프로젝트 종합 보고서

컴퓨터공학과 20181463 김기호
컴퓨터공학과 20181479 현호성
컴퓨터공학과 20181467 정우창

1. 개요

1-1. 프로젝트 목표

- 강화학습을 통한 원자재 가격 예측 시스템을 개발하여 시장 예측, 시스템 자동화, AI 학습 기능을 제공해야 한다.
- 잘못된 투자로 인한 손실을 줄이기 위해 종목의 가격을 예측하여 리스크를 감소시켜야 한다.
- 시스템의 자동화로 고객의 요구에 대한 수용 능력을 향상 시켜 신뢰서을 증대 시켜야 한다.
- 인력의 학습보다 비용이 절감되고, 방대한 용량을 학습할 수 있어 효율이 증대해야 한다.

1-2. 주제 선정 이유

일반적으로는 기업과 연관된 주식을 매수 및 매도하여 수익을 보는 개인투자자들이 많이 존재하였는데, 전문적으로 투자를 공부하지 않거나 불확실한 정보를 가지고 주식을 투자하는 사람들의 비율이 꽤 많았으며 그 중 상당 수는 신규 투자자들이며 그들의 수익률은 평균적으로 마이너스를 내고 있었다. 엎친 데 덮친 격으로 2020년에는 코로나 팬데믹 상황으로 인하여 전 세계적으로 경제가 어려워지며 주가들도 폭락하여 신규 투자자들도 손해가 막심한 상황이였다. 그럴수록 사람들은 금과 같은 원자재 주식에 눈을 돌려 안전자산으로서 해당 주식들을 구매하여 경기가 다시 좋아지면 원자재 주식들의 가격도 안정적으로 상승할 것을 기대하며 많은 개인 투자자들이 원자재 주식의 구매에 눈독을 들였다. 그러한 상황에서 더욱 수익률을 낼 수 있는 방법을 쉽게 마련하고자 강화학습을 이용한 주식 예측 프로그램을 개발하고자 마음을 먹었다.

1-3. 하드웨어, 소프트웨어 개발 환경

하드웨어 개발환경	
품목	컴퓨터 / 3식
기능	• 웹 서버 가동 및 유지 • 데이터 처리 관련 임무 수행 • 인공지능 학습
성능	HDD: 50GB 이상
요건	중복된 시스템 운용

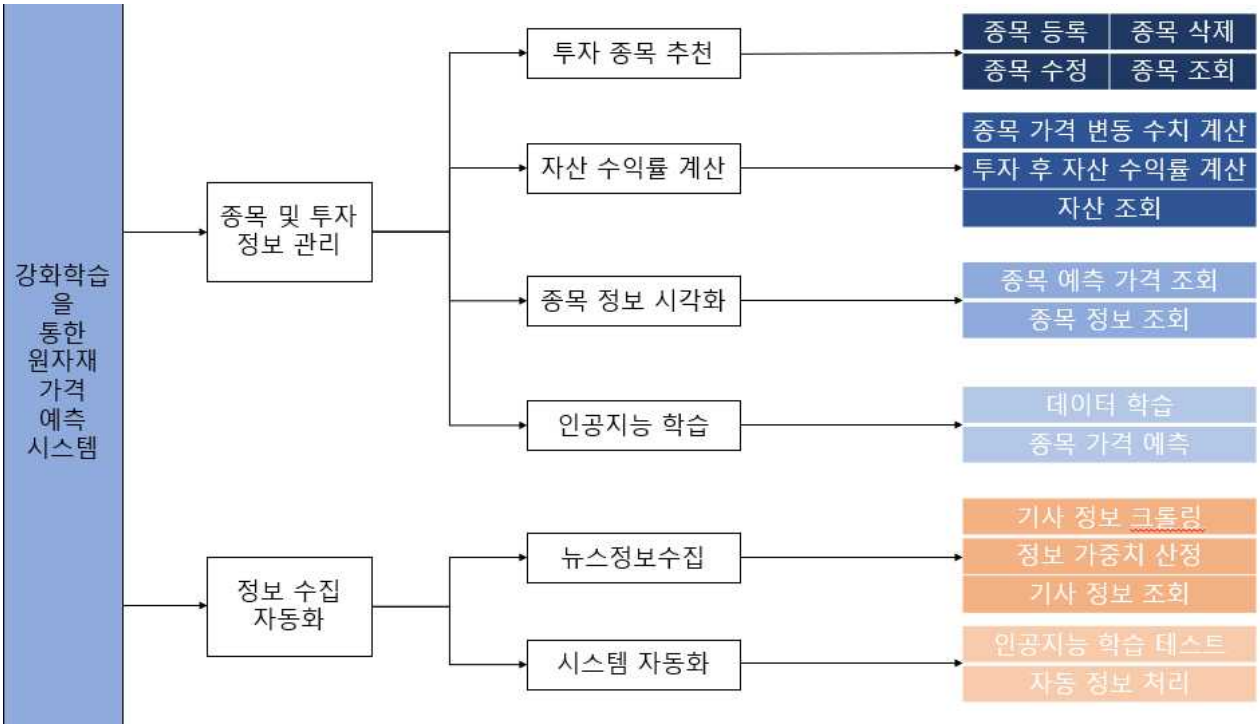
소프트웨어 개발환경	
품목	쥬피터 노트북 구글 코랩
기능	• 웹 서버 구축 • 데이터 처리 관련 임무 수행 및 뉴스 데이터 수집 • 인공지능 학습 및 백테스팅
요건	• 파이썬 3.0 버전 이상 • Window 10/11, Mac 등 다양한 OS 환경 지원 • 파이썬 라이브러리 설치 • 구글 계정 필요

1-4. 소프트웨어(서비스) 주요 기능

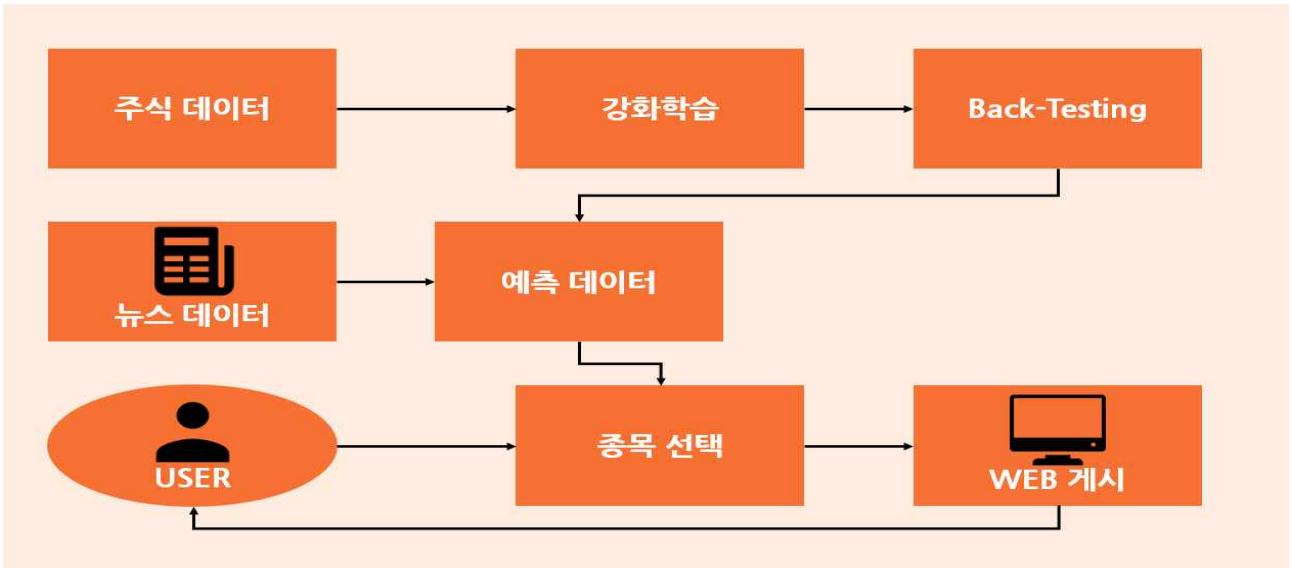
- 투자 정보 기능 : 투자 종목을 등록, 삭제, 수정, 조회한다.
- 자산 수익률 계산 기능 : 종목 가격 변동 수치 계산, 투자 후 자산 수익률 계산, 자산 조회한다.
- 종목 정보 시각화 기능 : 종목 예측 가격 조회, 종목 정보 조회한다.
- 인공지능 학습 기능 : 데이터 학습, 종목 가격 예측한다.
- 뉴스 정보 수집 기능 : 기사 정보 크롤링, 정보 가중치 산정, 기사 정보 조회한다.
- 시스템 자동화 기능 : 인공지능 학습 테스트, 정보 자동 처리한다.

2. 프로젝트 구조

2-1. 프로젝트 시스템 구조 개요



2-2. 프로젝트 구조도



3. 모듈 설계

모듈 이름	실행 모듈
모듈 형	사용자 정의 모듈
인터페이스	NA
오류 메시지	NA
사용하는 파일	main.py
호출하는 모듈	데이터 관리자, 학습기, 유틸리티
기능 설명	강화학습 실행 파일

모듈 이름	데이터 관리자 모듈
모듈 형	제3자 라이브러리 모듈
인터페이스	NA
오류 메시지	NA
사용하는 파일	data_manager.py
호출하는 모듈	세팅
기능 설명	데이터 로드, 전처리

모듈 이름	학습기 모듈
모듈 형	제삼자 라이브러릴 모듈
인터페이스	NA
오류 메시지	NA
사용하는 파일	learners.py
호출하는 모듈	유틸리티, 환경, 네트워크, 가시화, 에이전트
기능 설명	가치 네트워크, 정책 네트워크 설정, 배치 설정, 실행 설정, 강화학습 신경망 설정, 예측 설정

모듈 이름	유틸리티 모듈
모듈 형	내장 모듈
인터페이스	NA
오류 메시지	NA
사용하는 파일	utils.py
호출하는 모듈	NA
기능 설명	날짜 데이터 변환, 시그모이드 함수

모듈 이름	환경 모듈
모듈 형	사용자 정의 모듈
인터페이스	NA
오류 메시지	NA
사용하는 파일	environment.py
호출하는 모듈	NA
기능 설명	원자재 시계열 데이터 강화학습 환경 구축

모듈 이름	네트워크 모듈
모듈 형	제삼자 라이브러리 모듈
인터페이스	NA
오류 메시지	NA
사용하는 파일	networks.py
호출하는 모듈	NA
기능 설명	케라스, 파이토치를 활용한 DNN, LSTM, CNN 신경망 호출

모듈 이름	가시화 모듈
모듈 형	제삼자 라이브러리 모듈
인터페이스	NA
오류 메시지	NA
사용하는 파일	visualizer.py
호출하는 모듈	에이전트
기능 설명	Matplotlib 라이브러리를 활용한 시계열 데이터 그래프 작성

모듈 이름	에이전트 모듈
모듈 형	사용자 정의 모듈
인터페이스	NA
오류 메시지	NA
사용하는 파일	agent.py
호출하는 모듈	유틸리티
기능 설명	주식 보유 개수, 매매 수수료 및 세금, 행동(관망, 매수, 매도) 등 에이전트 구축

4. 기능 구현 현황 및 코드 설명

4-1. 주식 데이터

<table>
<tr><th>분류</th><th>설명</th></tr>
<tr><td>관련 파일</td><td>• data_manager</td></tr>
<tr><td>코드</td><td>

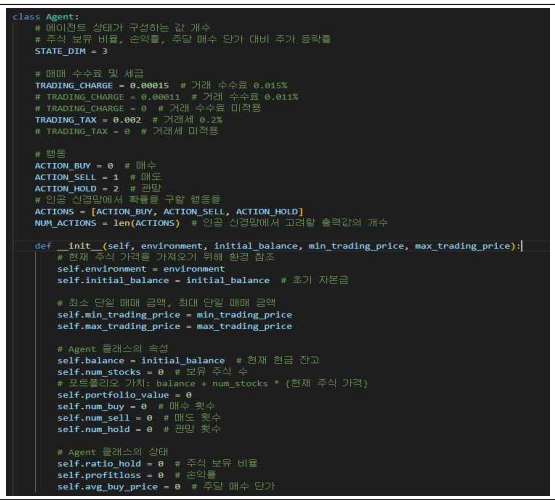
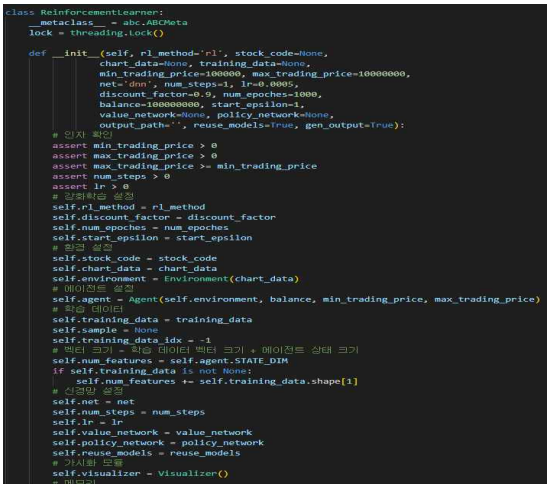
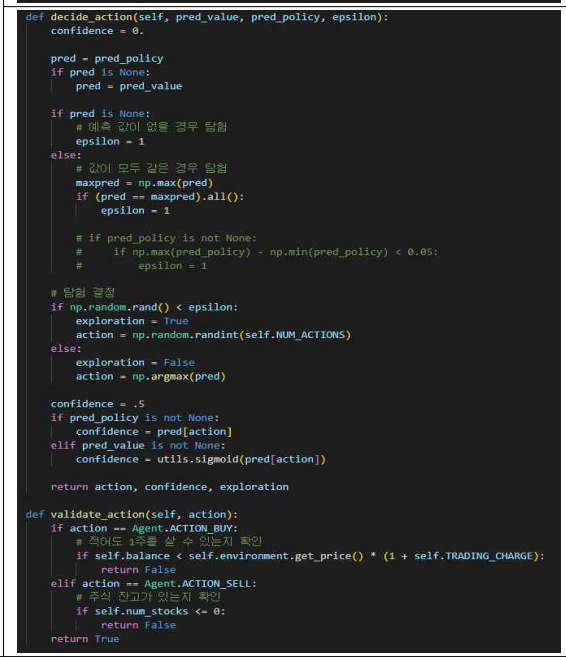
```python
def z_score_normalization(data):
    mean_value = np.mean(data)
    std_dev = np.std(data)
    normalized_data = (data - mean_value) / std_dev
    return normalized_data

def ydatareader(code, start_date, end_date):
    orig_df = yf.download(code, start_date, end_date)
    orig_df.drop(columns=['Adj Close'], inplace=True)
    orig_df.reset_index(drop=False, inplace=True)
    orig_df.columns = ['date', 'open', 'high', 'low', 'close', 'volume']

    prep_df = preprocess(orig_df)

    sp_df = ydatareader_v2('^GSPC', start_date, end_date)
    merge_df = pd.merge_asof(prep_df, sp_df, on='date', direction='nearest')
    dji_df = ydatareader_v2('^DJI', start_date, end_date)
    merge_df = pd.merge_asof(merge_df, dji_df, on='date', direction='nearest')
    ixic_df = ydatareader_v2('^IXIC', start_date, end_date)
    merge_df = pd.merge_asof(merge_df, ixic_df, on='date', direction='nearest')

    date = datetime.strptime(str(start_date), '%Y-%m-%d')
    for i in range(len(merge_df)):
        if merge_df['date'][i] >= datetime(date.year + 1, 1, 1):
            merge_df = merge_df[i:]
            break

    merge_df.reset_index(drop=True, inplace=True)

    return merge_df

def ydatareader_v2(code, start_date, end_date):
    orig_df = yf.download(code, start_date, end_date)
    orig_df.drop(columns=['Adj Close'], inplace=True)
    orig_df.reset_index(drop=False, inplace=True)
    orig_df.columns = ['date', 'open', 'high', 'low', 'close', 'volume']

    prep_df = preprocess_v2(code, orig_df)

    date = datetime.strptime(str(start_date), '%Y-%m-%d')
    for i in range(len(prep_df)):
        if prep_df['date'][i] >= datetime(date.year + 1, 1, 1):
            prep_df = prep_df[i:]
            break

    prep_df.reset_index(drop=True, inplace=True)

    return prep_df
```

```python
def preprocess(data):
    windows = [5, 10, 20, 60, 120]
    for window in windows:
        data[f'sma_close_ma{window}'] = ta.trend.sma_indicator(close=data['close'], window=window)
        data[f'sma_volume_ma{window}'] = ta.trend.sma_indicator(close=data['volume'], window=window)
        data[f'ema_close_ma{window}'] = ta.trend.ema_indicator(close=data['close'], window=window)
        data[f'ema_volume_ma{window}'] = ta.trend.ema_indicator(close=data['volume'], window=window)

        # z-점수-정규화
        data[f'sma_close_ma{window}_ratio'] = z_score_normalization(data[f'sma_close_ma{window}'])
        data[f'sma_volume_ma{window}_ratio'] = z_score_normalization(data[f'sma_volume_ma{window}'])
        data[f'ema_close_ma{window}_ratio'] = z_score_normalization(data[f'ema_close_ma{window}'])
        data[f'ema_volume_ma{window}_ratio'] = z_score_normalization(data[f'ema_volume_ma{window}'])

    data['bbh'] = ta.volatility.bollinger_hband(close=data['close'], fillna=True)
    data['bbl'] = ta.volatility.bollinger_lband(close=data['close'], fillna=True)
    data['rsi'] = ta.momentum.rsi(close=data['close'], fillna=True)
    data['sr'] = ta.momentum.stoch(close=data['close'], high=data['high'], low=data['low'], fillna=True)
    data['obv'] = ta.volume.on_balance_volume(close=data['close'], volume=data['volume'], fillna=True)
    data['adi'] = ta.volume.acc_dist_index(high=data['high'], low=data['low'], close=data['close'], volume=data['volume'], fillna=True)
    data['macd'] = ta.trend.macd(close=data['close'], fillna=True)

    data['diffratio'] = np.zeros(len(data))
    data.loc[1:, 'diffratio'] = (data['close'][1:].values - data['close'][:-1].values) / data['close'][:-1].values) * 100
    data['open_lastclose_ratio'] = np.zeros(len(data))
    data.loc[1:, 'open_lastclose_ratio'] = (data['open'][1:].values - data['close'][:-1].values) / data['close'][:-1].values * 100
    data['high_close_ratio'] = (data['high'].values - data['close'].values) / data['close'].values * 100
    data['low_close_ratio'] = (data['low'].values - data['close'].values) / data['close'].values * 100
    data['close_lastclose_ratio'] = np.zeros(len(data))
    data.loc[1:, 'close_lastclose_ratio'] = (data['close'][1:].values - data['close'][:-1].values) / data['close'][:-1].values * 100
    data['volume_lastvolume_ratio'] = np.zeros(len(data))
    data.loc[1:, 'volume_lastvolume_ratio'] = (data['volume'][1:].values - data['volume'][:-1].values) / data['volume'][:-1].replace(to
    data['buy_strength'] = np.zeros(len(data))
    data['buy_strength'] = np.log(1 + np.maximum((data['high'] - data['open']) + (data['close'] - data['low']), 0 ) * data['volume']).re
    data['sell_strength'] = np.zeros(len(data))
    data['sell_strength'] = np.log(1 + np.maximum((data['high'] - data['close']) + (data['open'] - data['low']), 0) * data['volume']).re

    # z-점수-정규화
    data['bbh_ratio'] = z_score_normalization(data['bbh'])
    data['bbl_ratio'] = z_score_normalization(data['bbl'])
    data['rsi_ratio'] = z_score_normalization(data['rsi'])
    data['sr_ratio'] = z_score_normalization(data['sr'])
    data['macd_ratio'] = z_score_normalization(data['macd'])
    data['obv_ratio'] = z_score_normalization(data['obv'])
    data['adi_ratio'] = z_score_normalization(data['adi'])
    data['buy_strength_ratio'] = z_score_normalization(data['buy_strength'])
    data['sell_strength_ratio'] = z_score_normalization(data['sell_strength'])
```

```python
def preprocess_v2(code, data):
    if code == '^GSPC':
        name = 'sp'
    elif code == '^DJI':
        name = 'dji'
    elif code == '^IXIC':
        name = 'ixic'

    windows = [5, 10, 20, 60, 120]
    for window in windows:
        data[f'sma_{name}_close_ma{window}'] = ta.trend.sma_indicator(close=data['close'], window=window)
        data[f'ema_{name}_close_ma{window}'] = ta.trend.ema_indicator(close=data['close'], window=window)
        data[f'sma_{name}_volume_ma{window}'] = ta.trend.sma_indicator(close=data['volume'], window=window)
        data[f'ema_{name}_volume_ma{window}'] = ta.trend.ema_indicator(close=data['volume'], window=window)
        # z_점수_정규화
        data[f'sma_{name}_close_ma{window}_ratio'] = z_score_normalization(data[f'sma_{name}_close_ma{window}'])
        data[f'ema_{name}_close_ma{window}_ratio'] = z_score_normalization(data[f'ema_{name}_close_ma{window}'])
        data[f'sma_{name}_volume_ma{window}_ratio'] = z_score_normalization(data[f'sma_{name}_volume_ma{window}'])
        data[f'ema_{name}_volume_ma{window}_ratio'] = z_score_normalization(data[f'ema_{name}_volume_ma{window}'])

    data[f'{name}_close_diff'] = np.zeros(len(data))
    data.loc[1:, f'{name}_close_diff'] = (data['close'][1:].values - data['close'][:-1].values) / data['close'][:-1
    data[f'{name}_volume_diff'] = np.zeros(len(data))
    data.loc[1:, f'{name}_volume_diff'] = (data['volume'][1:].values - data['volume'][:-1].values) / data['volume'

    data = data[['date', f'sma_{name}_close_ma5_ratio', f'sma_{name}_close_ma10_ratio', f'sma_{name}_close_ma20_rat
                 f'ema_{name}_close_ma5_ratio', f'ema_{name}_close_ma10_ratio', f'ema_{name}_close_ma20_ratio', f'em
                 f'sma_{name}_volume_ma5_ratio', f'sma_{name}_volume_ma10_ratio', f'sma_{name}_volume_ma20_ratio', f
                 f'ema_{name}_volume_ma5_ratio', f'ema_{name}_volume_ma10_ratio', f'ema_{name}_volume_ma20_ratio', f
                 f'{name}_close_diff', f'{name}_volume_diff']]

    return data

def load_data(code, start_date, end_date):
    # 파이낸스데이터리더 API 데이터 2변치 및 데이터 전처리
    df = ydatareader(code, start_date, end_date)
    # 기간 변환
    df['date'] = df['date'].astype(str)
    df['date'] = df['date'].str.replace('-', '')
    # 차트 데이터 분리
    chart_data = df[COLUMNS_CHART_DATA]
    # 학습 데이터 분리
    training_data = df[COLUMNS_TRAINING_DATA].values

    return chart_data, training_data
```

</td></tr>
<tr><td>구현 설명</td><td>• 원자재 종목 번호와 데이터 수집 기간을 입력받으면 yfinance 라이브러리를 통하여 일별 종목의 시작가, 종가, 거래량 등의 정보를 가져온다.
• 가져온 정보를 빠른 학습을 위하여 0과 1 사이의 값으로 정규화하고 이상치를 대치한다.</td></tr>
</table>

4-2. 강화학습

<table>
<tr><th>분류</th><th>설명</th></tr>
<tr><td>관련 파일</td><td>• agent
• learners</td></tr>
<tr><td>코드</td><td>




</td></tr>
<tr><td>구현 설명</td><td>• agent는 학습 전에 주식 보유 비율이나 주가 등락률, 매매 수수료 등의 주식의 환경에 관한 정보를 미리 설정하여 AI가 학습할 때에 환경을 설정해준다.
• learners는 개발자가 지정한 신경망 및 선택 알고리즘을 통해 가치 신경망과 정책 신경망을 생성 후 에포크의 크기만큼 임의의 값을 대입하여 학습을 수행 후 보상을 추가하는 식으로 신경망의 깊이 및 경로를 조정하여 최선의 값이 나오도록 반복 학습을 수행한다.</td></tr>
</table>

4-3. 뉴스 데이터 및 예측 데이터

<table>
<tr><th>분류</th><th colspan="2">설명</th></tr>
<tr><td>관련 파일</td><td colspan="2">• crawling
• learners</td></tr>
<tr><td rowspan="2">코드</td><td>

```
import pandas as pd
from bs4 import BeautifulSoup as bs
from selenium import webdriver
from selenium.webdriver.common.by import By

titles, news_titles = [], []

driver = webdriver.Chrome()

for i in range(0, 300, 10):
    url = 'https://www.google.com/search?q=us+gold+news&sca_esv=568002961&tbm=nws&sxsrf=AM9HkKnvrWYBzpAuj8OVbeK9-r2o53KrQQ:1695573480308&ei=o4AQZf5hIoeH2...'

    driver.get(url)

    driver.implicitly_wait(10) # 페이지가 모두 업로드될 때까지 기다림

    html_txt = driver.page_source # url에 접속해서 html 가져오기

    soup = bs(html_txt, 'html.parser') # html 파싱(문장 해석)

    for j in range(1, 11, 1):
        element = soup.select_one('#rso > div > div > div:nth-child({}) > div > div > a > div > div.iRPxbe > div.mBjPhd.ynAwRc.MBeuO.nDgy9d'.format(j))
        if element:
            titles.append(element.get_text())

    news_titles.append(titles)

    titles = []

driver.quit()

flat_titles = [title for sublist in news_titles for title in sublist]

df = pd.DataFrame({'뉴스 제목': flat_titles})
csv_filename = 'US_Gold_News.csv'
df.to_csv(csv_filename, index=False, encoding='utf-8-sig')
```

</td><td>

```
def predict(self):
    # 에이전트 초기화
    self.agent.reset()

    # step 샘플을 만들기 위한 큐
    q_sample = collections.deque(maxlen=self.num_steps)

    result = []
    while True:
        # 샘플 생성
        next_sample = self.build_sample()
        if next_sample is None:
            break

        # num_steps만큼 샘플 저장
        q_sample.append(next_sample)
        if len(q_sample) < self.num_steps:
            continue

        # 가치, 정책 신경망 예측
        pred_value = None
        pred_policy = None
        if self.value_network is not None:
            pred_value = self.value_network.predict(list(q_sample)).tolist()
        if self.policy_network is not None:
            pred_policy = self.policy_network.predict(list(q_sample)).tolist()

        # 신경망에 의한 행동 결정
        result.append((self.environment.observation[0], pred_value, pred_policy))

    if self.gen_output:
        with open(os.path.join(self.output_path, f'pred_{self.stock_code}.json'), 'w') as f:
            print(json.dumps(result), file=f)

    return result
```

</td></tr>
<tr><td>

```
    nltk.download('vader_lexicon')
    sia = SentimentIntensityAnalyzer()
```

```
[nltk_data] Downloading package vader_lexicon to
[nltk_data]     C:\Users\ghfod\AppData\Roaming\nltk_data...
[nltk_data]   Package vader_lexicon is already up-to-date!
```

```
    text = 'fuck you'
    sia.polarity_scores(text)
```

```
{'neg': 0.778, 'neu': 0.222, 'pos': 0.0, 'compound': -0.5423}
```

```
    score = []
    vader = SentimentIntensityAnalyzer()

    for index, row in df.iterrows():
        text = row['뉴스 제목']
        score.append(vader.polarity_scores(text))
```

```
    score
```

```
Output exceeds the size limit. Open the full output data in a text editor
[{'neg': 0.091, 'neu': 0.909, 'pos': 0.0, 'compound': -0.1531},
 {'neg': 0.0, 'neu': 1.0, 'pos': 0.0, 'compound': 0.0},
 {'neg': 0.0, 'neu': 1.0, 'pos': 0.0, 'compound': 0.0},
 {'neg': 0.138, 'neu': 0.526, 'pos': 0.336, 'compound': 0.4588},
 {'neg': 0.0, 'neu': 1.0, 'pos': 0.0, 'compound': 0.0},
 {'neg': 0.0, 'neu': 1.0, 'pos': 0.0, 'compound': 0.0},
 {'neg': 0.091, 'neu': 0.743, 'pos': 0.166, 'compound': 0.0772},
 {'neg': 0.203, 'neu': 0.797, 'pos': 0.0, 'compound': -0.4215},
 {'neg': 0.246, 'neu': 0.593, 'pos': 0.161, 'compound': -0.25},
 {'neg': 0.0, 'neu': 0.84, 'pos': 0.16, 'compound': 0.2732},
 {'neg': 0.18, 'neu': 0.82, 'pos': 0.0, 'compound': -0.296},
 {'neg': 0.0, 'neu': 1.0, 'pos': 0.0, 'compound': 0.0},
 {'neg': 0.145, 'neu': 0.855, 'pos': 0.0, 'compound': -0.296},
```

</td><td>

```
class A2CLearner(ActorCriticLearner):
    def __init__(self, *args, **kwargs):
        super().__init__(*args, **kwargs)

    def get_batch(self):
        memory = zip(
            reversed(self.memory_sample),
            reversed(self.memory_action),
            reversed(self.memory_value),
            reversed(self.memory_policy),
            reversed(self.memory_reward),
        )
        x = np.zeros((len(self.memory_sample), self.num_steps, self.num_features))
        y_value = np.zeros((len(self.memory_sample), self.agent.NUM_ACTIONS))
        y_policy = np.zeros((len(self.memory_sample), self.agent.NUM_ACTIONS))
        value_max_next = 0
        reward_next = self.memory_reward[-1]
        for i, (sample, action, value, policy, reward) in enumerate(memory):
            x[i] = sample
            r = reward_next + self.memory_reward[-1] - reward * 2
            reward_next = reward
            y_value[i, :] = value
            y_value[i, action] = np.tanh(r + self.discount_factor * value_max_next)
            advantage = y_value[i, action] - y_value[i].mean()
            y_policy[i, :] = policy
            y_policy[i, action] = utils.sigmoid(advantage)
            value_max_next = value.max()
        return x, y_value, y_policy
```

</td></tr>
<tr><td>구현 설명</td><td colspan="2">• selenium 라이브러리를 통하여 지정한 URL의 정보를 크롤링 후 정보를 텍스트로 저장한다.
• 저장된 텍스트 정보를 문장마다 vader 라이브러리를 통해 감성분석을 하여 긍정 및 부정을 수치화하여 저장한다.
• 수치화된 정보들을 예측 데이터에 넣어 분산 투자의 배분율에 적용한다.</td></tr>
</table>

4-4. 종목 선택

<table>
<tr><th>분류</th><th colspan="2">설명</th></tr>
<tr><td>관련 파일</td><td colspan="2">• main</td></tr>
<tr><td>코드</td><td>

```
import os
import sys
import logging
import argparse
import json

from src.quantylab.rltrader import settings
from src.quantylab.rltrader import utils
from src.quantylab.rltrader import data_manager


if __name__ == '__main__':
    parser = argparse.ArgumentParser()
    parser.add_argument('--mode', choices=['train', 'test', 'update', 'predict'], default='train')
    parser.add_argument('--ver', choices=['v1', 'v2', 'v3', 'v4', 'v4.1', 'v4.2'], default='v4.1')
    parser.add_argument('--name', default=utils.get_time_str())
    parser.add_argument('--stock_code', nargs='+')
    parser.add_argument('--rl_method', choices=['dqn', 'pg', 'ac', 'a2c', 'a3c', 'monkey'], default='a2c')
    parser.add_argument('--net', choices=['dnn', 'lstm', 'cnn', 'monkey'], default='dnn')
    parser.add_argument('--backend', choices=['pytorch', 'tensorflow', 'plaidml'], default='pytorch')
    parser.add_argument('--start_date', default='20200101')
    parser.add_argument('--end_date', default='20201231')
    parser.add_argument('--lr', type=float, default=0.001)
    parser.add_argument('--discount_factor', type=float, default=0.7)
    parser.add_argument('--balance', type=int, default=100000000)
    args = parser.parse_args()

    # 학습기 파라미터 설정
    output_name = f'{args.mode}_{args.name}_{args.rl_method}_{args.net}'
    learning = args.mode in ['train', 'update']
    reuse_models = args.mode in ['test', 'update', 'predict']
    value_network_name = f'{args.name}_{args.rl_method}_{args.net}_value.mdl'
    policy_network_name = f'{args.name}_{args.rl_method}_{args.net}_policy.mdl'
    start_epsilon = 1 if args.mode in ['train', 'update'] else 0
    num_epoches = 100 if args.mode in ['train', 'update'] else 1
    num_steps = 5 if args.net in ['lstm', 'cnn'] else 1

    # Backend 설정
    os.environ['RLTRADER_BACKEND'] = args.backend
    if args.backend == 'tensorflow':
        os.environ['KERAS_BACKEND'] = 'tensorflow'
    elif args.backend == 'plaidml':
        os.environ['KERAS_BACKEND'] = 'plaidml.keras.backend'

    # 출력 경로 생성
    output_path = os.path.join(settings.BASE_DIR, 'output', output_name)
    if not os.path.isdir(output_path):
        os.makedirs(output_path)
```

</td><td>

```
# Backend 설정
os.environ['RLTRADER_BACKEND'] = args.backend
if args.backend == 'tensorflow':
    os.environ['KERAS_BACKEND'] = 'tensorflow'
elif args.backend == 'plaidml':
    os.environ['KERAS_BACKEND'] = 'plaidml.keras.backend'

# 출력 경로 생성
output_path = os.path.join(settings.BASE_DIR, 'output', output_name)
if not os.path.isdir(output_path):
    os.makedirs(output_path)

# 파라미터 기록
params = json.dumps(vars(args))
with open(os.path.join(output_path, 'params.json'), 'w') as f:
    f.write(params)

# 모델 경로 준비
# 모델 포맷은 TensorFlow는 h5, PyTorch는 pickle
value_network_path = os.path.join(settings.BASE_DIR, 'models', value_network_name)
policy_network_path = os.path.join(settings.BASE_DIR, 'models', policy_network_name)

# 로그 기록 설정
log_path = os.path.join(output_path, f'{output_name}.log')
if os.path.exists(log_path):
    os.remove(log_path)
logging.basicConfig(format='%(message)s')
logger = logging.getLogger(settings.LOGGER_NAME)
logger.setLevel(logging.DEBUG)
logger.propagate = False
stream_handler = logging.StreamHandler(sys.stdout)
stream_handler.setLevel(logging.INFO)
file_handler = logging.FileHandler(filename=log_path, encoding='utf-8')
file_handler.setLevel(logging.DEBUG)
logger.addHandler(stream_handler)
logger.addHandler(file_handler)
logger.info(params)

# Backend 설정, 로그 설정을 먼저하고 RLTrader 모듈들을 이후에 임포트해야 함
from rltrader.learners import ReinforcementLearner, DQNLearner, \
    PolicyGradientLearner, ActorCriticLearner, A2CLearner, A3CLearner

common_params = {}
list_stock_code = []
list_chart_data = []
list_training_data = []
```

</td></tr>
<tr><td>구현 설명</td><td colspan="2">• 메인 함수를 호출하면서 매개변수로 종목 수, 자산액 및 투자특성을 입력하면 함수가 실행되면서 학습된 데이터 중 적합한 정보들을 호출하여 보여주고 이를 파일형식으로 저장한다.</td></tr>
</table>

5. 제약 사항 및 참고 사항

5-1. 제약 사항

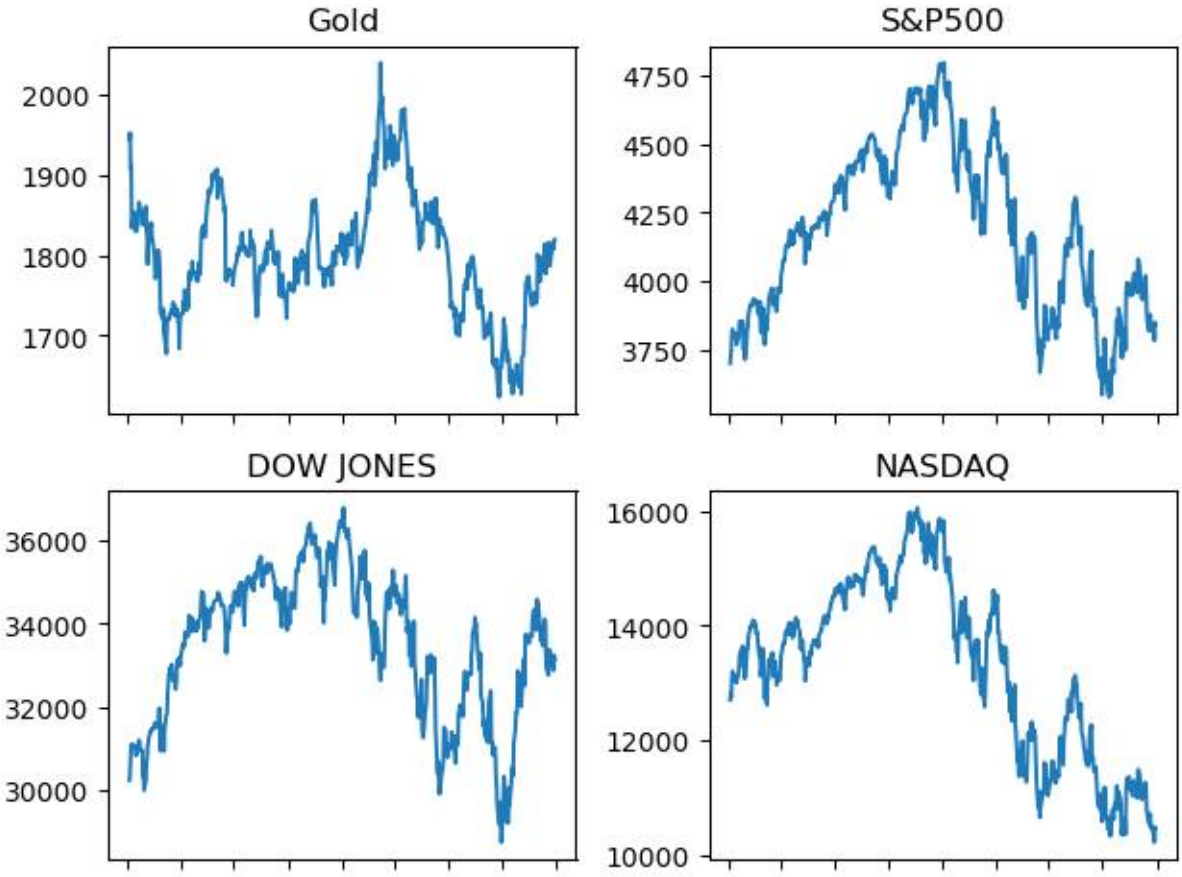
- 개발 시 다른 기업이 배포한 시스템을 이용하기 때문에 해당 시스템의 라이선스 기준 및 규격 범위 한에서 개발을 진행하여야 한다.
- 지원된 자금이 없기 때문에 개발 과정에서 발생한 모든 비품과 소모품 구매에 관한 비용은 개인 부담이므로 높은 금액이 발생하지 않는 한에서 개발을 진행하여야 한다.

5-2. 참고 사항

- 저장되는 데이터가 없기 때문에 별도의 데이터베이스 운영은 하지 않는다.
- 사용자의 정보는 일회성으로 사용되기 때문에 시스템을 이용할 때마다 정보를 기입하여야 한다.
- 개발 과정에서 앞에서 설명한 설계 항목 중 일부분은 변경될 수 있다.
- 같은 시간대라고 하더라도 학습된 데이터의 양에 따라 결과는 다르게 나타날 수 있다.

6. 결과 화면 및 해석

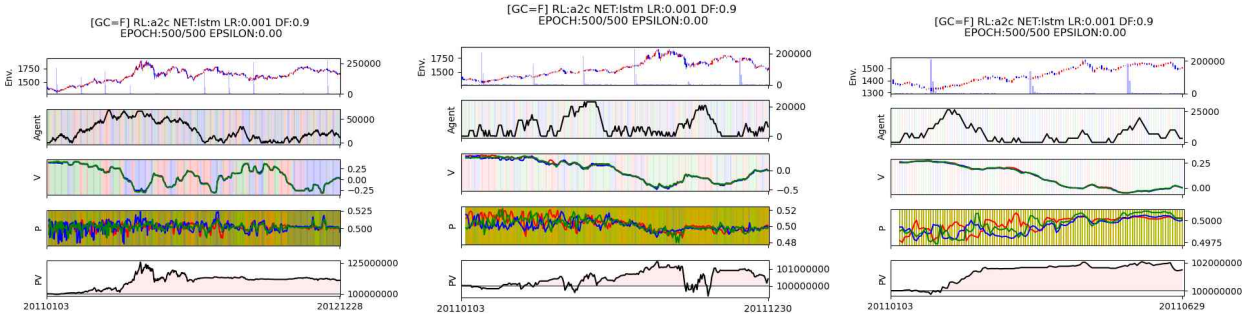
6-1. 종목 정보 수집 화면



- 화면 구성을 2*2 매트릭스로 구성하였으며 왼쪽 위부터 각각 원자재 종목, S&P500 지수, 다우존스 지수, 나스닥 지수이다.

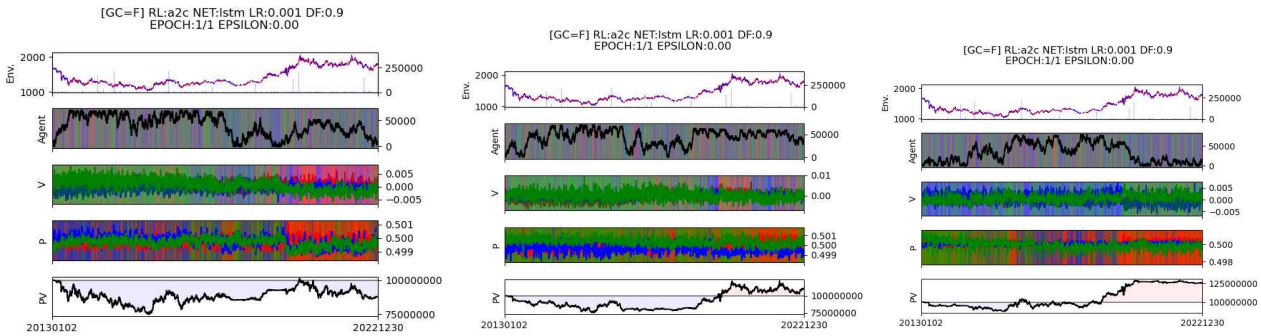
- 원자재 종목을 제외한 나머지 3개의 그래프는 경제 지표로 사용되며 학습에 사용되기도 하지만 이 그래프들을 사용자에게 보여주어, 단순한 프로그램의 판단에 맡기기보다는 사용자도 또한 학습 기간의 종목 대비 경제지표의 그래프 변화를 보면서 당시 상황을 판단할 수 있게 시각화하였다.

6-2. 데이터 학습 결과



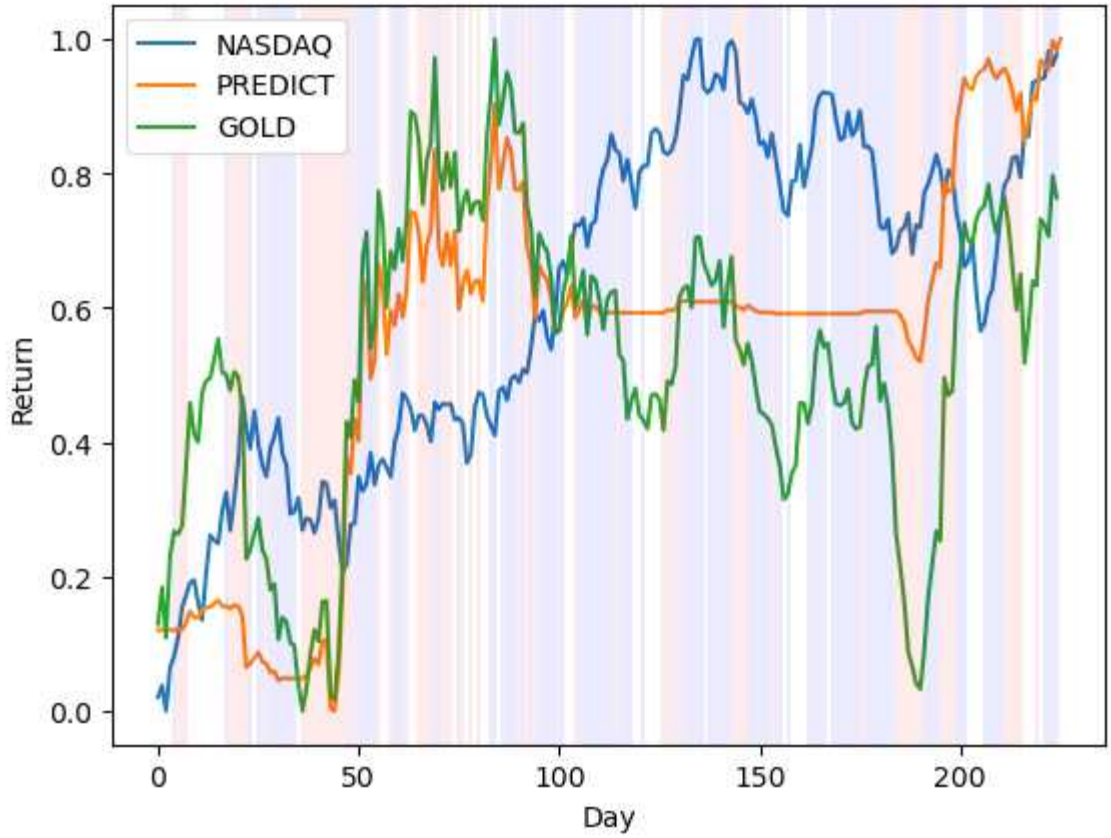
- 위의 이미지는 기간별로 종목에 대한 강화학습의 결과를 그래프로 나타낸 것이며, 같은 종목에 대해 학습 기간을 다르게 설정하여 학습을 수행한 것이다.
- 왼쪽부터 6개월, 1년, 2년으로 학습을 수행한 결과이고 결과적으로 보면 2년의 학습기간이 수익률이 가장 높게 나온 것을 볼 수 있다.
- 학습에 대한 결과 이미지는 5개의 그래프를 보여주는데, 위부터 종목의 거래가, 보유 주식 수, 가치 신경망 판단결과, 정책 신경망의 판단 결과, 자본금 순이다.
- 종목의 거래가에서는 일반적인 주식 프로그램처럼 양봉과 음봉, 거래량을 볼 수 있다.
- 보유 주식 수에서는 검은색 실선은 주식 수를 나타내며, 바탕의 파란색, 빨간색, 초록색은 각각 매도, 매수, 관망을 뜻하는 AI모델의 행동 의견을 보여준다. (여기서 관망은 매수와 매도를 하지 않고 그대로 유지한다는 뜻을 의미한다.)
- 가치 신경망과 정책 신경망은 각각 AI 모델의 행동에 따른 가치 판단과 매수 및 매도의 정책 판단을 나타낸다.
- 자본금은 AI모델이 판단한 결과를 통해 자본금을 토대로 매수 및 매도를 시행했을 때 자본금의 변화를 나타낸 것이다.
- 해당 학습을 시행할 때는 초기 자본금을 1억으로 설정하였으며 수수료 및 세금, 공매도는 없다고 가정하고 AI모델의 각각의 행동은 일별로 판단하도록 설정하였다.

6-3. 데이터 백테스팅 결과



- 위 결과는 앞서 학습한 AI모델을 가지고 학습 기간 이후의 날짜를 대상으로 거래 가격을 예측하고 이를 실제 데이터와 비교하여 유사도를 판단한 것이다.

6-4. 시장 지수와 트레이딩 시뮬레이션 결과



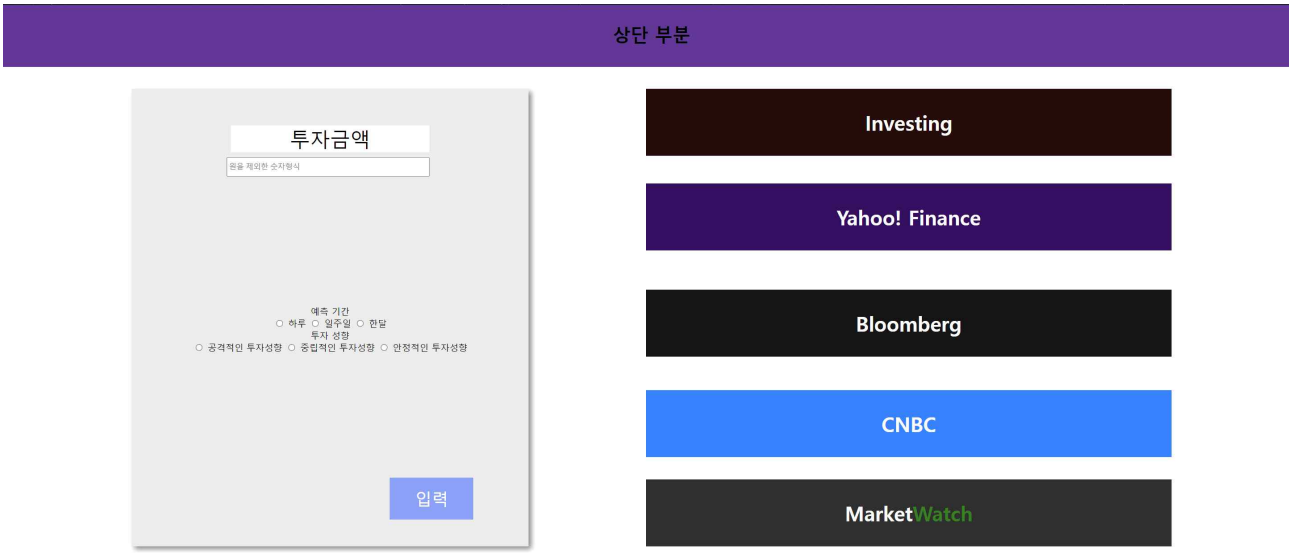
- 결과적으로 총 9.74%의 수익률을 기록하였다.
- 시장 지수가 급격히 상승하는 3월 달에는 투자 모델도 매수를 진행하면서 수익률을 5.56% 달성하였으며 시장 지수가 3.88% 하락하는 5~6월 달은 투자 모델이 관망하는 태도를 유지하였다.
- 시장 지수가 8.46% 상승하는 9월 달에는 투자 모델이 매수를 진행하며 수익률을 9.23% 달성하였으며 11월 20일을 기준으로 수익률이 소폭 상승하면서 테스트 최종 수익률은 9.74%를 달성하였다.
- 전체적으로 볼 때 시장 지수가 급격히 상승 및 하락할 때 강화학습을 통해 학습한 AI 모델은 빠르게 대응하는 판단을 보여주며 안정적인 등락률을 보여주었다.

7. 개선할 점

- 먼저 학습 데이터를 19~20년도 데이터를 사용했기에 21~23년도 데이터가 학습데이터의 min-max 값 범위 안에 들어가기에 당분간은 문제가 없지만 언젠가 문제가 생길 것이기에 향후

전처리 작업을 개선해야 한다고 생각하였다.

- 뉴스 기사 크롤링 및 분석 기능은 구현하였지만 AI학습에 필요한 20년치 뉴스 데이터를 수집하기에 시간이 부족하다고 느껴 수집량이 적어 AI 학습에 추가하지 못하여 향후 더 조사 후 기능을 구현해야한다고 생각하였다.



- 위 이미지와 같이 웹 사이트의 프레임워크 작성 및 기능은 일부 구현하였으나 완성도가 부족하고 AI 학습 코드와 연계하지 않아 초기에 작성하였던 기능 중 하나인 웹으로 게시하는 기능을 아직 구현하지 못해 추후에 개선하여 결과 화면을 웹으로 게시할 것이다.
- AI학습에 필요한 데이터 양이 현저히 부족하기도 하며 직접 데이터를 만들 수도 없기에 데이터를 확보할 수 있는 곳을 더욱 찾아보아야 한다고 생각하였다.

8. 역할 구성

20181463 김기호 (팀장)

- 데이터 분석 및 학습
- AI 모델 개발
- 알고리즘 이해 및 구현

20181479 현호성

- 데이터 분석 및 학습
- 웹 사이트 프레임워크 작성 및 기능 구현
- 차트 분석 및 자료 조사

20181467 정우창

- 발표자료 작성 및 자료 조사
- 데이터 학습