Explainable AI model for Stock trading

Seonghyun Ban[1], Dongyoung Choi[2], and Minseung Lee[3]

[1] SungKyunKwan University
2066, Seobu-ro, Jangan-gu, Suwon-si, Gyeonggi-do, Republic of Korea
bansh123@skku.edu
[2] ttiger0114.skku.edu
[3] mkms3580@skku.edu

Abstract. With the rapid development of artificial intelligence (AI), attempts to incorporate it into the financial industry have also increased. Numerous approaches have been proposed to deal with the stock price prediction problem. However, these approaches suffered from difficulties in generalizing the stock market and lack of explanation. Therefore, in order to solve these two problems, we applied a so-called "divide & conquer" approach and model visualization technique. We verified the performance of the AI model in the real-time stock market by calculating the return in mock investment.

Keywords: Divide and Conquer · Explainable AI · t-SNE · transformer · gradient saliency map

1 Introduction

In order to develop an explainable AI model for stock trading, we worked on the project with two key ideas: "divide & conquer" and "model visualization". First, data selection and preprocessing strategies were established by exploring the data distribution using the unsupervised technique such as t-SNE[1]. Second, we constructed a transformer-based classification model and achieved fairly high precision. Third, using Kiwoom API, we link our AI model to the actual stock market in real time, and profitability was verified through mock investment. Finally, we gave explainability and transparency to the model by visualizing on what basis the model makes decisions through the gradient saliency map[2]. Through this process, we achieved fairly high precision(0.6763) and verified this result in mock investment. Also, we succeeded in building a real time model visualization system.

1.1 Related Work

Various algorithms have been proposed to establish a stock trading strategy. Neural network models such as RNN have also been introduced, following genetic algorithms. In addition, reinforcement learning, in which agents recognize the current state and learn how to maximize cumulative rewards within a given

environment, is regarded as an appropriate way for learning the stock market with rapid changes and many interference factors, and is also actively used to establish stock trading strategies. However, these approaches suffered from difficulties in generalizing the stock market and lack of explanation. This project is differentiated from previous works in that we try to solve these two problems.

2 Project design

2.1 Overall scheme

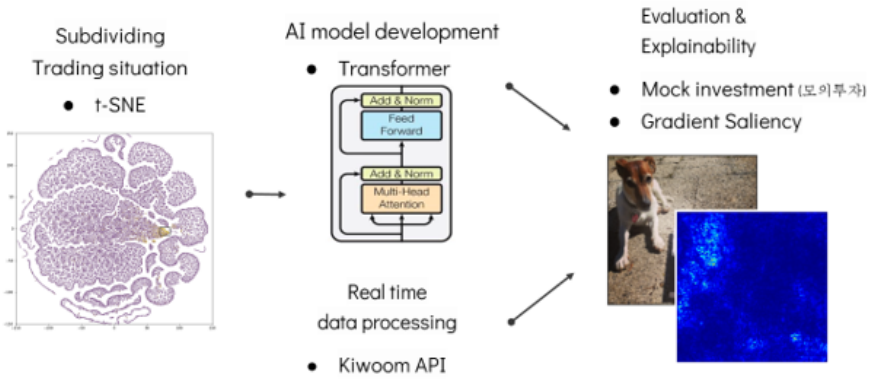

Fig. 1. The overall scheme. This project is consisted of 3 stages, subdividing trading situation, AI model and real time data processing system development, evaluation and visualization.

2.2 Subdividing Trading situation

We have Kosdaq data over the past year and can obtain 40 million data. This vast amount of data can preoccupy favorable conditions for training deep learning models. However, this is rather a big obstacle in the face of a lack of computing resources. If we try to use the whole data of population for one model, realistic restrictions are inevitable to the experiment in trying multiple hyperparameters or various model architectures within a limited time. To effectively solve this problem, we sought to explore partial data distribution from the population. The partial data distribution is smaller in number than population but has to effectively train models. It is a strategy in the data collection phase of this project called "divide and conquer".

We had trained autoencoders using the entire data at the beginning of the project. A well-trained autoencoder can express the entire data with smaller

information on smaller dimensions. We used this advantage to effectively compress high-dimensional input data into low-dimensional data. Next, we cluster this compressed data using t-SNE and specify these clusters using DBSCAN. Finally we select the most fluctuating data cluster, and use this partial data distribution as training data for the AI model. However, there are three problems with the above method.

1. **Constraints on time and space.**
   It took about five hours to train the auto encoder. We tried to use a CPU-based t-SNE of Scikit-learn, which took a total of 5 hours to fit the t-SNE model for about 400,000 data. Despite reducing the latent dimension to 4, a memory shortage problem occurred.

2. **Effectiveness.**
   We labeled each cluster using DBSCAN and then sorted them in the ascending order of density of yield and loss labels. However, the reduction of data was too large(-94.7%), while the rate of increase in fluctuation was not large enough (+3.3% and +4.1%).

3. **Feasibility in a real-time transaction.**
   Since the data we choose belongs to a specific cluster of latents from the encoder part of the pre-trained autoencoder, it is a very tricky problem to find out which transaction data belongs to our target cluster.

	Number of data	Loss density	Yield density	Else
Before	380,000	5.2%	4.6%	90.2%
After	20,000 (-94.7%)	8.5% (+3.3%)	8.7% (+4.1%)	82.8% (-8.1%)

Table 1. The effectiveness of t-SNE clustring using latents from econder

Therefore, we establish a simpler and more effective strategy. The revised strategy has to be done faster and ensure that the number of data after filtering is large enough to train the trading AI model. To this end, we explored how t-SNE can be applied directly to the raw data, and found a machine learning library which supports GPU acceleration. Therefore, after establishing an environment for a gpu-accelerated library cuml in a colab, 1 million data randomly sampled from the population and we used t-SNE over the sampled data. It took only about two minutes each time of t-SNE clustering, and we tried to find the most fluctuating data distribution by adjusting the two hyperparameters of t-SNE, perplexity and the number of neighborhood.

As a result, we tracked the distribution where the most fluctuating data (color of yellow) is concentrated to find the most prominent features of the data. As a result, it was found that their average transaction volume for 6 minutes

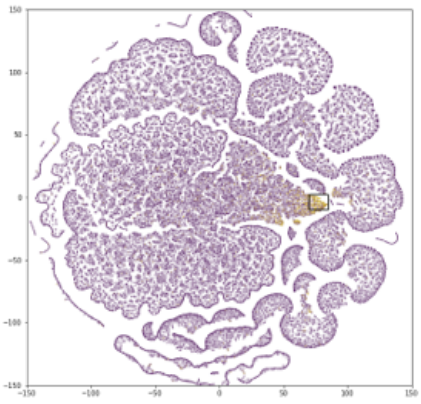

Fig. 2. t-SNE result of 1 million data uniformly sampled from population

was very large compared to other data distributions. It was also confirmed that they have relatively high volumes by measuring the z-score of their average of transaction volumes. (z-score = 9.09)

Therefore, we assumed that the transaction volume is the variable that has the greatest influence on the price fluctuation. We select the data between 9 am and 10 am, which is the time zone with the largest transaction volume.

2.3 AI Model

2.3.1 Reasons of design choice Recently, self-attention-based architectures, in particular Transformers [3], have become the model of choice in natural language processing (NLP). Thanks to Transformers' computational efficiency and scalability, it shows great performance in other fields such as computer vision and time series tasks. The stock trading tasks must be able to have an advantage on the transformer. Thus, we adapt transformers to construct an AI-based stock trading model. For visualizing the important weights of the model decision, we make a saliency map using the back-propagated gradient. Although we can exploit the attention weights which can be obtained in the transformers for the visualization, an attention value represents the importance weight over each time. Thus, attention weight itself cannot tell us which feature of the data on same time point is important. Since we want to know which features have the greatest impact on decision making, we don't adapt the attention-related visualization method. Instead, a traditional visualization technique, gradient saliency map is used to show the importance weights for the model's decision making by pixels.

2.3.2 Model structure The model was composed of a column-wise fully connected embedding layer, a transformer encoder layer, and a classification head. Empirically, we decide the architectural details of the model which shows the

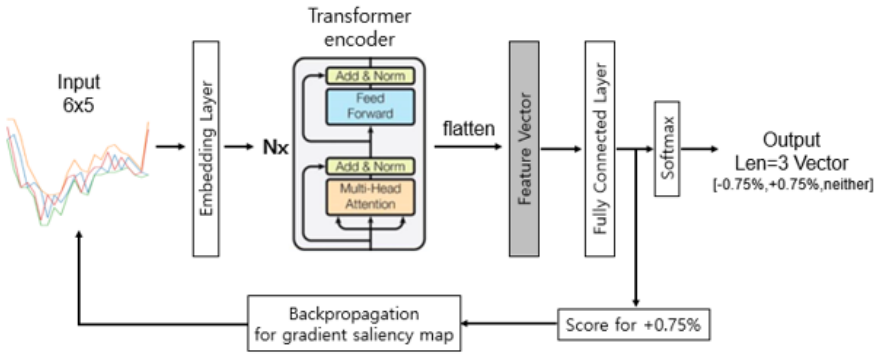

Fig. 3. The overall structure transformer-based classification model and backpropagation flow for calculating gradient saliency map.

best performance in test data. After model training is completed, we forward input data into the model, and backpropagate the gradients over the score corresponding to the target yield score to obtain a gradient saliency map.

3 GUI

3.1 Main window

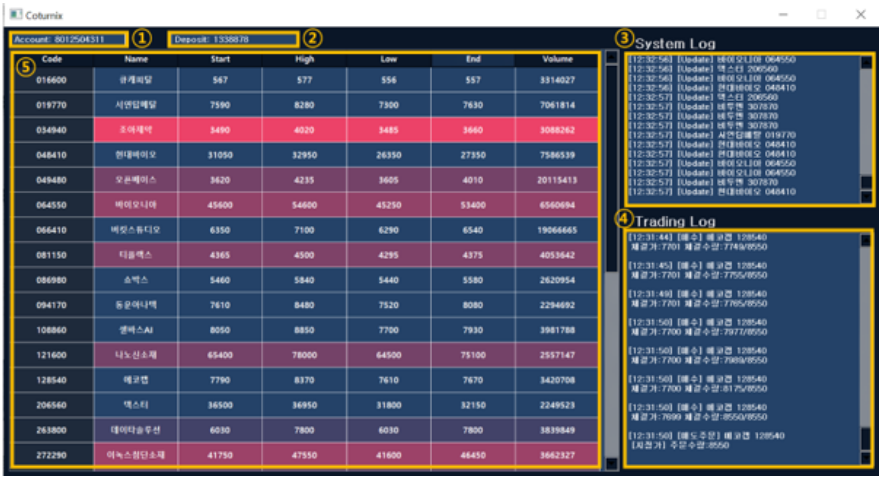

Fig. 4. The main window that displays real-time stock information including several logs

1. **Account number of connected account**
2. **Deposit of connected account**
3. **System log displays subscription and update logs**
4. **Trading log displays trading logs including trading type, code number, trading price and trading volume**
5. **Main table shows code number, name and price information**
   Change background color depending on confidence of decision

3.2 Subwindow

Fig. 5. A subwindow that visualizes price chart and volume histogram with gradients of each data

1. **Price chart and volume histogram so far**
2. **Price chart and volume histogram for only 6 minutes**
3. **Marker size that shows gradient degrees**
4. **Price gradient for 6 minutes**
   Show how important each price data is in making decision
5. **Volume gradient for 6 minutes**
   Show how important volume data is in making decision

4 Implementation

4.1 Preparing dataset for model training

We collected 1 minute candlestick data of KOSDAQ stocks for the recent 1 year. Data selection and preprocessing steps are followed.

1. **Among total 1487 stocks, exclude 300 stocks in order of high sparsity of data due to the small transaction history**
2. **Start, high, low, end price and transaction volume data are selected.**
3. **Transactions from 9 am to 10 am.**
4. **Transactions whose cumulative volume should be 180% greater than the cumulative volume at the same time of the previous day.**
5. **Concatenate candlestick data for six minutes. Thus, input data shape : [6x5]**
6. **Each label of transaction data depends on whether the stock price rises 0.75%, falls 0.75% compared to last end price or neither in the next 12 minutes.**
7. **Price-related features (open, high, low and end) are divided by the start price of the day and subtracted by 1.**
8. **The transaction volume is divided by the average of the transaction volume for the previous 5 days.**
9. **Price-related features are multiplied by 100, and the volume of transactions is multiplied by 0.1 to make the variance similarly.**

Through the above process, 600,428 input data were collected. (shape=[600428, 6, 5])

4.2 AI model server

The AI model server was implemented with PyTorch. When receiving input data from a GUI client, the next price is predicted using a pretrained model, and then three types of packets are transmitted to the client. Stock item codes whose prediction result is upward(+0.75%) is transmitted to the client with a “buy” tag. The confidence of up-prediction and gradient for all data will be transmitted with the ”confidence” and ”gradient” tags, respectively.

4.3 GUI Client

The GUI client was implemented with pyqt5. The client plays three main roles, real-time stock data processing, trading, and visualization. Through the whole process, a client accumulates real-time stock data received from Kiwoom server internally, transmits it to the AI model server after preprocessing it and finally trades stocks based on the AI’s judgements.

The first is that as mentioned above, time and resource consumption are large. It is definitely ideal to train a single model using data under all trading

conditions. However, as can be seen in BERT and GPT-3 in the field of natural language processing, training a very large, precise and well-designed model required to train complex and diverse data distribution, takes a lot of time, and hardware resources such as high-performance GPU and memory.

4.3.1 Real-time data preocess

Fig. 6. The protocol and method to receive real-time data from Kiwoom Server

If the client subscribes to the desired stock using the SetRealReg method to the kiwoom server, the Kiwoom server sends transaction information to the client whenever the sale of the item occurs. The client can check the information received through the OnReceiveRealReg method. The client updates self.TradingInfo dictionary through the received information, where self.TradingInfo stores data per minute in such a way that a new row is added every minute.

4.3.2 Trading process

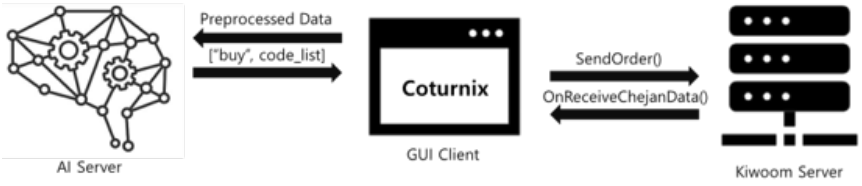

Fig. 7. The overall protocol and method to send a sale order to Kiwoom Server

When the client transmits the preprocessed data packet to the AI server, the AI model transmits the code of upward prediction with a "buy" tag. When receiving "buy" packets, the client requests purchases of the stock at market price using the SendOrder method. The result of requests could be confirmed using OnReceiveChejanData method. A 12 minutes timer starts from request accepted, and after completing purchases a sale order is placed at the target price. If the timer expires or a current price falls below the baseline, the client cancels the previous orders and attempts to sell it at the market price. In order to do distributed investment, the purchase quantity per stock was designated as 50% of the current balance. This was calculated internally because Kiwoom API does not support deposit data following stock transactions. The balance is deducted by the amount of the order at purchase order accepted. If the order is canceled or stocks sold, the transaction price is added to the balance.

4.3.3 Visualizing process

Fig. 8. The visualization process. After receiving confidence and saliency map from AI server, GUI shows the value through main and subwindow.

When an AI model calculates the outputs from preprocessed data, the gradient saliency map of the result is also calculated through backpropagation. The calculated saliency map is transmitted to the client as a "gradient" packet, and the confidence of the upward prediction is also transmitted as a "confidence" packet. After receiving these packets, the client stores each data in self. Gradient and self.Confidence. Confidence is expressed as the background color of the code in the main table, and gradient saliency map is expressed as the marking size in the subwindow.

dim. \ #	1	2	6	12
35	0.6298	0.6432	0.6553	0.6522
70	0.6648	0.6701	0.6763	0.6562
140	0.6443	0.6697	0.6583	0.6581

Table 2. Test set precision on different model capacities # : number of transformer encoder layers, dim. : embedding dimension (hidden dimension)

# of FC layer in embedding layer	Embedding dimension	Hidden dimension	# of transformer encoder layers	# of FC layer in classification layer
2	70	70	6	2

Table 3. Final model configuration

The above example is when the trading volume suddenly rises after the stock price falls in a short term. Looking at the gradient saliency map, we can conjecture that the model also focuses on the sudden rise in trading volume, thinking that the high price at the last point is high hence there is a potential of a price rise. Also, the closing price is still low thus the model thinks it is possible to buy it at a low price.

5 Discussion and Limitation

5.1 Discussion

5.1.1 Gradient saliency map

In order to get an intuition about how the model actually interprets the data, we look at a gradient saliency map for data that the model succeeds in predicting the price increase. As a result, we could observe patterns that could be understood by human common sense.

5.1.2 Average analysis using gradient saliency map

We analyze the data that the pre-trained model predict as 'rising soon' for validation and test data. These are approximately 3000 data, and we infer the overall preferred data shape for the model by averaging them. By averaging the gradients transferred to the data, it is also possible to check where the direction of correction is for higher score. As a result, the high price of the last two minutes and the closing price have a greater gradient on average, and the higher the

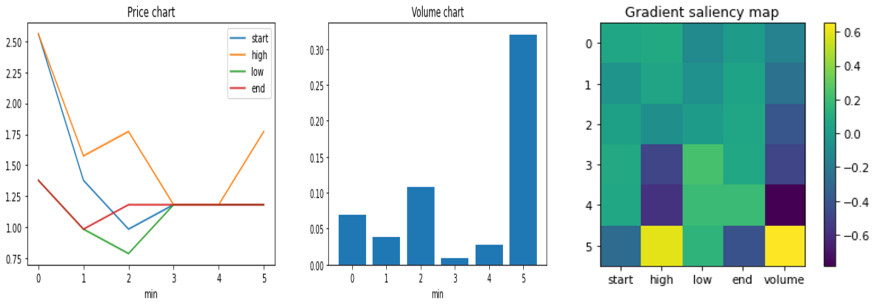

Fig. 9. An example of input data and its gradient saliency map.

high increase rate of the last two minutes and the higher the decline rate of the closing price, the larger the score.

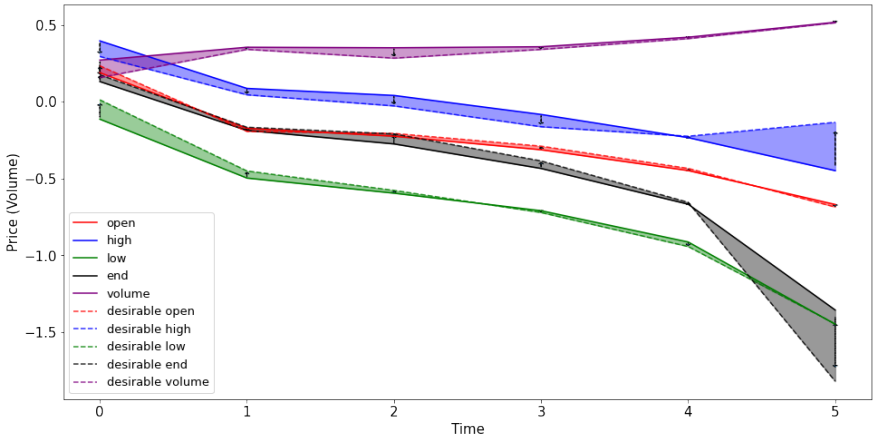

Fig. 10. Solid lines are the average values of data that model predict as +0.75% increasing candidate. Dashed lines are the corrected data with given gradients

5.2 Limitation

5.2.1 Overestimation In mock investment, each of every user's order is always considered to have the highest priority regardless of actual transaction priority. In addition, virtually completed transactions do not affect the actual stock price. These two factors could be advantages, especially to short-term stock trading. Therefore, further evaluation in real investment is needed.

5.2.2 Tax and Fee Korea's stock market has a tax rate 0.23% and a fee rate 0.03% for every transaction. Due to the large amount of tax and fee, even with high precision of model prediction, expected annual profit is not that high. Therefore, further analysis is necessary for mitigating this problem to actually achieve a higher profit rate.

6 Evaluation

6.1 Test set precision

The performance of the model is evaluated by a precision for the test set. Precision means how correctly the model predicts the future stock price arising more than 0.75%. For model training, we use cross entropy loss and Adam optimizer, and the learning rate is set to 0.001 and the batch size is set to 1024. The number of fully connected layers in embedding layers and classification head is set to 2. After determining the basic model structure and training parameters, we have tried to find an optimal configuration by changing the embedding dimension and the number of transformer encoder layers.

6.2 Result of Mock Investment

In order to create a similar environment where the AI model was trained, in mock investment, the client places a sale order at a target price that was 0.75% higher than the average purchase price as soon as the client finished buying. If a current price is 0.75% lower than the purchase price or 12 minutes passes after the purchase, the client sells stocks at market price. Our team tested the model from November 22th to December 5th for 9 am to 10 am in the Kiwoom mock investment. During the test period, 56 transactions were attempted in total, and 39 of them were sold at the target price, and 6 of them expired, and 11 of them were sold at the market price because of a price plunge below baseline. Final empirical precision is 69.94

7 Assessment

Objectives of this project are 1) achieve meaningful profit rate, 2) get 'Explainability'. We achieved fairly high precision(67.63%) in the test set, and verified this result in mock investment (69.64% precision, corresponding to 27% annual profit rate). Although there could be an overestimation problem, this is a very promising result which is beyond our initial expectations. In addition, we successfully developed a real-time visualization system that can show which data is important for the decision of the model. We think that this can give 'Explainability' to our model, so that model could be more reliable. Thus, we strongly believe that our system meets the objectives of the project.

8 Conclusion

Objectives of the project have been successfully achieved, and the final result is beyond our expectation. However, there are still many things to move forward. First of all, we have to get more data. Since this project focuses on explanability rather than maximizing profit, the model is trained with the data which have only five most basic features(4 kinds of price, transaction volume). However, of course, there are still much more features that affect stock prices, and it is necessary to collect various features for a better model. In addition, it seems possible to develop a model for mid-term or long-term trading as well as short-term trading that use only six minutes of data. Next, the validity period of the model. The stock market is not a static environment, but a dynamic environment that changes over time. Therefore, as in this project, using a model trained only with data from the last year without additional updates does not guarantee its validity over time. Thus, to solve this problem, it is necessary to establish a system that continuously updates the model to fit with the recent stock market trends by applying methods such as continual learning. Finally, risk management in real investment. Although the model performance has been successfully verified in the mock investment in this project, as mentioned in the limitation, there could be an overestimation problem for several reasons. Furthermore, as a huge trading volume at once could affect actual market price, the risk of this problem is expected to increase as the operating amount increases. Therefore, in order to ensure the scalability of this project, it is necessary to manage the problem, and we think this is the most important and difficult part.

References

1. Laurens Van der Maaten and Geoffrey Hinton. Visualizing data using t-sne. *Journal of machine learning research*, 9(11), 2008.
2. Karen Simonyan, Andrea Vedaldi, and Andrew Zisserman. Deep inside convolutional networks: Visualising image classification models and saliency maps, 2014.
3. Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N. Gomez, Lukasz Kaiser, and Illia Polosukhin. Attention is all you need, 2017.