Improving Signal Mapping and Utilizing Ensemble Algorithm for Efficient Stock Trading

IIE4122 Final Project | 25.06.16 Mon

TEAM 13 | Kim Wonjun(2022147002), Kim Hyeonjin(2021125085), Jang Seohyun(2021190002)

Background

Consequently, trading systems designed to generate high stock market returns are developed via several supervised learning methods

✓ However, methods based on them make it difficult to adapt to the real-time nature of the stock market (can be noisy and fail to consider the nonlinear and complex nature of stock prices)

Contribution

1. By applying reinforcement learning to stock trading, flexible strategies can be developed to adapt to changing market conditions

2. Existing literatures use price up/down labeling as labels, while this paper uses buy/sell signals trained by DQN

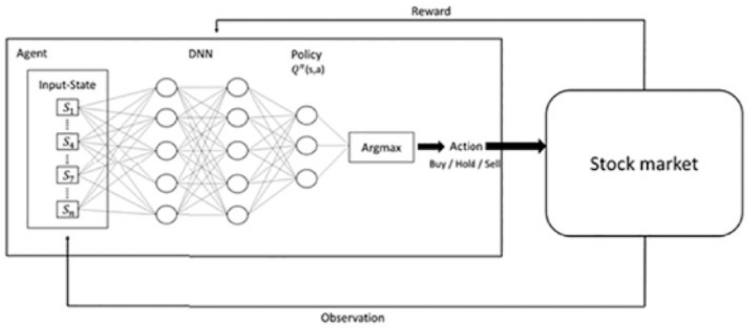
⇩

This study proposes a Deep Q-Network (DQN) Action Instance Selection Trading System (DAIS) to improve the limitations of both supervised learning and reinforcement learning trading systems

Literature Methodology

2) The learning data (buy/sell signal + all technical indicators)
and machine learning algorithm were constructed using instance selection

DQN Training



Date	Close	DQN Action
T-3	260.05	Buy
T-2	265.25	Hold
T-1	255.53	Sell
T	264.24	Buy
T+1	270.35	Hold
T+2	264.21	Sell

1) Gather information about the stock market and extract actions

Instance selection

Date	Feature 1	···	Feature N	DQN Action (Target)
T-4	0.001	···	0.23	Buy
T-2	0.003	···	0.24	Sell
T-1	0.002	···	0.01	Buy
T+1	0.005	···	0.55	Sell

Machine learning models training

Decision Tree / GBM / K-NN
Logisitc Regression / MLP / Randomforest / XGB

Label prediction

3) Use a machine learning algorithm to predict and generate trading signals

Trading simulation

Generate trading signal

Trading

Performance evaluation

Winning rate / Payoff ratio / Profit factor

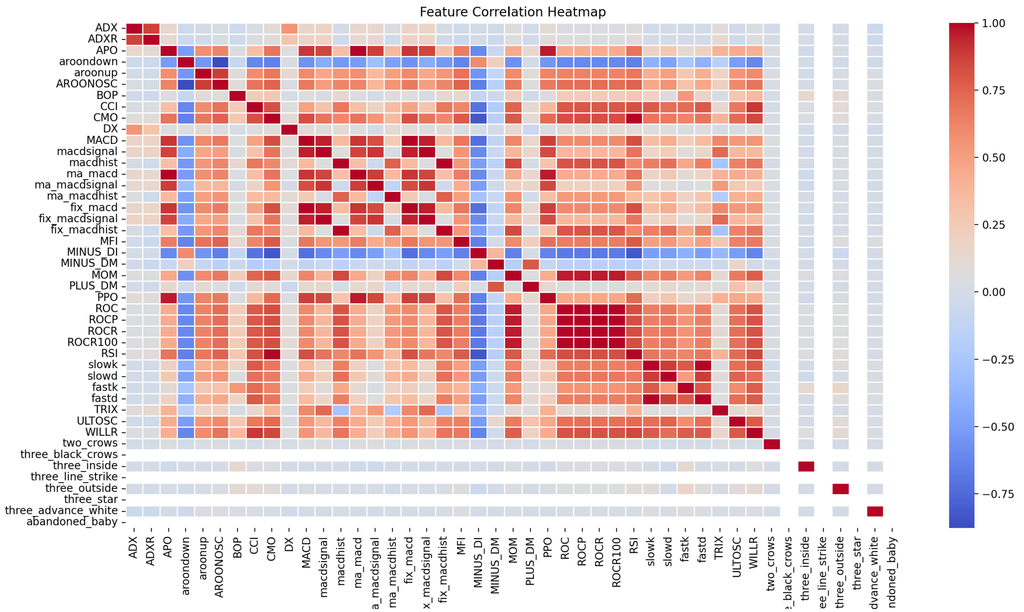
4) Conduct a simulation using the generated trading signal and evaluate its performance

Midterm Contribution

✓ Limitation of Paper:

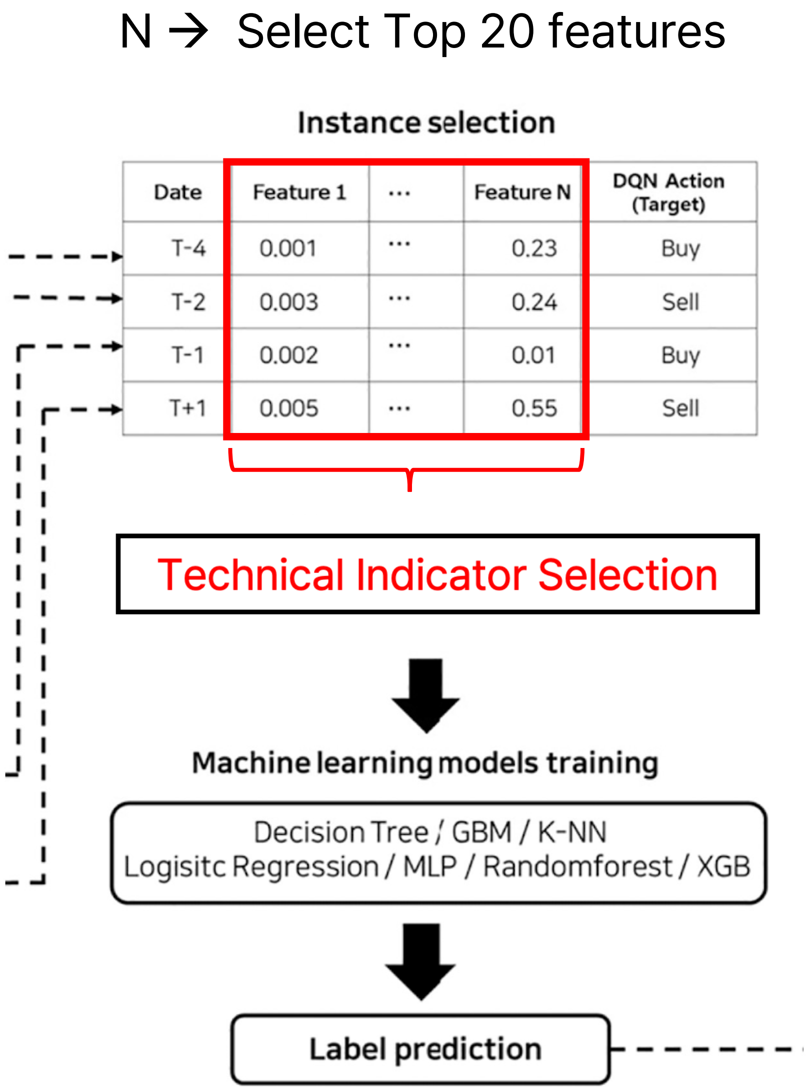
This study had some limitations. First, we arbitrarily selected the technical indicators to construct the reinforcement learning environment. The results may have been different if we had used more technical indicators or significant technical indicators. Second, we used the DQN

✓ Correlations among technical indicators



∴ Technical Indicator Selection is needed.

→ Feature selection is done using 1) Randomforest
2) mRMR



Midterm Results

Table 5
Trading performance by DQN and DAIS trading system.

Model	No. trades	Winning ratio	Payoff ratio	Profit factor	Sharpe ratio
Decision tree	90.30	0.48	1.10 (0.30)	1.11 (0.53)	1.03
GBM	88.27	0.47	1.10 (0.28)	1.10 (0.57)	0.66
knn	78.62	0.49	1.10 (0.40)	1.13 (0.67)	0.92
Logisitic Regression	52.70	0.49	1.12 (0.59)	1.21 (1.93)	0.58
MLP	61.35	0.49	1.12 (0.53)	1.21 (0.89)	0.67
Random forest	76.98	0.49	1.15 (0.34)	1.20 (0.58)	1.09
XGB	89.04	0.49	1.04 (0.28)	1.09 (0.52)	0.93
DQN	84.01	0.46	0.97 (0.45)	0.94 (0.59)	0.38

Note: Values for trading performance are given as average (standard deviation).

Trading Performance of Original Paper

methodology	year	stock_name	No.trades	Win%
pred_logistic	2019~2020	005930_Samsung	33	0.666667
pred_decision	2019~2020	005930_Samsung	98	0.581633
pred_naive (DQN)	2019~2020	005930_Samsung	29	0.758621
pred_randomforest	2019~2020	005930_Samsung	65	0.461538
pred_knn	2019~2020	005930_Samsung	51	0.509804
pred_neural	2019~2020	005930_Samsung	42	0.52381
pred_voting	2019~2020	005930_Samsung	45	0.488889
pred_gbm	2019~2020	005930_Samsung	107	0.336449

Trading Performance of Proposed Methodology (using RF)

- ✓ The **winning ratio** has generally improved compared to the pre-improvement range (0.48~0.49).
- ✓ Several models achieved a **winning ratio above 50%**, suggesting a **higher likelihood of making profitable trades**.

Contribution (1) Improve Signal Mapping

Problem 1. Inconsistency in Reactions to Signal Changes

- Original signal mapping accounted for potential mean reversion due to volatility.
- Yet, it reduces model responsiveness and hinders number of trades.

Improvement 1. Change Action Mapping of [Rise, Drop] to Sell and [Drop, Rise] to Buy

- This enables the agent to respond promptly to directional changes, reducing lag in execution.
- Furthermore, it increases clarity in decisions.

Price Trend Signals		Time t-1: 0 (Drop)	Time t-1: 1 (Steady)	Time t-1: 2 (Rise)
Time t	0 (Drop)	Hold	Sell	Hold
Time t	1 (Steady)	Buy	Hold	Sell
Time t	2 (Rise)	Hold	Buy	Buy

→

Price Trend Signals		Time t-1: 0 (Drop)	Time t-1: 1 (Steady)	Time t-1: 2 (Rise)
Time t	0 (Drop)	Hold	Sell	Sell
Time t	1 (Steady)	Buy	Hold	Sell
Time t	2 (Rise)	Buy	Buy	Hold

Contribution (1) Improve Signal Mapping

Problem 2. Unrealistic Reactions under Continuous Signals

- Since money isn't infinite (2,2) continue buy doesn't make sense in real life situations.
- In actual code simulations, if there is an initial rising signal, the algorithm is not properly trained by buying stocks until resources are lost.

Improvement 2.

- Modify **(2, 2)** to **Hold** in the same way as (0, 0).
- The number of transactions will decrease a little more, but considering limited resources, it is reasonable to buy only in situations such as a definite buy signal (0, 1) or (1, 2).

Price Trend Signals		Time t-1: 0 (Drop)	Time t-1: 1 (Steady)	Time t-1: 2 (Rise)
Time t	0 (Drop)	Hold	Sell	Hold
Time t	1 (Steady)	Buy	Hold	Sell
Time t	2 (Rise)	Hold	Buy	Buy

→

Price Trend Signals		Time t-1: 0 (Drop)	Time t-1: 1 (Steady)	Time t-1: 2 (Rise)
Time t	0 (Drop)	Hold	Sell	Sell
Time t	1 (Steady)	Buy	Hold	Sell
Time t	2 (Rise)	Buy	Buy	Hold

Code for Contribution (1)

```
for i in range(0,limit):
    for e in train_data[i].index:
        try:
            if train_data[i]['label'][e]+train_data[i]['label'][e+1]==0:
                train_data[i]['position'][e+1]='no action'
                print("Nothing at %d" %e)
            elif train_data[i]['label'][e]+train_data[i]['label'][e+1]==2:
                train_data[i]['position'][e+1]='holding'
                print("Hold at %d" %e)
            elif train_data[i]['label'][e] > train_data[i]['label'][e+1]:
                train_data[i]['position'][e+1]='sell'
                print("Sell at %d" %e)
            else:
                train_data[i]['position'][e+1]='buy'
                print("Buy at %d" %e)
        except:
            pass
```

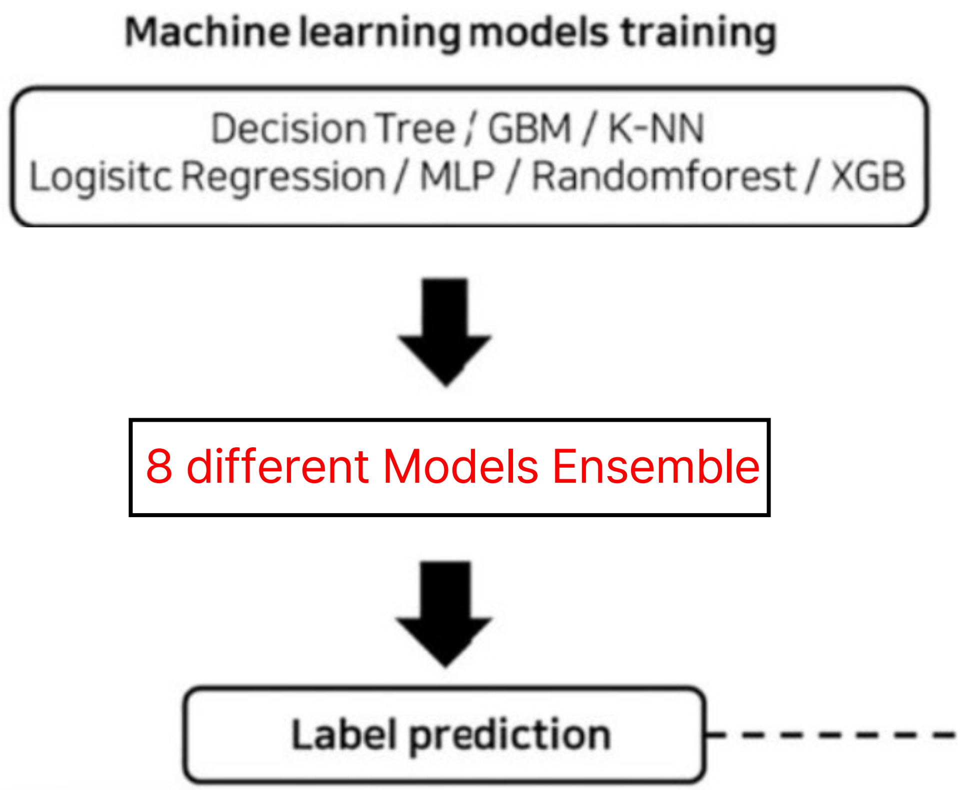
[Original Code]

```
for i in range(0,limit):
    for e in train_data[i].index:
        try:
            if train_data[i]['label'][e]==train_data[i]['label'][e+1]:
                train_data[i]['position'][e+1]='no action'
            elif train_data[i]['label'][e] > train_data[i]['label'][e+1]:
                train_data[i]['position'][e+1]='sell'
            else:
                train_data[i]['position'][e+1]='buy'
        except:
            pass
```

[Modified Code]

Contribution (2)

✓ MLP Model Ensemble before Prediction



✓ Model prediction ensemble example

Naive	D.T	GBM	K-NN	L.R	MLP	R.F	XGB
0	1	0	0	1	1	0	0
Final Pred : 0 (Buy)							

✓ Since the Sell can be enforced only when Buy is preceded, a code was added to group (Buy, Sell) into a pair.

✓ If there was only Buy signal, corresponding Sell signal was added, and if only Sell signal was present, it was removed. (Because Sell can only be done if Buy precedes it)

Code for Contribution (2)

```python
c = 0
for i in range(0,limit):
    for e in test_7219[i].index:
        pos_results[i][c].append(test_7219[i]['position'][e])
        c += 1
```

```python
vote_results = [[] for i in range(0,limit)]
for i in range(0,limit):
    for c in range(len(pos_results[i])):
        vote = [0, 0, 0]
        for k in range(0,8):
            if pos_results[i][c][k] == 'buy':
                vote[0] += 1
            elif pos_results[i][c][k] == 'sell':
                vote[1] += 1
            else: vote[2] += 1
        if vote[0] == max(vote):
            vote_results[i].append('buy')
        elif vote[1] == max(vote):
            vote_results[i].append('sell')
        else: vote_results[i].append('no action')
```

```python
for i in range(0,limit):
    trend = 0
    for c in range(len(vote_results[i])):
        if vote_results[i][c] == 'buy':
            trend += 1
        if vote_results[i][c] == 'sell':
            trend -= 1
        if trend > 1:
            vote_results[i][c - 1] = 'sell'
            trend -= 1
        if trend < 0:
            vote_results[i][c] = 'no action'
            trend += 1

for i in range(0,limit):
    c = 0
    for e in test_7219[i].index:
        test_7219[i]['position'][e] = vote_results[i][c]
        c += 1
```

[Newly Added Code]

Final Results

year	stock_nam	No.trades	Win%	Average g	Average lo	Payoff rati	Profit fact	Model
2019~202	005930_삼	40	0.6	3353.449	2594.405	1.29257	1.938855	pred_logistic
2019~202	005930_삼	62	0.645161	2735.934	2462.944	1.110839	2.019707	pred_decision
2019~202	005930_삼	21	0.619048	4763.01	2389.794	1.993063	3.238727	pred_naive
2019~202	005930_삼	47	0.531915	1747.27	1845.27	0.946891	1.076013	pred_randomforest
2019~202	005930_삼	43	0.44186	1137.451	1213.071	0.937663	0.742316	pred_knn
2019~202	005930_삼	37	0.783784	6199.032	4748.931	1.305353	4.731905	pred_neural
2019~202	005930_삼	54	0.574074	2629.491	1182.786	2.223134	2.996398	pred_voting
2019~202	005930_삼	88	0.318182	1813.46	1844.46	0.983193	0.458823	pred_gbm

[Midterm Results with Instance Selection]

year	stock_name	No.trades	Win%	Average gain($)	Average loss($)	Payoff ratio	Profit factor
2019~2020	005930_Samsung	16	0.625	2098.4775	1625.829167	1.2907122	2.151187

[Final Results with Improved Signal Mapping & Ensemble Algorithm]

- ✓ The ensemble method **lowers trading frequency,** enabling the agent to execute **only high-confidence trades, thereby improving overall decision stability.**
- ✓ Although the number of trades reduced, **the win ratio, profit factor** increased in average.
- ✓ The average of **[Average gain – Average loss] is much better** than the midterm results.

Code Instruction Manual

✓ Development Environment: Python 3.10.1, Torch 2.5.1 => MUST DOWNGRADE

✓ TA-Lib must to be downloaded at https://github.com/TA-Lib/ta-lib-python.git
=> Then, run !pip install ta-lib-0.6.4-src.tar.gz within VScode (Linux version)

Windows

For 64-bit Windows, the easiest way is to get the *executable installer*:

1. Download ta-lib-0.6.4-windows-x86_64.msi.
2. Run the Installer or run `msiexec` from the command-line.

Alternatively, if you prefer to get the libraries without installing, or would like to use the 32-bit version:

- Intel/AMD 64-bit ta-lib-0.6.4-windows-x86_64.zip
- Intel/AMD 32-bit ta-lib-0.6.4-windows-x86_32.zip

Linux

Download ta-lib-0.6.4-src.tar.gz and:

```
$ tar -xzf ta-lib-0.6.4-src.tar.gz
$ cd ta-lib-0.6.4/
$ ./configure --prefix=/usr
$ make
$ sudo make install
```

> If you build `TA-Lib` using `make -jX` it will fail but that's OK! Simply rerun `make -jX` followed by `[sudo] make install`.

✓ Other libraries(ex. ko_KR.UTF-8, mplfinance, tqdm) can be installed within VScode if needed (installation commands are inserted within submitted codes -> Google-colab basis)

<u>Brief Instructions</u> -> Detailed instructions are given

1. Run ‘main.py’
   1a. Relocate prediction results to appropriate folders
2. Run ‘main2.py’
3. Retrieve final results in .csv format

Code Instruction Manual

Run ‘main.py’ with the following commands sequentially

0. Modify path at line[7]

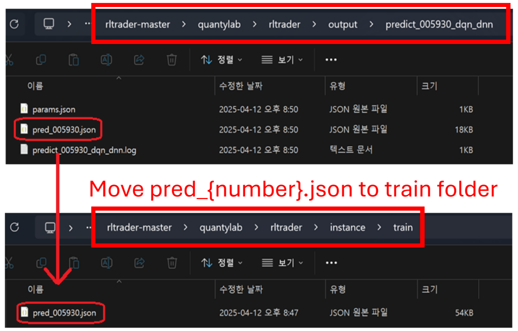
1. train data

```
python main.py --mode train --ver v5 --name 005930 --stock_code 005930 --rl_method dqn --net dnn --start_date 20130101 --end_date 20181231
```

2. predict train set

```
python main.py --mode predict --ver v5 --name 005930 --stock_code 005930 --rl_method dqn --net dnn --start_date 20130101 --end_date 20181231
```

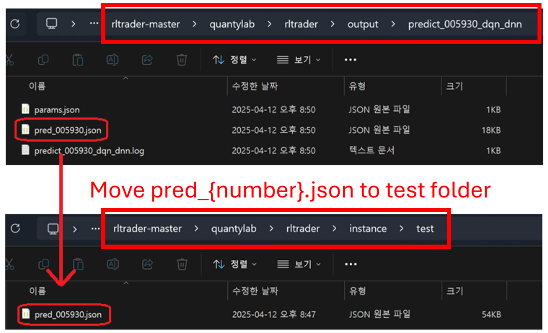
2a. Relocate train set prediction results



3. predict test set

```
python main.py --mode predict --ver v5 --name 005930 --stock_code 005930 --rl_method dqn --net dnn --start_date 20190101 --end_date 20201231
```

3a. Relocate test set prediction results



4. test data

```
python main.py --mode test --ver v5 --name 005930 --stock_code 005930 --rl_method dqn --net dnn --start_date 20190101 --end_date 20201231
```

Code Instruction Manual

Run ‘main2.py’ => Retrieve final results in .csv format

Example of final result:

year	stock_name	No.trades	Win%	Average gain($)	Average loss($)	Payoff ratio	Profit factor
2019~2020	005930_Samsung	16	0.625	2098.4775	1625.829167	1.2907122	2.151187

We have attached an inference code ‘inference.ipynb’ to help with code execution

Thank You

IIE4122 Final Project | 25.06.16 Mon

TEAM 13 | Kim Wonjun(2022147002), Kim Hyeonjin(2021125085), Jang Seohyun(2021190002)

[Reference] Park, M., Kim, J., & Enke, D. (2024). A novel trading system for the stock market using Deep Q-Network action and instance selection. Expert Systems with Applications, 257, 125043.