Success in any financial market requires one to identify solid investments. When a stock or derivative is undervalued, it makes sense to buy. If it's overvalued, perhaps it's time to sell. Predicting the future trend of stocks based on current information is an important problem for investors.

Professional investors may have lots of experience based on market history. However, human based prediction is limited to a person's knowledge scope and not time efficient for high frequency trading. Machine learning has become a promising choice for stock prediction with the recent success of deep neural networks in modeling sequential data.

Take an example in the Japanese stock market. For KYOKUYO CO., LTD. Which is labeled as 1301 in JPX stock exchange. The Moving average (MA) for daily closing price is shown in the blue curve. Suppose we are in 1/31/2017 and the market is already closed. The best strategy is to sell your stocks tomorrow since the price may drop on the day after tomorrow. Consider the situation that we don't know the price for tomorrow and the day after tomorrow. The price data we can use is shown in the blue curve. Besides price data, we have a dataset called options. This dataset represents the market expectation or some statistical information for stocks. Based on the information we have; we expect the machine learns a function

$$f(histrocial_{data}) = sell$$

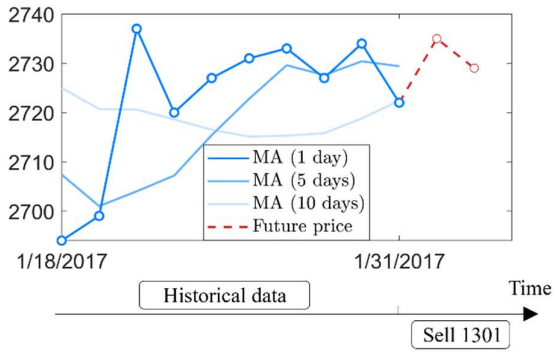


Figure 1: Illustration for stock price prediction. On 1/31/2017, the historical close price with different Moving Averages (MA) is shown in blue. The future price is shown in red.

Let's take another example to consider what happens when there are multiple stocks. Suppose there are 5 individuals stocks labeled from A to E, the changing price ratio between the day after tomorrow and tomorrow is defined as

$$Real\ return = \frac{Close\ price\ (t+2) - Close\ price\ (t+1)}{Close\ price\ (t+1)}$$

	Stock A	Stock B	Stock C	Stock D	Stock E
Real return	-20 %	-10 %	0 %	10 %	20 %
Best strategy	Sell	Keep	Keep	Keep	Buy

Ideally, we should sell the stock if the price tends to decrease and buy the stock if the price tends to increase. However, either the close price at tomorrow or the price the day after tomorrow is not known. In this situation, we want to buy some stocks with relatively high expectations. Similar to the selling process.

Tree model based stock rank prediction

Convert the equation into a machine learning regression problem. On the left hand side, the historical data could contain multiple days with lots of different features. We only use data from the last day to simplify the problem. For example, if we are on 1/31/2017 and the market is already closed. We use the open price, high, low, close price and volume on that day as input to give strategy.

On the right hand side, we want to give a strategy based on the changing rate between tomorrow and the day after tomorrow. Like the above example, the changing rate is -0.0021. The negative sign indicates the price will drop, so we want to sell the stock soon.

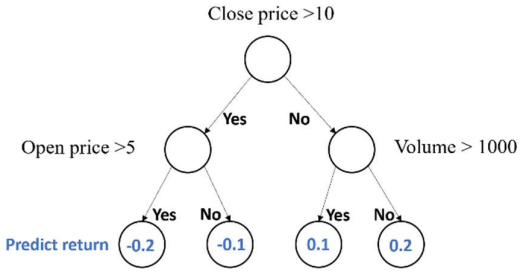
By combining the left hand side and right hand side, we get one training data. In reality, we use data from the beginning of 1/2017 to the end of 2/2021 that contains more than 1000 trading days with 2000 stocks. Valid at 2/2021 to 12/2021 contains 200 trading days.

Now we move to the machine learning model. Before introducing the algorithm, we start with how the decision tree works for regression problems. Convert stock ranking problem into a mathematical function we get

$$f(open, high, close, low, volume) = return$$

To fit such function, we use a machine learning method called decision trees. A decision tree is a tree like structure in which each node contains some tests (except for the leaf node), each branch represents the outcome of the test, and each leaf node represents the final answer. The paths from the root to leaf represent classification rules.

For example, for the tree shown in below, if our input is open=1, high=11, low=1, close=11 and volume=900. Then our final predicted return is -0.1. In reality, the tree may contain multiple nodes and become more complicated.



We select one leaf node for each step to build a decision tree for the regression problem. Then, split this node based on the greedy algorithm to minimize the mean square loss. Continue do this until all the nodes meet some requirements.

Using a single tree very easily overfits the training data. Therefore, we use one algorithm called LightGBM. This algorithm is based on gradient boost, which uses multiple trees to avoid overfitting. For the full description of this method, please see [1]. Here we summarized some keep points for this algorithm.

(1) In LightGBM, when computing the splitting points, we use an algorithm called Gradient-based One-Side Sampling (GOSS) that concentrates on the large distance data. This sampling method works faster than a linear scan.

(2) After one tree is finished, we will compute the residual as training data for the next tree. This process is called gradient boost.

(3) In the training process, we stop the training when the validation accuracy does not increase.

Deep learning based stock rank prediction

The procedure above has one problem since we do not consider how stocks interact. In reality, the relationship between stocks is important. For example, when people find Tesla has a battery problem and the price for Tesla goes down, then the whole market expectation will drop for electric cars.

To solve this problem, we proposed a neural network structure called a stock relation neural network [2]. This neural network contains 3 parts. The first part is Long short-term memory (LSTM). This part is used to extract features from the time series. The second part is Graph neural network (GNN). We design this neural network based on hypergraph attention. This part will combine the feature from stocks together with its related neighborhood. The third part is multi layer perceptron (MLP). The output is the price for each stock. Finally, based on the predicted price, we give the best plan for each day.

In the following sections, we will describe each of the parts in detail. Finally, we present a numerical test. After this neural network is trained based on more than 1000 trading days, we implement this neural network to testing datasets and compute the Sharpe ratio.

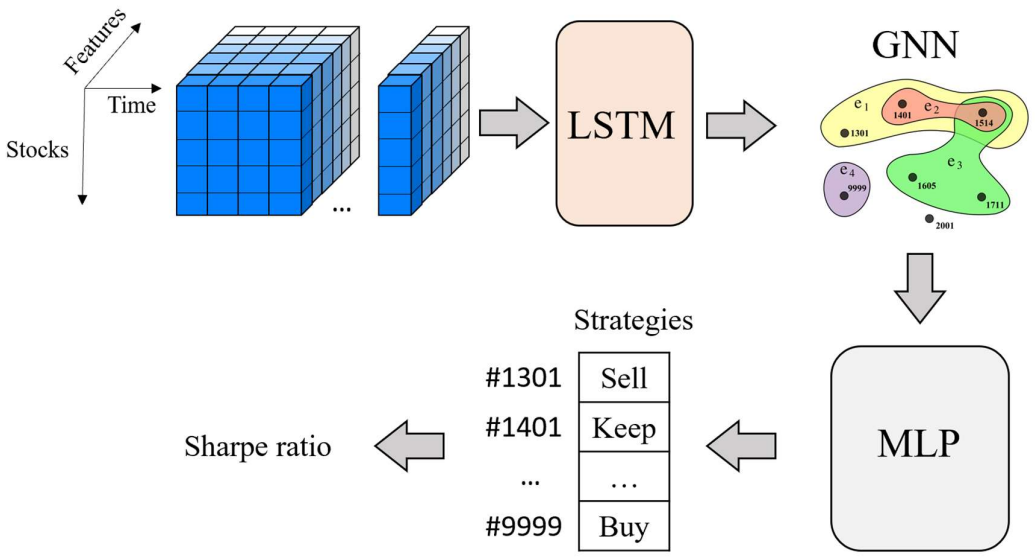


Figure 2: Relation-based stock ranking prediction. The neural network contains three parts: LSTM, GNN and MLP. The final output is the predicted close price.

Feature selection and LSTM

In the beginning, we discuss how to encode the historical feature and input them into the neural network. The final target is to predict the close price in the future. We expect that the close price in recent days may have an important effect on the future price.

In financial problems, simple predictions based on the price data may have overfitting problems due to the high noise and sensitivity for the dataset. In this work, we also use the data from options. This dataset measures what is the average market expectation for many important features, such as volatility, volume, and put-call ratio.

Recurrent neural network is one type of neural network used in time series prediction. In this work, we use the Long Short Term Memory (LSTM). LSTM is one kind of recurrent neural network. LSTM networks have been widely used to process sequential data, such as natural language, voice, and video. The input of the LSTM is the time series x. After following operations, it will output hidden states which encode the important feature for the input as h. The operations are [3]

$$
\begin{aligned}
i_t &= \sigma(W_{ii}x_t + b_{ii} + W_{hi}h_{t-1} + b_{hi}) \\
f_t &= \sigma(W_{if}x_t + b_{if} + W_{hf}h_{t-1} + b_{hf}) \\
g_t &= \tanh(W_{ig}x_t + b_{ig} + W_{hg}h_{t-1} + b_{hg}) \\
o_t &= \sigma(W_{io}x_t + b_{io} + W_{ho}h_{t-1} + b_{ho}) \\
c_t &= f_t \odot c_{t-1} + i_t \odot g_t \\
h_t &= o_t \odot \tanh(c_t)
\end{aligned}
$$

In Figure 3, we show LSTM encode feature for stock 1301. In deep learning we do the parameter sharing for all different stocks. In other words, the batch size equals the number of stocks.

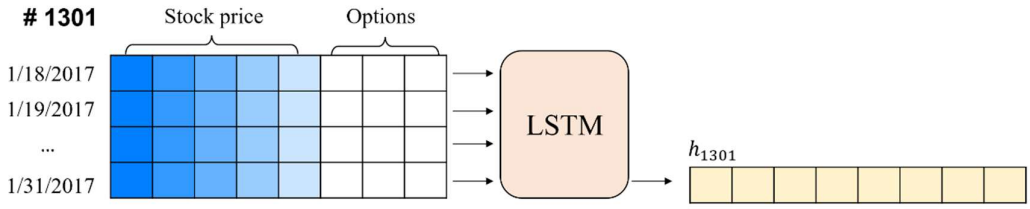


Figure 3: Illustrate the feature selection and LSTM for stock 1301. We input five features based on close price and three features from options. LSTM will map the input states x to h.

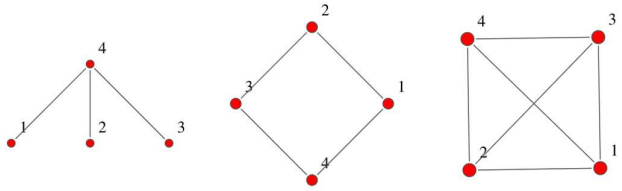
For more technical detail, the number of hidden states is 32 and the length of the time series input is 8. Due to the appearance of the moving average, we encode more than 30 days of data in the hidden state.

Hypergraph

In this first section, we discuss how to use LSTM to extract features. However, each stock is still separated when input to the LSTM. In this section, we start to consider the relation information. There are many ways to consider the effect from other stocks. Such as correlation matrix, use unsupervised learning to do the clustering.

In this work, we consider building a relational graph based on current information between companies. To fully describe the method, we start from the based term in the graph. The graph contains nodes and edges. For example, the undirect graph shown below contains four nodes.

To represent an undirect graph together with its edges. We introduce the adjacency matrix. This matrix with dimension n*n where n is the number of nodes. If two nodes i and j are connected by one edge. The matrix element in position ij is 1. If two nodes are not connected, then the corresponding matrix element is 0. In figure 5, we show three different graphs with the number of nodes equal to 4. By examining the adjacency matrix, we can mathematically represent a graph.



$$\begin{pmatrix} 0 & 0 & 0 & 1 \\ 0 & 0 & 0 & 1 \\ 0 & 0 & 0 & 1 \\ 1 & 1 & 1 & 0 \end{pmatrix} \quad \begin{pmatrix} 0 & 1 & 0 & 1 \\ 1 & 0 & 1 & 0 \\ 0 & 1 & 0 & 1 \\ 1 & 0 & 1 & 0 \end{pmatrix} \quad \begin{pmatrix} 0 & 1 & 1 & 1 \\ 1 & 0 & 1 & 1 \\ 1 & 1 & 0 & 1 \\ 1 & 1 & 1 & 0 \end{pmatrix}$$

Figure 4. Undirect graph with corresponding adjacency matrix. The image taken from [4].

The adjacency matrix is enough if the graph only contains two-body interactions. There could exist multiple nodes that interact together. For example, a couple of companies may have sold electric cars and they may relate together. We call it a hypergraph if one graph contains more than two body interactions. We show a simple hypergraph below.

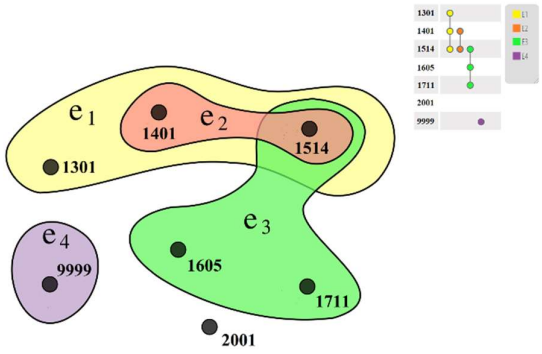


Figure 5: Illustration for hypergraph. This hypergraph contains six nodes. Suppose each node corresponds to a real stock id. Then in this graph, the first edge contains 3 nodes 1301, 1401 and 1514 labeled as yellow. The second edge contains 2 nodes 1401 and 1514 labeled as orange.

To represent a hypergraph, we expand the idea from the adjacency matrix and create a matrix for each edge. Finally, we will end up with a 3D matrix. This matrix is also called tensor representation for a hypergraph. In the later we call this matrix A.

Use Figure 5 as an example. The tensor representation contains a matrix with dimension 6*6*4. The first and second dimensions equal the number of nodes in this hypergraph. The third dimension is equal to the number of edges in this hypergraph. Use the first and the second edge examples, suppose we labeled the node as [1301, 1401, 1514, 1605, 1711, 2001, 9999], then the matrix representation for these two edges are

$$A[:,:,0] = \begin{pmatrix} 1 & 1 & 1 & 0 & 0 & 0 \\ 1 & 1 & 1 & 0 & 0 & 0 \\ 1 & 1 & 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 \end{pmatrix}, \quad and\ A[:,:,1] = \begin{pmatrix} 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 1 & 1 & 0 & 0 & 0 \\ 0 & 1 & 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 \end{pmatrix},$$

In the implementation, we have 2000 stocks, correspond to 2000 nodes in our hypergraph. We choose three pieces of information to change whether two stocks belong to the same edge. We use markets, products, and sectors. In total, there are 44 edges.

Graph Neural Network

The idea of Graph Neural Network (GNN) is similar to the Convolutional Neural Network (CNN). GNN tries to combine the information based on graph edges, and CNN combines

the information between nearby pixels. The difference is that the number of edges from certain nodes is not fixed, so we can not introduce some kernel with fixed size.

To solve this issue, GNN usually combines information from nearby nodes by adding them or performing some average. Recently, the attention mechanism is widely used in self-supervised learning, such as BERT in natural language processing [5].

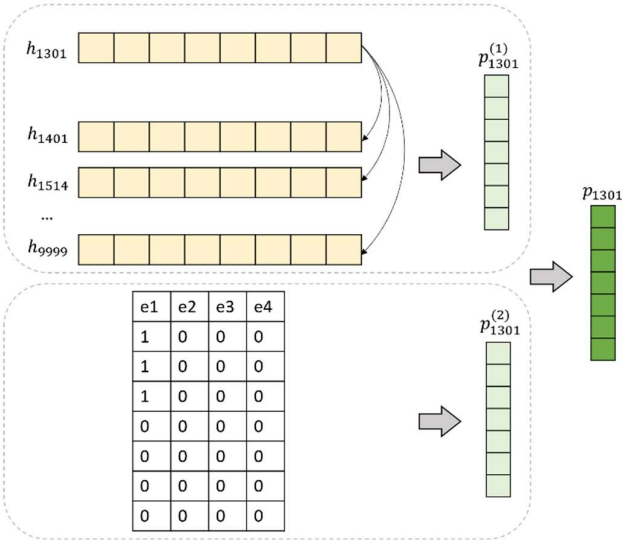


Figure 6: Illustration for hypergraph attention structure. After LSTM, we get one feature vector for each stock labeled as h. For each stock, we compute two probabilities called $p^{(1)}$and $p^{(2)}$ finally use cross entropy as normalization.

Introduce W_{target} and W_{query} for transformation matrix to transform hidden state (with dimension 32) to a number.

$$p_{ij}^{(1)} = W_{target}h_i + W_{query} * h_j$$

While $p^{(1)}$ means the relationship between hidden states. This is similar to the attentions structure used in the transformer. However, this information is purely gotten from hidden states from LSTM and it does not include any prior information from the hypergraph. We introduce another transformation from the relation matrix for each node. We want to know the probability of other nodes nearby.

$$p_{ij}^{(2)} = \sum_k A_{ijk}W_k$$

Here k represents the index of the edge and A is the 3D tensor representation for the hypergraph. W_k means how important a certain edge. For example, suppose for companies selling product1, their relationship is more important than companies selling product2. We should have a large W_k for product 1. In other words, our neural network can solve when the connection strength is not equal.

Finally, combine all the things and do softmax normalization. Then, we can get the probability to represent the importance of other nodes for current nodes. This probability is called p.

After this probability, we concatenate the hidden states for certain nodes together with the weighted sum from other nodes. We can expect that these hidden states currently include information from themselves and nearby nodes.

Multi layer perceptron and loss function

In the third part, we use multi layer perceptron. The neural network input features from per stock and weighted features from nearby stocks. The output is the predicted price with one dimension. The structure for the neural network is shown below.

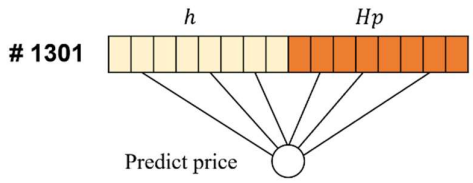


Figure 7: Illustration for the multi layer perceptron structure. The input with dimension 64 contains information per stock and information from the most related stocks. After a fully connected layer, the output with one dimension.

After the predicted price is known, we can compute the predicted return as the changing rate by using the following formula.

$$Predict\ return = \frac{Predict\ close\ price\ - current\ close\ price}{current\ close\ price}$$

Before we move on, we give an important note here. For this problem, we assume that we design strategy at a certain day when the market is already closed. As shown from Figure 1. The computation for the real ratio requires minus and divided by the close price tomorrow. However, here we divided the close price today. Therefore, the predicted close price does not correspond to any real number.

	Stock A	Stock B	Stock C	Stock D	Stock E
Real return	-20 %	-10 %	0 %	10 %	20 %
Best strategy	Sell	Keep	Keep	Keep	Buy
Predict return	-5 %	-22 %	3 %	18 %	15 %
Predict strategy	Keep	Sell	Keep	Buy	Keep

To judge the predicted price, we need a loss function. In this work, the loss function contains

two parts. Mean square error (MSE) and Ranking loss. The total loss function writes as

$$L = L_{MSE} + \alpha L_{Rank}$$

where \alpha=0.01. For these two loss functions, we have

$$L_{MSE} = \left\| r_{real} - r_{predict} \right\|^2$$

$$L_{Rank} = \sum_i \sum_j \max\left[0, -(r^i_{real} - r^j_{real})(r^i_{predict} - r^j_{predict})\right]$$

Where r_real and r_predict are the real and predict return.

Sharpe ratio

The prediction strategy may not be the best. We need to use another metric to evaluate the performance of our model. Suppose we have 2000 stocks. For each trading day, we want to buy the top 10 % of stocks and sell the bottom 10 % of stocks. Besides, for the top 10 % stocks, we may put more money if the predicted return is relatively larger.

We introduce two variables. S_up is computed based on the top 200 stocks with the highest predicted return.

$$S_{up} = \frac{\sum_{i=1}^{200}(r_{up_i} * linearfunction(2,1)_i)}{Average\ (linearfunction(2,1))}$$

where a linear function from 1 to 2 represents the different weights for different predicted returns. S_down is computed based on the top 200 stocks with the lowest predicted return.

$$S_{down} = \frac{\sum_{i=1}^{200}(r_{down_i} * linearfunction(2,1)_i)}{Average\ (linearfunction(2,1))}$$

Finally introduce daily spread return

$$R_{day} = S_{up} - S_{down}$$

Now let's take an example how this is computed. Still use stock in the table as an example, and we want to buy the top 1 stock with expected high return and sell the bottom 1 stock with expected low return.

From the table, the predicted value for stock B is the lowest, and the predicted return for stock D is the highest. Then S_up is equal to the real return for stock D and S_down is the real return for stock B. The normalized factor is 1. Therefore, for today the R is equal to 0.2.

If the model works very well on the close price prediction. The predicted top 200 stocks with high returns should have a relatively high return. S_up will be larger than S_down and the daily spread return R will be a positive number. To further characterize the model

performance, we introduce the Sharpe ratio.

$$Sharpe\ ratio = \frac{Average\ (R)}{STD(R)}$$

Where R is a time series for some time interval. This ratio measures the performance of an investment such as a security or portfolio compared to a risk-free asset. For the detail description please see [6].

Real implementation (Decision trees)

The decision trees are trained based on data from the beginning of 1/2017 to the end of 2/2021. Totally contains more than 1000 trading days. Valid at 2/2021 to 12/2021 contains 200 trading days and test on the following 100 trading days. In Figure 8, we plot the daily spread return R versus time.

In this testing dataset, we can get a daily Sharpe ratio 0.2991. Convert to yearly shape ratio is around 5. According to the standard metric for the Sharpe ratio [7]. This value can be treated as very good.

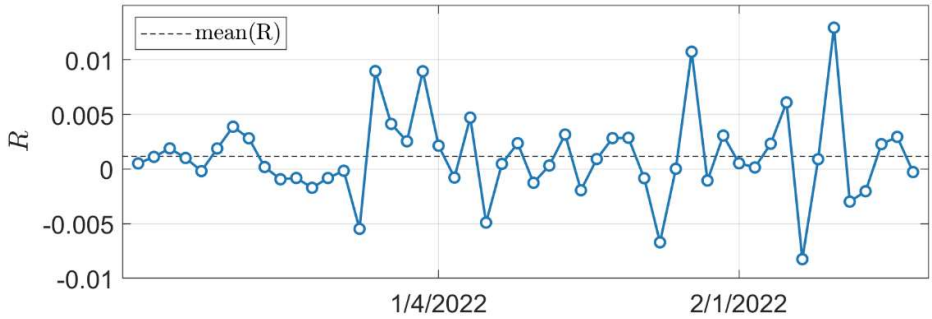


Figure 8: Daily spread return R versus time in test dataset. The mean value is approximately 0.0012. The daily Sharpe ratio is 0.2991.

Real implementation (Neural network)

The neural network is trained based on data from the beginning of 1/2017 to the end of 2/2021. Totally contains more than 1000 trading days. Valid at 2/2021 to 7/2021 contains 100 trading days and test on the following 100 trading days. In Figure 8, we plot the daily spread return R versus time.

In this testing dataset, we can get a daily Sharpe ratio 0.3206. Convert to yearly shape ratio is around 5. According to the standard metric for the Sharpe ratio [7]. This value can be treated as very good.

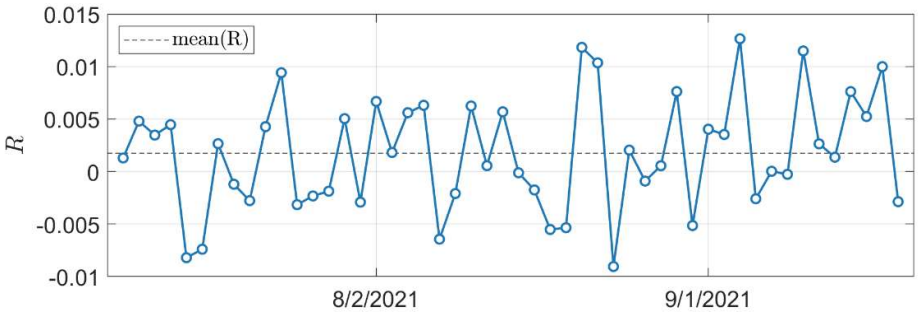


Figure 9: Daily spread return R versus time in test dataset. The mean value is approximately 0.0017. The daily Sharpe ratio is 0.3206.

References

[1]. Ke, Guolin, et al. "Lightgbm: A highly efficient gradient boosting decision tree." Advances in neural information processing systems 30 (2017).
[2]. Feng, Fuli, et al. "Temporal relational ranking for stock prediction." ACM Transactions on Information Systems (TOIS) 37.2 (2019): 1-30.
[3]. Pytorch LSTM document https://pytorch.org/docs/stable/generated/torch.nn.LSTM.html.
[4]. Adjacency matrix https://mathworld.wolfram.com/AdjacencyMatrix.html.
[5]. Devlin, Jacob, et al. "Bert: Pre-training of deep bidirectional transformers for language understanding." arXiv preprint arXiv:1810.04805 (2018).
[6]. https://www.kaggle.com/code/smeitoma/jpx-competition-metric-definition.
[7]. https://www.investopedia.com/articles/07/sharpe_ratio.asp.