute intervals]

> price normalization

multiplied book price levels by estimated tick size to recover real-world prices

## … tick size recovery

$$\text{tick}_i = \min\left(\text{diffs} \in \{p_j - p_{j-1} \mid p_j > p_{j-1} > 0\}\right)$$

> purpose

- raw book prices are normalized [unit-less]
- recovering tick size helps map them back to actual monetary values

> method

for each stock:

- □ - collected all unique non-zero level-1 prices [“ask_price”, “bid_price1”]
- □ - sorted them and computed differences between adjacent prices
- □ - used the minimum meaningful positive difference [above 1e-8] as the tick size

# time-order reconstruction

.> objective

- reconstruct a global ordering of “time_ids” across all stocks based on similarity
- enables time-aware models like CNNs and avoids temporal leakage in cross-validation

1> pivot table creation

- constructed a matrix of shape:
  *(time_id) x (stock_id)* where each entry holds the end-of-bucket WAP
- missing entries filled using:
  □ - forward fill (ffill), backward fill (bfill)

$$\text{WAP} = \frac{P_{\text{ask}} \cdot Q_{\text{bid}} + P_{\text{bid}} \cdot Q_{\text{ask}}}{Q_{\text{ask}} + Q_{\text{bid}}}$$

2> PCA (principal component analysis) for denoising

- applied PCA to reduce dimensionality
- captured dominant patterns in WAP movements

3> t-SNE for embedding

- applied t-distributed stochastic neighbor embedding (t-SNE) on PCA output:
  □ - reduced dimensionality to 1D
  □ - preserved temporal proximity of similar “time_ids”
- *- perplexity=30, learning_rate=200, n_iter=1000*

4> time index mapping

- - sorted the embedded 1D values to produce a ranked ordering of all “time_ids”

# nearest neighbor aggregates

## … purpose

enhance each (stock, time) entry with features based on similar contexts using k-Nearest Neighbors (k-NN)

$$\text{log\_return}_t = \ln\left(\frac{\text{WAP}_t}{\text{WAP}_{t-1}}\right)$$

> time-based nearest neighbors (time-NN)

- For each stock, construct vectors of recent *log_return* values (e.g. last 10 buckets).
- Apply Euclidean distance to find k-nearest time windows within the same stock.
- Aggregate: mean_ret_3

> stock-based nearest neighbors (stock-NN)

- For each time ID, normalize features like log_wap across stocks
- Use k-NN to find similar stocks at the same time
- Aggregate: mean_ret_3

## LGBM

> LightGBM is a high-performance gradient boosting framework based on decision trees

> optimized for speed and memory usage, making it ideal for large tabular datasets

> why it works?

- captures nonlinear interactions
- handles missing data efficiently
- fast histogram-based learning

> training objective

regression using mean squared error (MSE)

$$\hat{y} = \sum_{m=1}^{M} f_m(x), \quad f_m \in \text{Trees}$$

$$\mathcal{L} = \frac{1}{n} \sum_{i=1}^{n} (y_i - \hat{y}_i)^2$$

# 1D-CNN

> 1D-CNN models sequences of time-series data (e.g. log-returns across previous time buckets) to detect local patterns and temporal structures

> why it works?

- efficient at capturing temporal locality
- learns spatial hierarchies with depth
- suited for sequence modeling

$$h_t = \sigma\left(\sum_{k=1}^{K} w_k \cdot x_{t-k+1} + b\right)$$

> structure

- input: shape (“n_samples”, “seq_len”, “n_features”)
- layers: Conv1D → ReLU → MaxPooling → Flatten → Dense

# MLP

> Multi Layer Perceptron is a fully connected neural network that maps static tabular features to predictions using nonlinear transformations

> why it works?

- learns complex nonlinear relationships
- effective with engineered features
- simple architecture with high flexibility

> structure

- Input → Dense (ReLU) → Dense (ReLU) → Output (Linear)

For layers $l = 1, 2, ..., L$:

$$a^{(l)} = \sigma(W^{(l)} a^{(l-1)} + b^{(l)})$$

Final output:

$$\hat{y} = W^{(L)} a^{(L-1)} + b^{(L)}$$

## Cross-validation techniques

… to evaluate model generalization and prevent overfitting by simulating real-time prediction scenarios

> time-ordered cross validation (time-series aware folds)

- unlike random shuffling, time-series data requires folds that respect temporal order
- for each fold:
  - □ - training is done on earlier time windows
  - □ - validation is performed on a later, unseen time segment

> advantages

- Mimics real-world forecasting, ensuring model learns only from the past.
- Prevents data leakage from future time buckets.
- Enhances robustness of model selection and hyperparameter tuning.

> Process

1. Sort data by time_ord (reconstructed chronological order).
2. Divide into K folds (e.g., 5-fold).
3. For fold k, train on time buckets “< t_k”

$$\mathrm{MSE}_k = \frac{1}{n_k}\sum_{i=1}^{n_k}(y_i - \hat{y}_i)^2$$

$$\text{Final MSE} = \frac{1}{K}\sum_{k=1}^{K}\mathrm{MSE}_k$$

# ensemble and weights optimization

to improve prediction accuracy by combining the strengths of multiple diverse models—LightGBM, 1D-CNN, and MLP

$$\hat{y}_{ensemble} = w_1 \cdot \hat{y}_{LGB} + w_2 \cdot \hat{y}_{CNN} + w_3 \cdot \hat{y}_{MLP}$$

where $w_1 + w_2 + w_3 = 1$

> ensemble strategy

- Model Outputs: each model predicts the target independently
- Stacking: combine outputs as weighted average

> weight optimization

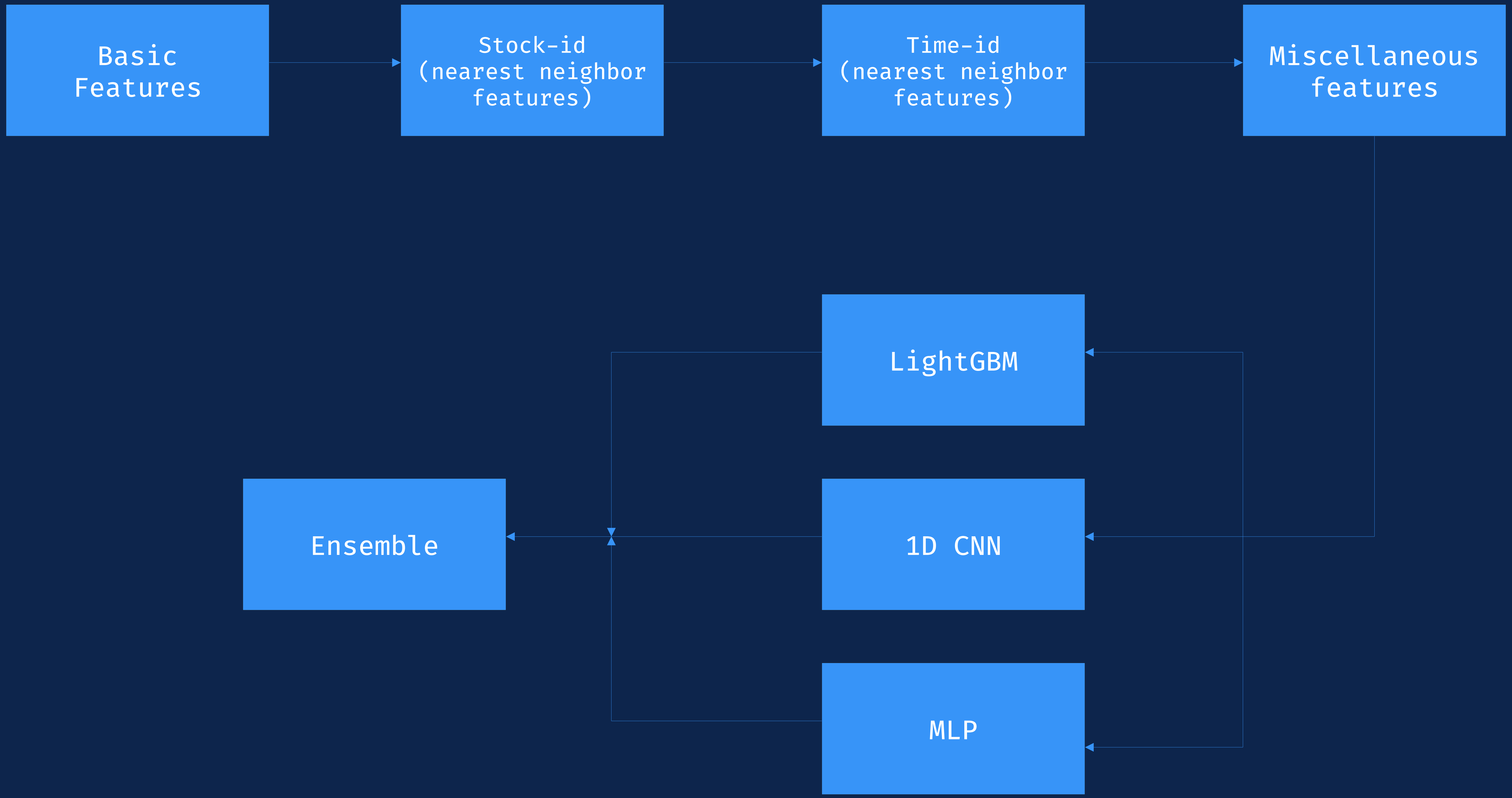
- grid search or optimization over validation set to minimize Mean Squared Error (MSE)
- constraints:
  - □ - Non-negative weights
  - □ - Sum to 1 for normalization

$$\mathrm{MSE}(w) = \frac{1}{n} \sum_{i=1}^{n} \left( y_i - \hat{y}^{(i)}_{ensemble} \right)^2$$

> why ensemble works?

- Captures both linear/tabular (LGB), sequential (CNN), and dense relational (MLP) patterns
- Reduces individual model biases and variances

# workflow summary


Basic Features
Stock-id (nearest neighbor features)
Time-id (nearest neighbor features)
Miscellaneous features
LightGBM
1D CNN
MLP
Ensemble

# results

| Model | Fold 1 MSE | Fold 2 MSE | Fold 3 MSE | Fold 4 MSE | Fold 5 MSE | Avg MSE |
|---|---|---|---|---|---|---|
| LightGBM | .0.0172 | 0.0180 | 0.0169 | 0.0175 | 0.0178 | 0.0175 |
| CNN | 0.0165 | 0.0169 | 0.0167 | 0.0171 | 0.0168 | 0.0168 |
| MLP | 0.0181 | 0.0176 | 0.0179 | 0.0180 | 0.0177 | 0.0179 |
| Ensemble | _ | _ | _ | _ | _ | 0.0162 |

thank you