Instructor: Shuo Zhang, Ph.D., Columbia University
Contact: shuozhang1985@gmail.com

Kaggle Featured Code Competition: Optiver Realized Volatility Prediction

1. Introduction

"Knowledge starts with practice." Kaggle is a good platform for learning data science skills through practice. We found an interesting feature code competition on Kaggle in June and decided to use it for a systematic exercise. The competition called "Optiver Realized Volatility Prediction" was launched by Optiver, a leading global electronic market maker. This competition lasts for three months and aims to predict short-term volatility for hundreds of stocks across different sectors through the establishment of models. In financial markets, volatility captures the amount of fluctuation in prices. For trading firms like Optiver, accurately predicting volatility is essential for the trading of options, whose price is directly related to the volatility of the underlying product. Optiver's teams have spent countless hours building sophisticated models that predict volatility and continuously generate fairer options prices for end investors. Optiver wants to take its model to the next level with the help of this competition.
In addition, this competition will use the root mean square percentage error (RMSPE) to evaluate the submissions of participants. RMSPE is defined as:

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

In the two months, we performed EDA, feature engineering, built and optimized three machine learning models, and finally used ensemble strategies to boost model performance. When the competition ends, we ranked 131 among 3852 international teams.

2. Workflow

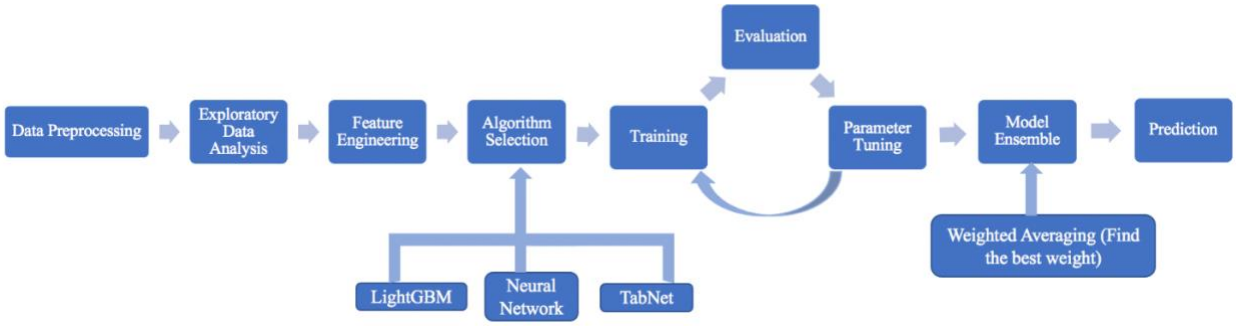


Figure 1: Workflow of this project

3. EDA

The main purpose of EDA is to help review the data before making any assumptions. It can help identify obvious errors, better understand patterns within the data, detect outliers or abnormal events, and find interesting relationships between variables. Our team has deeply explored the information in the data through data analysis and data visualization.

3.1. Description of Competition Data

The dataset provided by this competition contains stock market data relevant to the practical execution of trades in the financial markets. In particular, it includes order book snapshots and executed trades. The following describes the four files contained in this dataset and the variables of each file.

File 1:

book_[train/test].parquet: A parquet file partitioned by stock_id. Provides order book data on the most competitive buy and sell orders entered into the market.

stock_id	ID code for the stock.
time_id	ID code for the time bucket.
seconds_in_bucket	Number of seconds from the start of the bucket, always starting from 0.
bid_price[1/2]	Normalized prices of the most/second most competitive buy level.
ask_price[1/2]	Normalized prices of the most/second most competitive sell level.
bid_size[1/2]	The number of shares on the most/second most competitive buy level.
ask_size[1/2]	The number of shares on the most/second most competitive sell level.

File 2:

trade_[train/test].parquet: A parquet file partitioned by stock_id. Contains data on trades that actually executed.

stock_id	ID code for the stock.
time_id	ID code for the time bucket.
seconds_in_bucket	Number of seconds from the start of the bucket. This field is not necessarily starting from 0.
price	The average price of executed transactions happening in one second.
size	The sum number of shares traded.
order_count	The number of unique trade orders taking place.

File 3:
train.csv: The ground truth values for the training set.

stock_id	ID code for the stock.
time_id	ID code for the time bucket.
target	The realized volatility computed over the 10 minute window following the feature data under the same stock/time_id.

File 4:
test.csv: Provides the mapping between the other data files and the submission file.

stock_id	ID code for the stock.
time_id	ID code for the time bucket.
row_id	Unique identifier for the submission row. There is one row for each existing time ID/stock ID pair.

3.2. Data Distribution Pattern

We have an understanding of the distribution of the main variables. Figure 2 below shows the distribution of the variable target in train.csv. Obviously, this distribution is right-tailed. The value of target is mainly concentrated between 0 and 0.005.

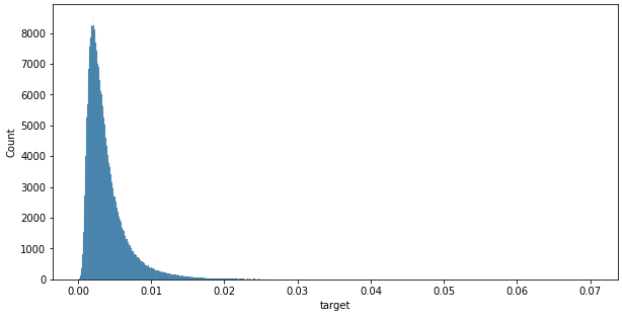


Figure 2: Distribution of target

Figure 3 shows the comparison between the four prices in the book file and the trade prices in the trade file, and their distribution. According to the figure, we know that ask_price2>ask_price1>trade_price>bid_price1>bid_price2. In addition, these prices vary greatly over time.

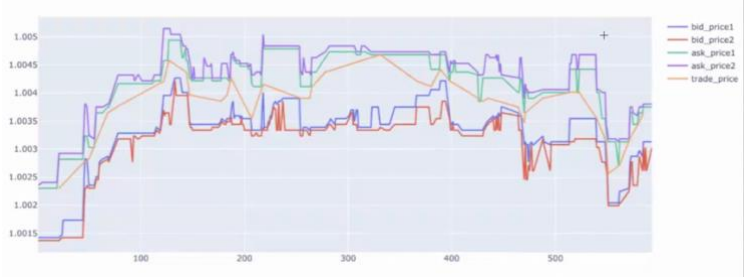


Figure 3: Comparison of different prices and their distribution

Figure 4 and Figure 5 below show the distribution of bid_size and ask_size. According to Figure 4, we can know that bid_size1 is a little higher than bid_size2. According to Figure 5, we can know that ask_size1 is a little higher than ask_size2. In this case, the stock price is expected to rise because the buy level is higher than the sell level. It seems to be relatively high volatility.

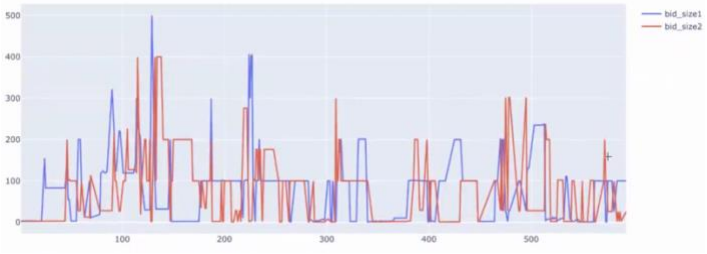


Figure 4: Distribution of bid size

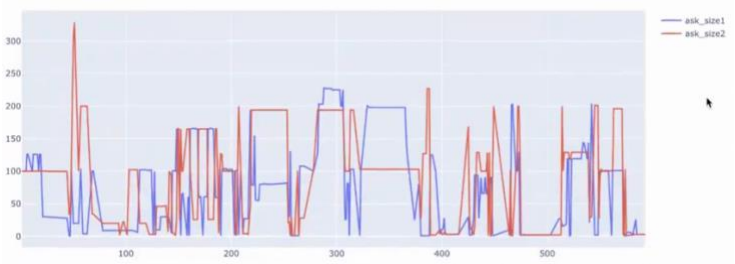


Figure 5: Distribution of ask size

As realized volatility is a statistical measure of price changes on a given stock, to calculate the price change we first need to have a stock valuation at the fixed interval (1 second). We will use weighted average price (WAP), of the order book data. Figure 6 below shows the distribution of WAP. WAP is defined as:

$$WAP = \frac{bid\_price1 \times ask\_size1 + ask\_price1 \times bid\_size1}{bid\_size1 + ask\_size1}$$

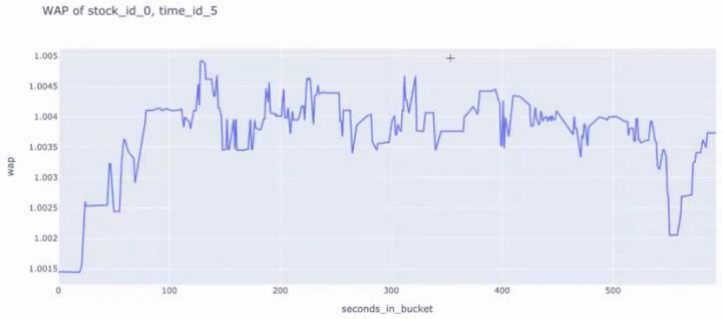


Figure 6: Distribution of WAP

To compute the log return, we can simply take the logarithm of the ratio between two consecutive WAP. The first row will have an empty return as the previous book update is unknown, therefore the empty return data point will be dropped. Figure 7 below shows the distribution of log return.

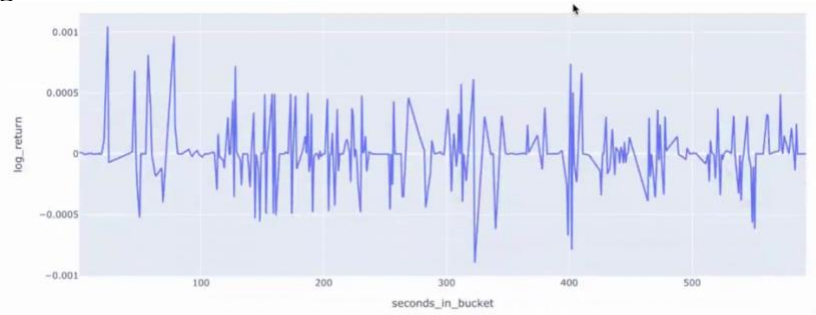


Figure 7: Distribution of log return

4. Feature Engineering

Since the data contains limited feature, we create multiple new features using given features.

4.1 book_train

We firstly create 4 difference WAP using a combination of bid_price and ask_price.

WAP1	$\frac{bid\_price1 \times ask\_size1 + ask\_price1 \times bid\_size1}{bid\_size1 + ask\_size1}$
WAP2	$\frac{bid\_price2 \times ask\_size2 + ask\_price2 \times bid\_size2}{bid\_size2 + ask\_size2}$
WAP3	$\frac{bid\_price1 \times bid\_size1 + ask\_price1 \times ask\_size1}{bid\_size1 + ask\_size1}$
WAP4	$\frac{bid\_price2 \times bid\_size2 + ask\_price2 \times ask\_size1}{bid\_size2 + ask\_size2}$

Then we compute 4 log return using 4 WAP, which can be calculated by simply taking the logarithm of the ratio between two consecutive WAP.

Then we add extra features by combining original feature and derive following new features

Wap balance	Wap1 – Wap2
Price spread	$\frac{askPrice1 - bidprice1}{(askprice1 + bidprice1)/2}$
Price_spread2	$\frac{askPrice2 - bidprice2}{(askprice2 + bidprice2)/2}$
Bid spread	bid price1 – bid price 2
Ask spread	ask price1 – ask price 2

Bid ask spread	| bid spread – ask spread |
Total volume	$asksize1 + asksize2 + bidsize1 + bidsize2$
Volume imbalance	| (ask size1 + ask size 2) – (bid size 1 + bid size 2)|

We create 5 times windows based on seconds_in_bucket. Then we generate several statistics including summation and maximum for 5 times windows for each new feature by grouping their time_id

```
create_feature_dict = {
    'wap1': [np.sum, np.std],
    'wap2': [np.sum, np.std],
    'wap3': [np.sum, np.std],
    'wap4': [np.sum, np.std],
    'log_return1': [realized_volatility],
    'log_return2': [realized_volatility],
    'log_return3': [realized_volatility],
    'log_return4': [realized_volatility],
    'wap_balance': [np.sum, np.max],
    'price_spread':[np.sum, np.max],
    'price_spread2':[np.sum, np.max],
    'bid_spread':[np.sum, np.max],
    'ask_spread':[np.sum, np.max],
    'total_volume':[np.sum, np.max],
    'volume_imbalance':[np.sum, np.max],
    "bid_ask_spread":[np.sum,  np.max],
}
```

4.2 trade_train

Firstly, we calculate log return trade price, and add feature called amount which is defined as price * size. For each feature in trade_train, we further calculate their corresponding statistics including summation, maxium and minimum in 5 times window.

4.3 Realized Volatility Statistics

Realized volatility is defined as the squared root of the sum of squared log returns, we compute the realized volatility for different time windows, then we group by each stock_id and time_id to compute mean, standard deviation, maximum and minimum for realized volatility.

4.4 tau features

4.5 K-means clustering aggregation

We use K-means to separate stock id into 7 subgroups. We remove two groups since they have little stock id, for remaining 5 groups, we calculate several statistics including realized volatility, total_volume_sum, trade_size_sum, trade_order_count_sum, price_spread_sum, volume_imbalance sum, bid_ask_spread_sum.

5. Models

5.1. Two Split Strategies

Learning the parameters of a prediction function and testing it on the same data is a methodological mistake: a model that would just repeat the labels of the samples that it has just seen would have a perfect score but would fail to predict anything useful on yet-unseen data. This situation is called overfitting. To avoid it, it is common practice when performing a (supervised) machine learning experiment to hold out part of the available data as a test set X_test, y_test. Therefore, our team uses two split strategies which are KFold and GroupKFold. First, we use KFold split strategy. KFold divides all the samples in k groups of samples, called folds (if k=n, this is equivalent to the Leave One Out strategy), of equal sizes (if possible). The prediction function is learned using n-1 folds, and the fold left out is used for test.
Second, we use GroupKFold split strategy. GroupKFold is a variation of KFold which ensures that the same group is not represented in both testing/validation and training sets. For example, if the data is obtained from different subjects with several samples per-subject and if the model is flexible enough to learn from highly person specific features it could fail to generalize to new subjects. GroupKFold makes it possible to detect this kind of overfitting situations.

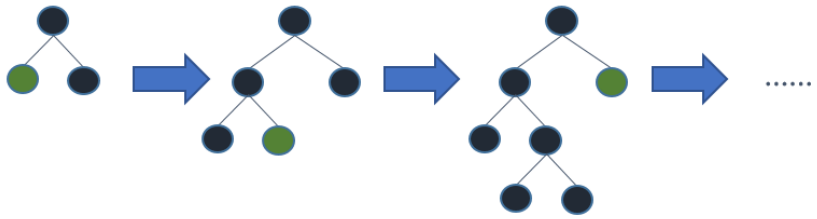
5.2. Three Algorithms

5.2.1. LightGBM

LightGBM is an advanced implementation of gradient boosting, and it uses a more regularized formalization to control over-fitting which gives a better performance within the model. It has optimization in speed and memory usage, it uses histogram-based algorithms [4, 5, 6], which bucket continuous feature (attribute) values into discrete bins. This speeds up training and reduces memory usage.The advantages of histogram-based algorithms reduced cost of calculating the gain for each split, use histogram subtraction for further speedup, Reduce communication cost for distributed learning.

It uses leaf-wise tree growth strategy. It will choose the leaf with max delta loss to grow. Holding leaf fixed, leaf-wise algorithms tend to achieve lower loss than level-wise algorithms.

Leaf-wise may cause over-fitting when #data is small, so LightGBM includes the max_depth parameter to limit tree depth. However, trees still grow leaf-wise even when max_depth is specified.



Leaf-wise tree growth

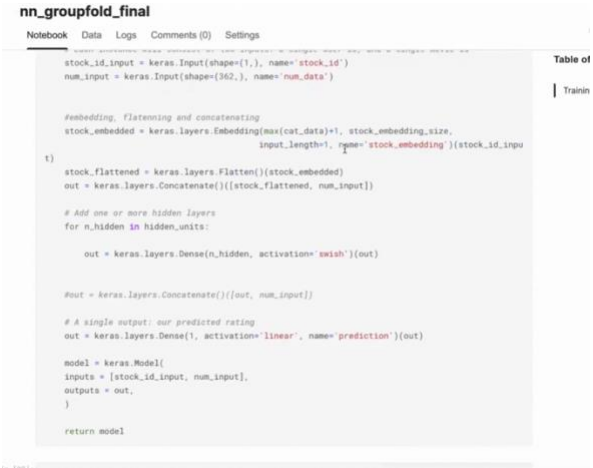
The LightGBM tree has optimization in distributed learning. It is capable of feature parallel and data parallel. The LightGBM doesn't need to communicate for split result of data since every worker knows how to split data. And #data won't be larger, so it is reasonable to hold the full data in every machine. The LightGBM reduce communication cost of data parallel in LightGBM, LightGBM uses histogram subtraction to speed up training. Based on this, we can communicate histograms only for one leaf, and get its neighbor's histograms by subtraction as well.

We first use Optuna framework to optimate the hyperparameters, then we use the optimizated hyperparameters on LightGBM, we change the initial seed in LightGBM. The following table lists the work when we use TabNet to build a model.

Tuning parameters	explanation	values	rmpse	Submission score
Kind,n_fold,seed	Kind: the kind of fold, its values is k(kfold) or g(groupfold) N_fold:the number of fold Seed: initial seed	K,5,105	0.19509	0.20344
Kind,n_fold,seed		G,5,105	0.21539	0.20228
Kind,n_fold,seed		K,10,105	0.19389	0.20348
Kind,n_fold,seed		G,10,105	0.22278	0.20810
Kind,n_fold,seed		K,5,200	0.19498	0.20410
Kind,n_fold,seed		G,5,200	0.21666	0.20233
Kind,n_fold,seed		K,10,200	0.19107	0.20300
Kind,n_fold,seed		G,10,200	0.22203	0.20677

We choose the highlighted models.

5.2.2. Neural Network



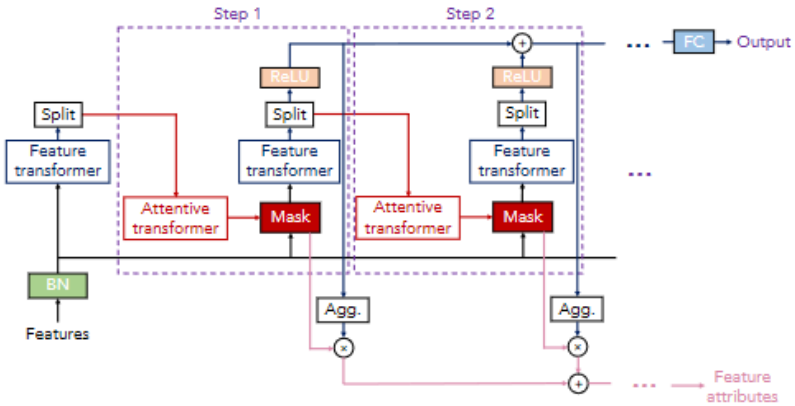
The neural network is consisted of several embedded layer and a flatten layer, several swish layers as the hidden layers and a linear layer as the out layer. We first train the model with datas then inference with the trained model. We choose the learning rate as the hyperparameter, and we use various learning rates to train the model. The following table lists the work when we use NN to build a model.

Learinin rate	Rmpse`	score
0.003	0.21404	0.19985

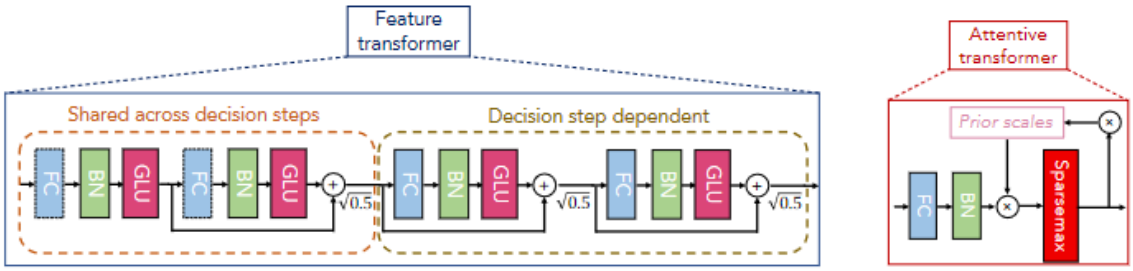
0.004	0.21270	0.19956
0.005	0.21284	0.19920
0.006	0.21406	0.20048
0.007	0.21315	0.20055

5.2.3. TabNet

TabNet is basically a type of neural network using attention mechanism (sequential attention to be exact) on tabular data to learn and make predictions. Besides, TabNet outperforms other neural network and decision tree variants on a wide range of non-performance-saturated tabular datasets and yields interpretable feature attributions plus insights into the global model behavior. The following is an explanation of TabNet's basic architecture.



TabNet encoder architecture



TabNet encoder composed of a feature transformer, an attentive transformer and feature masking. A split block takes and divides the preprocessed representation to be used by attentive transformer in subsequent step as well as overall output which we get through aggregating masks. This process continues by number of steps, each step composed of attentive transformer, mask, feature transformer and splits. Number of steps is a hyperparameter where we can experiment on it. Each step will have their own weight at the final classification.

Feature transformer is a multi-layer network (including FC, BN and GRU's) some of these layers will be shared across every step while some of them are treated locally. The number of

independent and shared layers are hyperparameter too and will have effect on your final predictions.
Once features have been preprocessed and transformed they passed into the attentive transformer and mask. Attentive Transformer includes a FC, BN and Sparsemax Normalization. Therefore, this block gains the information by using prior scales. In this step model learns how much each feature has been used before the current decision step. With mask model focuses on the important features and uses them.
Enough details I guess, in short, we can use full power of the neural networks while keeping the interpretability which is pretty important for tabular data.
The following table lists the work when we use TabNet to build a model.

n_d is the number of the dimension of independent and shared layers, n_a is the number of attentive transformer layers, n_steps is the number of steps.

Tuning parameters	Explanation	Values	RMSPE	Submission score
n_d,n_a,n_steps		8,8,1	0.20860	0.19900
n_d,n_a,n_steps		10,10,1	0.20898	0.19833
n_d,n_a,n_steps		12,12,1	0.20617	0.19808
n_d,n_a,n_steps		13,13,1	0.20812	0.19827
n_d,n_a,n_steps		14,14,1	0.20660	0.19786
n_d,n_a,n_steps		15,15,1	0.20721	0.19966
n_d,n_a,n_steps		16,16,1	0.20751	0.19806
n_d,n_a,n_steps		8,8,2	0.21101	0.20101
n_d,n_a,n_steps		10,10,2	0.21011	0.19948
n_d,n_a,n_steps		12,12,2	0.20924	0.19881
n_d,n_a,n_steps		14,14,2	0.20893	0.19955
n_d,n_a,n_steps		16,16,2	0.20849	01.9852

At last, we choose the hyperparameters highlighted in yellow as our ultimate TabNet model.

6. Averaging Based Ensemble Methods
Ensemble methods are techniques that create multiple models and then combine them to produce improved results. Ensemble methods usually produces more accurate solutions than a single model would. This has been the case in a number of machine learning competitions, where the winning solutions used ensemble methods. Therefore, we use simple averaging method and weighted averaging method.

The following three tables show the weights of each of our models and their corresponding submission scores. We finally chose the model composed of the two weights marked in red in the following three tables as our final submission.

Table 1: The ensemble results of the models based on KFold Split Strategy

LightGBM weight	Neural Network weight	TabNet weight	Submission scores
0.35	0.35	0.30	0.19615
0.3333333333	0.3333333333	0.3333333333	0.19611
0.3294033524	0.3323965678	0.3382000798	0.19610
0.3308999601	0.3338017161	0.3352983238	0.19610
0.30	0.30	0.40	0.19606
0.30	0.35	0.35	0.19602
0.25	0.375	0.375	0.19593
0.15	0.425	0.425	0.19592
0.175	0.4125	0.4125	0.19590
0.21	0.39	0.40	0.19589
0.20	0.40	0.40	0.19589

Table 2: The ensemble results of the models based on GroupKFold Split Strategy

LightGBM weight	Neural Network weight	TabNet weight	Submission scores
0.3333333333	0.3333333333	0.3333333333	0.19716
0.3302785802	0.3338977986	0.3358236212	0.19715
0.30	0.35	0.35	0.1971
0.30	0.30	0.40	0.19709
0.25	0.375	0.375	0.19706
0.20	0.40	0.40	0.19706

Table 3: The ensemble results of six models

GroupKFold			KFold			
LightGBM weight	Neural Network weight	TabNet weight	LightGBM weight	Neural Network weight	TabNet weight	Submission scores
0.1	0.2	0.2	0.1	0.2	0.2	0.19624
0.16667	0.16667	0.16667	0.16667	0.16667	0.16667	0.19638

7. Competition results - Public Leaderboard

We won a silver medal (Public Leaderboard) in the competition. On January 10, 2022, the ranking results of Private Leaderboard will be announced.