- 개선 방안 시도 및 결과 정리
  - 앙상블의 앙상블: 알고리즘트레이딩 new 에서 evaluation 에서 best performance 12 개 앙상블 조합(reinfo 0.5, pred_term 5 로 3 개의 모델에 대한 앙상블 조합 중 best 5 를 선택, 이들의 reinfo, pred_term 을 변화시켜 나온 결과 중에서)들의 앙상블을 만든다. 즉 모델들의 매도, 매수 결과 시그널 결과들을 majority voting 한 3 개의 앙상블의 결과들을 다시 majority voting 한다는 것이다.

    - 알고리즘트레이딩 new 에서는 향상된 결과를 보였지만 (profit_product 최고 6.72 → 19.36 로 향상, 평균 4.15 → 7.89 로 향상)

    - 알고리즘트레이딩 4 에서는 뚜렷한 개선효과 안 보임:
      이유: 알고리즘트레이딩 4 는 알고리즘트레이딩 new 와는 다르게 모델들의 타입이 보다 많다.
      알고리즘트레이딩 new 는 모델 타입이 10 개인 반면 알고리즘트레이딩 4 는 21 개 이다. 따라서 선별된 앙상블 조합들이 폭 넓은 모델들을 포함하고 있을 확률이 알고리즘트레이딩 new 보다 훨씬 낮다.
      알고리즘트레이딩 new 는 폭 넓은 모델들의 앙상블로부터 앙상블 조합을 만들기 때문에 안정적이고 높은 performance 를 보이는 것이 아닌가 싶다.

      알고리즘트레이딩 4 의 이러한 문제를 개선하기 위하여 앙상블 조합의 소스가 되는 앙상블들을 best 5 로 선별하여 확장된 list 가 아닌 전체 앙상블 list 에서 profit_product 가 높은 순으로 50 개의 앙상블로부터 앙상블 조합을 랜덤

추출하는 방식으로 바꾸어 시도한다.(10000 개의 앙상블 조합 추출)

문제는 50 개에서 3 개를 뽑는 앙상블의 앙상블 조합은 경우의 수가 19600 개가 된다는 것이다. 각각의 1 년치 performance 를 15 일 마다 학습한 모델들의 앙상블들의 앙상블 조합들이 19600 개이므로 실제 계산량은 24(1 년간 구간 수)*3(하나의 앙상블의 모델 수)*1330(21 개 모델로부터의 랜덤 앙상블 수)*19600 이 될 것이다. 이는 일반 PC 로 계산하려면 10 일 이상이 소요될 것이라 추측된다.

- 알고리즘트레이딩 new, 4 모두 performance 의 일관성이 결여되었다. 최근 년도에 좋았던 앙상블 조합이 과거 전체 기간에도 비슷하게 좋을 것인지 test 해 봤으나 결과는 실망적이었다. 예를 들어
  알고리즘트레이딩 new 의 최근년도에서 최고 performance 를 보였던 앙상블 조합 [[1, 6, 7, 05, 5], [1, 6, 7, 05, 4], [6, 8, 10, 05, 5]]
  *최근 년도(2022-01-01~2023-03-15)에서 4.5 배의 수익률을 기록했으나 2019-01-01~2023-03-15 에서의 수익률은 5.13 배에 불과했다. 알고리즘트레이딩 4 도 비슷한 결과를 보였다. 프로그램상에 에러가 있는지 살펴보아야 하겠지만 일단은 앙상블의 앙상블은 일관성이 없는 것 같다.
- 앙상블의 앙상블 관련 프로그램
  ensemble_ensembl_test.py: 앙상블의 결과 파일

(ensemble_results_..., eval_reflection_....)로부터 주어진 index 에 해당되는 앙상블들의 결과를 다시 앙상블하여 주어진 구간에서 수익률을 계산하여 보여준다.

create_all_term_ensemble_ensemble.py:
performance 순으로 정렬된 앙상블의 결과 파일로부터 정해진 개수 안에서 랜덤으로 3 개의 앙상블을 선택하여 주어진 구간에서 ensemble_ensemble_test.py 를 이용하여 보름 단위로 수익률을 차례로 계산한다. 즉 앙상블의 개수가 10 개이면 랜덤으로 선택되는 앙상블의 앙상블 조합은 120 개가 된다.

all_term_ensemble_ensemble.py: 주어진 앙상블의 앙상블 조합의 주어진 구간에서의 수익률을 계산한다.

- 알고리즘트레이딩 2 에서 target 수를 3(0:중립 1:매도 2:매수)에서 2(0:매도 1:매수)변경: 2022-01-01~2023-03-15 까지의 performance 를 비교하면 3 개 모델의 random ensemble 의 수익률, 표준편차, 복리 수익률 등의 평균값이 target 이 2 인 경우가 조금 낮다. (좀 더 실험 필요)

  2019 년 실험결과 3 target 에 비해서 최고 수익 앙상블은 3.3 → 3.72 로 최고 복리 수익은 11 → 17 배로 증가

- Target 숫자를 3 개에서 2 개로 줄여서 한 트레이딩 종목들
  - 알고리즘트레이딩 2 → 알고리즘 2-1
  - Stock_1D → kakao_1D(2), Samsung_1D(2)
  - Coint3 → coin3(2)

◆ 위의 3 가지 종목들에 대하여 target 숫자의 변경으로 인한 train 시 target 을 생성하는 make_train 모듈과 prediction 을 reinforcing 하는 모듈 (make_reinfo)에 대한 수정이 필요했음.

◆ 코드 예시

< make_train>

```
def make_train_data():
    df = pd.read_csv(df_raw_path, encoding='euc-kr')
    norm_df = pd.read_csv(df_pred_path, encoding='euc-kr')

    # target data 생성
    if target_type == 'HL' or target_type == 'C':
        # target column 생성 : 1:고점, 2:저점
        고저점 = []  # [0 for i in range(future_day-1)]

        for i in range(len(df)):
            if i > len(df) - future_day:
                if df.loc[i, '종가'] >= np.mean(df.loc[i + 1:len(df) - 1, base1]):
                    고저점.append(0)
                elif df.loc[i, '종가'] <= np.mean(df.loc[i + 1:len(df) - 1, base2]):
                    고저점.append(1)
                else:
                    고저점.append(1)
            else:
                if df.loc[i, '종가'] >= np.mean(df.loc[i + 1:i + future_day - 1, base1]):
                    고저점.append(0)
                elif df.loc[i, '종가'] <= np.mean(df.loc[i + 1:i + future_day - 1, base2]):
                    고저점.append(1)
                else:
                    고저점.append(0)
        df['고저점'] = 고저점
        norm_df['고저점'] = 고저점[19:]

    elif target_type == 'P':
        # target column 생성 : 1:고점, 2:저점
        고저점 = []

        for i in range(len(df)):
            if i > len(df) - future_day - 1:
                if df.loc[i, '종가'] >= df.loc[len(df) - 1, '종가']:
                    고저점.append(0)
                elif df.loc[i, '종가'] <= df.loc[len(df) - 1, '종가']:
                    고저점.append(1)
                else:
                    고저점.append(0)
```

```
                else:
                    if df.loc[i, '종가'] >= df.loc[i + future_day, '종가']:
                        고저점.append(0)
                    elif df.loc[i, '종가'] <= df.loc[i + future_day, '종가']:
                        고저점.append(1)
                    else:
                        고저점.append(0)
            df['고저점'] = 고저점
            norm_df['고저점'] = 고저점[19:]
        else:
            print("error!! no target type")
            exit(1)

        df.to_csv(df_raw_path, index=False, encoding='euc-kr')
        norm_df.to_csv(df_pred_path, index=False, encoding='euc-kr')
        norm_df.to_csv(norm_df_path, index=False, encoding='euc-kr')
```

make_reinfo

```
import numpy as np
import random
import pandas as pd

pred_term = 20
target_type = 'C'

th = 0.5

def make_train_data(df):
    if target_type == 'HL':
        base1 = '고가'
        base2 = '저가'
    else:
        base1 = '종가'
        base2 = '종가'
    # 고저점 예측 - 0: 정상  1: 고점 2: 저점 until pred_term
    target = []
    if target_type == 'P':
        for i in range(len(df)):
            if i > len(df) - 1 - pred_term:
                if df.loc[i, '종가'] > df.loc[len(df) - 1, '종가']:
                    target.append(0)
                elif df.loc[i, '종가'] < df.loc[len(df) - 1, '종가']:
                    target.append(1)
                else:
                    target.append(0)
            else:
                if df.loc[i, '종가'] > df.loc[i+pred_term, '종가']:
                    target.append(0)
                elif df.loc[i, '종가'] < df.loc[i+pred_term, '종가']:
                    target.append(1)
                else:
                    target.append(0)
    else:
```

```
            for i in range(len(df)):
                if i > len(df) - 1 - pred_term:
                    if 0 > np.average(np.array(df.loc[i+1:len(df)-1, base1].values).astype(np.float) -
float(df.loc[i,'종가'])):
                        target.append(0)
                    elif 0 < np.average(np.array(df.loc[i+1:len(df)-1, base2].values).astype(np.float) -
float(df.loc[i, '종가'])):
                        target.append(1)
                    else:
                        target.append(0)
                else:
                    if 0 > np.average(np.array(df.loc[i+1:i+pred_term-1, base1].values).astype(np.float)
- float(df.loc[i, '종가'])):
                        target.append(0)
                    elif 0 < np.average(np.array(df.loc[i+1:i+pred_term-1,
base2].values).astype(np.float) - float(df.loc[i, '종가'])):
                        target.append(1)
                    else:
                        target.append(0)

        return target

def reinfo(pred, pred_results):
        global th

        target = make_train_data(pd.DataFrame(pred_results, columns=['date', 'results', '시가', '고가',
'저가', '종가']))

        new_pred = []
        for i in range(len(pred)):
            n = i - pred_term
            cnt = np.zeros(shape=(3, 3), dtype=int)
            for j in range(i+1):
                if j <= n:
                    cnt[target[j], pred[j]] += 1
                elif j > n and float(pred_results[i, 5]) - float(pred_results[j, 5]) > 0:
                    cnt[1, pred[i]] += 1
                elif j > n and float(pred_results[i, 5]) - float(pred_results[j, 5]) < 0:
                    cnt[0, pred[i]] += 1
                else:
                    cnt[0, pred[i]] += 1

            if cnt[pred[i], pred[i]] < np.array(cnt[pred[i], :]).sum() * th:
                if pred[i] == 0:
                    new_pred.append(1)
                else:
                    new_pred.append(0)
            else:
                new_pred.append(pred[i])

        return np.array(new_pred)
```

* 빨간색 부분은 예측한 값을 변경할 것인지 그냥 쓸 것인지를 결정하는 알고리즘이다.

매수 매도 각각의 Precision 수치에 따라 별도로 처리한다. 그러나 매수, 매도 각각 처리하는 것보다 전체 precision 에 따라 일괄적으로 처리하는 것이 performance 가 더 좋게 실험 결과가 나왔다.

```
If cnt[1, 1] + cnt[0, 0] < np.array(cnt).sum() * th:
    If pred[i] == 0:
            New+pred.append(1)
  Else:
            New_pred.append(0)
 Else:
       New_pred.append(pred[i])
```

그리고 coin 은 매수 예측에 대한 precision 값에 따라 예측 값 조정을 하는 것이 preformance 가 더 좋게 나왔다.

```
If cnt[1, 1] < np.array(cnt[:, 1]).sum() * th:
    If pred[i] == 0:
            New+pred.append(1)
  Else:
            New_pred.append(0)
 Else:
       New_pred.append(pred[i])
```

- 정리를 해 보자.
  - 선물의 경우는 예측 값 매수, 매도 각각 precision 평가와 전체 precision 평가와의 차이는 미미한 것 같다. 개별 평가가 좀

낮은 것 같은데 그것은 종목별, 상황 별로 다른 결과가 나올 것 같아 단정할 수는 없다.

- 코인이나 주식의 경우 전체 평가는 아닌 것 같고 코인의 경우 매수만 평가가 좋다. 주식의 경우는 개별 평가와 매수만 평가가 비슷한 것 같다. 그러나 이것은 전체 시장이 하락세이기 때문이 아닌가 싶다. 상승장, 하락장이 골고루 포함된 광범위한 구간에서의 평가가 이루어져야 할 것이다.

**** 현재 coin3 는 매수 예측만 조정. 매수로 예측한 경우 과거 매수 예측 시 가장 많은 실제 시그널로 대체한다.

**** 현재 stock_1D 는 매수, 매도, 중립 모두 과거 예측 시 실제 시그널이 전체의 50%이상인 것으로 대체한다.

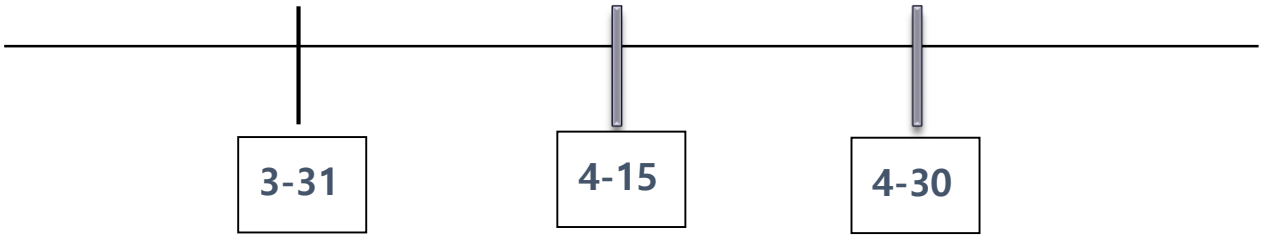
*** 기존 알고리즘트레이딩 개선 알고리즘트레이딩의 차이:**
target type 'HL'에서 차이를 보임

기존: 현재 종가가 미래 고가 평균보다 높을 때 매도 아니면 매수

개선: 현재 종가가 미래 고가 평균보다 높을 때 매도 미래 저가 평균보다 낮을 때 매수, 그 외엔 이전 시그널을 유지

사실 코스피 선물 지수를 조사해 보면 종가가 고가 평균 보다 낮거나 저가 평균 보다 높은 경우(매수, 매도도 아닌 경우)의 수는 전체의 1/5 정도이다. 그 중 매수 시그널이 약간 높다. 따라서 기존과 개선의 차이는 미미하다.

- Target 숫자를 3 에서 2 로 줄이는 것은 약간의 개선 효과가 있을 것으로 보이나 정책적인 관점을 고려하면 타겟의 숫자를 굳이 2 개로 줄일 필요는 없다고 본다.
- 학습 및 모델 적용 전략에 대한 제고
  지금까지는 15 일마다 학습하여 모델을 생성하였다.



  모델 이름이 03-31/**_**_best 이면 03-31 까지 학습하고 04-01~ 기간을 test 한 모델,
  모델 이름이 04-15/**_**_best 이면 04-15 까지 학습하고 04-16~ 기간을 test 한 모델 이었는데

  따라서 매달 말일 또는 15 일 되기 까지는 학습이 이루어지지 않았다.

2023 년 5 월 부터는

필요하다고 생각한 시점에서 학습을 수행한다. 즉, 오늘이 4 월 25 일이고 학습이 필요하다고 생각한다면 4 월 24 일 까지 학습하고 04-25~04-25 에 test 한 모델을 04-30 폴더 안에 넣는다. 5 월달로 진입하는 순간 04-30 폴더 내의 모델들은 다시 학습시킨다.

- Ensemble.py - 자동거래에서 시그널을 발생시키는 모듈. 이 때 해당 앙상블 모델을 평가하기 위한 현재 시간으로부터 이전 x 일까지의 예비 기간이 필요하다.

23 년 5 월 14 일 현재 평가 예비 기간은
알고리즘 2 : 15 일
알고리즘 4, 알고리즘 new : 31 일
이었는데 . . .
이는 ensemble_test 와 맞지 않는다.
왜냐 하면 create_all_term_ensemble 과 all_term_ensemble 를 통한 모든 앙상블들의 1 년 정도의 평가를 할 때 각 15 일 간격으로 되어 있는 last_train 일자로부터 평가 시점까지를 평가 기간으로 하여 예측 시그널의 적중 여부를 평가하게 되어있다. 따라서 평가 시점이 진행됨에 따라 평가 기간도 자동으로 늘어나게 되는데 앞서 기술한 실제 자동 거래 시 앙상블의 평가 기간은 15 일, 31 일로 고정되어 있어 test 결과와 상이하다.

따라서 될 수 있으면 test 결과와 유사한 결과를 기대하기 위해 last_train 으로부터 평가 시점까지의 기간을 평가 기간으로 하는 유동적 평가 시스템으로 ensemble.py 를 수정한다.

- 학습 주기, 모델 선정 주기 및 모델 선정 기준에 관하여 코스피 선물 자동 거래 시그널 발생 시스템에서 다음과 같은 것을 결정해야만 한다.

  1. 학습 주기

  2. bet 5 모델을 선정하기 위한 주기

  3. best 5 에서 best pick 선정 기준

학습 주기는 15 일 간격으로 한다. 15 일 마다 마지막 날을 train 기간의 끝으로 하여 1000 개의 데이터로 학습한다. 모델 평가는 매 주말 마지막 일자까지 모든 앙상블을 1 년간 평가하여 15 일 수익률의 1 년간 복리의 best 5 를 뽑아 다양한 reinfo rate 과 pred term 에서 다시 평가하고 과거 3~5 년간 결과의 기하평균과 곱하여 전체 기하평균을 구해서 best 5 를 선정한다.

15 일 마다 모델을 바꾸는데 이 때 모델 선정 기준은 현재 best 5 중 과거 best 5 로 선정된 횟수가 많은 모델을 선정한다. (선정 기준은 변경 가능하다.)

1. 앞에서 언급한 모델 선정의 신뢰성에 대한 실험

실험일시 : 2023-06-04 일요일
실험대상 기간: 2023-01

방법 : 각 알고리즘 유형별로 매 주기 마다 best 3 를 뽑아 적용하고 또한 매 주기 마다 random 으로 앙상블 조합과 reinfo, pred_term 을 선정하여 적용해서 두 방식을 비교한다.

실험 결과: 과거 best 선정 방식이 random 선정 방식보다 수익률이 좋다.

best3 모델 선정 실험 결과
  알고리즘 2 : 108%, 115%, 159% (평균 127%)
  알고리즘 4 : 109%, 69%, 55% (평균 77%)
  알고리즘 new : 64%, 54%, -18% (평균 33%)
random 선정 모델 실험 결과

알고리즘 2 : 92, 64 (평균 74)
알고리즘 4 : 64, 73 (평균 68)
알고리즘 new : 25, 30 (평균 27)

2. Best 모델 선정 방식에 대한 의견

과거 수익률이 좋았던 모델이라고 해서 앞으로도 좋다는 법은 없다. 그러나 모델의 건전성이 유지된다면 어느 정도는 과거 수익률을 따라가게 되어 있다. 즉 모델의 건전성이란 예측 가능한 수익률을 보이는 모델이라 할 것이다. 여기서 주장하고자 하는 가설은 모델이 불완전하면 과거 수익률에 의거한 모델 선정은 의미가 없다. 바꾸어 말하면 모델의 신뢰성은 미래의 수익률이 과거 모델의 수익률에 의존하는 확률의 정도라 할 것이다.

본 시스템은 그런 측면에서 어느 정도 신뢰성이 보장된다고 믿는다.

3. Best 5 모델을 선정하는 방법

1. 우선 고정된 reinfo 와 pred_term 으로 주어진 기본 모델들의 모든 앙상블에 대한 최근 1 년 동안의 수익률, 복리 수익률, 표준편차 등을 구한다.

2. 1 에서 복리 수익률이 큰 순서로 10 개를 뽑아 reinfo [0.3, 0.4, 0.5], pred_term [10, 20, 40] 를 각 모델에 적용한다.

3. 1 의 기간에서 2 에서 선별한 모델과 reinfo, pred_term 조합에 수익률, 복리 수익률, 표준편차를 구한다.

4. 1 의 기간 이전의 최대한 기간(2017 ~ 1 의 기간 시작 직전)에서 3 과 같은 것을 구한다.

5. 3 과 4 의 복리 수익률을 1 년 동안의 복리로 환산하여 곱을 구해서 그 값이 큰 순서로 정렬하여 best 5 를 뽑는다.

4. 여기서 잠깐....
알고리즘트레이딩 2, 4, new 에서 모델 평가를 위해 사용하고 있는 프로그램들을 설명하겠다.

1. ceate_all_term_ensemble.py
주어진 기간 동안의 기본 모델들의 주어진 reinfon, pred_term 하에서 모든 앙상블들(2, 4 는 1330 개, new 는 120 개)의 누적 수익률, 누적 15 일 단위 복리 수익률, 표준 편차를 기록한다. 중복되지 않게 랜덤으로 앙상블을 조합하여 하나씩 주어진 기간에서 15 일 단위로 평가 요소들을 계산하여 병합한다. 이 때 누적 수익률 best 가 갱신될 때 마다 best 에 해당되는 결과 파일들을 save 한다.

2. all_term_ensemble.py
1 의 결과에서 복리 수익률 기준 best 5 또는 10 의 앙상블을 뽑아 reinfo, pred_term 조합([0.3, 0,4, 0,5] X [10, 20, 40] = 9 개 조합)에서 1 의 평가 요소들을 주어진 기간에서 산출하여 모든 결과 파일들을 저장한다.

3. all_term_ensemble3.py

2 와 동일 하나 결과 파일들이 모든 거래를 포함하지 않고 15 일 단위 누적 수익률로 기록된다.

5. 또 다른 시도: reinfo, pred_term 조합([0.3, 0,4, 0,5] X [10, 20, 40] = 9 개 조합으로 fix 시키지 말고 reinfo 는 0 ~ 1 사이의 random 값으로 pred_term 은 1 ~ 40 사이의 random 값으로 여러 번 반복하여 주어진 앙상블에 적용하는 것이다.
   최적의 reinfo, pred_term 조합을 찾고자 하는 방법인 것이다.
   이것들을 trainable variable 로서 train 하는 방법도 고려하고 있지만 기술적 어려움에 봉착하여 고민 중이다.
   생각하고 있는 최적의 앙상블 찾기 process 는 다음과 같다.

   1. 우선 최근 기간(6 개월 이상 1 년 미만)의 주어진 reinfo, pred_term 에서 기본 모델들의 모든 앙상블들의 수익률(누적, 복리)을 계산하여 최고 복리 수익률의 앙상블 10 개를 선택한다.

   2. 선택된 앙상블에 대하여 번복하여 random 선택된 reinfo, pred_term 에 대하여 최근 기간에 수익률을 계산하여 최고 복리 수익률의 (앙상블, reinfo, pred_term) 조합을 10 개 선택한다.

   3. 2 에서 선택된 앙상블, reinfo, pred_term 조합에 대하여 최근 기간을 제외한 전체 기간에서 수익률을 계산한다.

   4. 2 와 3 의 복리 수익률의 기하 평균을 구한다. 이렇게 매 주마다 5 개의 best 조합을 기록하고 최근 5 개의 best 중 이전에 가장 많이 언급된 조합을 선택한다.

6. 다시 생각해 보니... 여러 조합의 reinfo, pred_term 은 번거롭기만하고 별 효과는 없을 것 같다. 하나의 reinfo, pred_term 으로 고정하여 최근 기간, 이전 기간의 기하 평균으로 앙상블을 선택하는 것이 편하다.
   그러면 어떤 reinfo, pred_term 으로 고정할 것인가 하는 문제가 남는다.

   그래서 다음과 같이 일시적으로 정해본다.

   최근에 선택된 3 개의 앙상블들에 대한 최근 15 일간 n 개의 reinfo, pred_term random 조합의 수익률을 비교하여 best 인 reinfo, pred_term 조합을 결정한다.

7. 앙상블 평가 프로그램들 설명:

   create_all_term_ensemble.py:
   21 개(알고리즘트레이딩 2, 4) 또는 10 개(알고리즘트레이딩 new)의 기본 모델로부터 가능한 모든 앙상블의 주어진 기간 동안의 수익률을 계산하다. 이 평가 기간 동안의 적용되는 모델들은 15 일마다 최신 모델들로 변경된다. reinfo 와 pred_term 은 주어진 값으로 일괄 적용된다.

   create_all_term_ensemble_ramdom_reinfo_term.py:
   21 개(알고리즘트레이딩 2, 4) 또는 10 개(알고리즘트레이딩 new)의 기본 모델로부터 가능한 모든 앙상블의 주어진 기간 동안의 수익률을 계산하다. 이 평가 기간 동안의 적용되는 모델들은 15 일마다 최신 모델들로 변경된다. reinfo 와 pred_term 은 random 으로 각 앙상블에 n 번씩 주어진다

all_term_ensemble.py:
주어진 앙상블들(주로 10 개)에 대하여 reinfo [0.3, 0.4, 0.5], pred_term [10, 20, 40] 조합을 적용하여 전체 90 개(앙상블이 10 개인 경우)의 앙상블들의 주어진 기간 동안의 수익률을 계산하여 eval_reflection_[term]_****.csv 에 저장하고 개별 앙상블들에 대한 주어진 기간 동안의 거래 history 를 각각 저장한다.

all_term_ensemble2.py:
하나의 앙상블에 대하여 주어진 reinfo, pred_term 값, 또는 random reinfo, pred_term 값(매 15 일 간격 구간 마다 reinfo [0.3, 0.4, 0.5] 중 택일, pred_term [10, 20, 40] 중 택일)을 적용하여 주어진 기간 동안의 거래 history 를 저장한다.

all_term_ensemble3.py:
주어진 앙상블, reinfo, pred_term set 에 대하여 또는 random 앙상블, reinfo, pred_term 값(매 15 일 간격 구간마다 reinfo [0.3, 0.4, 0.5] 중 택일, pred_term [10, 20, 40] 중 택일)을 적용하여 주어진 기간 동안의 거래 history 를 저장한다. 거래 history 는 all_term_ensemble, *2 는 매 거래 시 마다 기록하는 반면 all_term_ensemble3 는 15 일 수익률을 기록한다.

all_term_ensemble_random_reinfo_term.py:
주어진 앙상블 대하여 또는 random reinfo, pred_term 값 (reinfo 0 ~ 1, pred_term 1 ~ 40 중 택일)을 n 번 적용하여 주어진 기간 동안의 거래 history 를 저장한다.

8. 적용 앙상블 선택의 예를 들어 설명하자면

    1. 2023-01-01 ~ 2023-06-23 기간의 모든 앙상블들 (reinfo = 0.3, pred_term = 40)의 수익률을 계산한다. – create_all_term_ensemble.py

    2. 복리 수익률 best 10 을 뽑는다.

    3. 2023-01-01 ~ 2023-06-23 기간의 reinfo [0.3, 0,4, 0,5], pred_term [10, 20, 40] 값을 차례로 주어 best 10 앙상블에 적용한 후 수익률을 계산한다. –all_term_ensemble.py

    4. 2017-01-01 ~ 2022-12-31 기간의 reinfo [0.3, 0,4, 0,5], pred_term [10, 20, 40] 값을 차례로 주어 best 10 앙상블에 적용한 후 수익률을 계산한다. –all_term_ensemble.py

    5. 3 과 4 의 기하평균을 구하여 복리 수익률 기준 best 5 를 뽑는다.
    6. 최근 best 5 선정 history 을 참고하여 가장 많이 선정된 모델에 대하여 최근 1 달 동안의 가능한 모든 reinfo (0 ~ 1), pred_term(1 ~ 40)에 대하여 수익률을 계산한다. (random 으로 1000 번 반복) – all_term_ensemble_random_reinfo_term.py

    7. 6 에서 복리 수익률 기준 최고의 reinfo, pred_term 을 선정된 모델에 적용한다.

9. 최적의 reinfo, pred_term 을 먼저 찾아서 기존 방식에 적용하기

최근 기간의 (예: 2023-01-01 ~ 2023-06-23) random 앙상블, reinfo, pred_term 으로 1000 번 시뮬레이션을 실행. 상위 n 번째 수익률의 평균 reinfo, pred_term 을 계산하여 찾은 reinfo, pred_term 으로 create_all_term_ensemble 을 실행한다.

앞서 실행의 결과 중 상위 10 개의 앙상블에 적용된 reinfo, pred_term 값에 상 하 1 개씩을 보태 3 개씩 2 개의 set 를 만든다. 이 2 개의 set 의 조합에 대하여 선택된 10 개의 앙상블에 all_term_ensemble 을 최근 기간 그 이전 기간에 적용하여 연율 기하 평균을 구해 best 5 을 선정.

매 주말 마다 선정 된 best 5 를 선정해서 지금까지 가장 많이 선정된 앙상블, reinfo, pred_term 을 실전에 적용한다.

10. 다시 요약하자면...

최적의 reinfo, pred_term 을 찾는다.

최근 기간에서 최고 수익률의 앙상블들을 찾는다.

그 앙상블에 대해 최적의 reinfo, pred_term 주변의 수익률을 계산한다.

최근 기간과 그 이전 기간에서 수익률을 각각 계산하여 기하 평균을 구하여 best 5 를 기록한다.

이 같은 작업을 매 주마다 실행한다.

기록된 best 5 로 가장 많이 언급된 앙상블, reinfo, pred_term 조합을 선택하여 적용한다.

11. Best 모델 선정 프로그램 실행 절차

all_term_ensemble_random_reinfo_term.py :
최적의 reinfo, pred_term 선정

create_all_term_ensemble.py : 최적의 reinfo, pred_term 에 대하여 모든 앙상블 수익률 계산. Best 3 선정

all_term_ensemble.py : 선정된 best 3 에 대하여 reinfo [0.3, 0.4, 0,5, 0.5, 0.7], pred_term [5, 10, 15, 20, 25, 30, 40] 조합에 대한 수익률 계산. 최근 기간과 이전 기간에 대하여 수익률 각각 계산하여 기하평균 값이 큰 best 5 를 선정하여 기록

12. 또 다른 방법은...

최근 기간 이전의 기간 (최근 기간이 2023-01-01~현재일 때, 2017~2022)의 모든 앙상블 (reinfo, pred_term 은 고정)의 수익률을 계산하여 복리 수익률 기준 best 10 을 뽑는다. – all_term_ensemble.py

앞에서 선정된 best 10 에 대하여 최근 기간에서 reinfo [0.2, 0.3, 0.4, 0.5, 0.6, 0.7, 0.8], pred_term [5, 10, 15, 20, 25, 30, 40]의 모든 조합에 대하여 수익률을 계산하여 복리 수익률 기준 best 10 의 앙상블, reinfo, pred_term 조합을 뽑는다. – all_term_ensemble.py

앞에서 선정된 best 10 조합에 대하여 이전 기간의 수익률을 계산하여 (all_term_ensemble3.py) 최근 기간의 수익률과의 기하평균을 구하여 best 3 를 각각 ensemble.py, ensemble1py, ensemble2py 에 넣는다.

13. 사실 먼저 최적의 reinfo, pred_term 을 찾는다는 것은 의미가 없어 보인다. 왜냐하면 앙상블마다 reinfo, pred_term 은 제 각각 이기 때문이다. 그래서 모든 앙상블과 reinfo, pred_term 조합에 대하여 최상의 best 10 을 최근 기간에서 찾고 – all_term_ensemble_random_reinfo_term.py – 그 best 10 의 이전 기간의 년 복리를 계산하여 둘의 곱을 구하여 최고의 것을 선정한다.

    이 방법으로 2023 년 6 월 30 일부터 7 월 10 일 까지 적용한 결과 7 월 3 일을 제외하고 모두 수익을 기록하였다.

14. 앞서의 방법 또한 무시하지 않고 병행하여 적용하는 것도 고려 중이다. Reinfo, pred_term 조합 [0.3, 0.4, 0.5]-[10, 20, 30, 40] (알고리즘트레이딩 2, 4, 알고리즘트레이딩 new 의 pred_term 은 [2, 3, 4, 5, 6, 7])과 모든 앙상블에서 2000 번 이상 random 으로 중복없이 최근 기간의 복리를 계산하여 -- all_term_random_reinfo_term.py -- best n 에 가장 많이 출현하는 reinfo 와 pred_term 을 선정하여 모든 앙상블에서 최근 기간의 복리 수익률 중 best 10 을 선정하고 – create_all_term_ensemble.py -- 그 best 10 에 대하여 최근 기간과 그 이전 기간의 reinfo, pred_term 조합에 대한 수익률을 모두 구한 다음, -- all_term_ensemble.py -- 년 복리 수익률 기준으로 둘의 기하평균을

구해 best 5 를 선정 매 주 기록한다. Best 5 중 가장 많이 언급된 모델 조합을 선택한다.

15. 앞서의 모든 방법들의 공통점을 종합해서 정리하자면. . .

    1. 최근 기간의 복리 수익률 기준으로 모델 조합 (ensemble, reinfo, term) best n 개를 선정 (모델 조합 선정은 고정된 reinfo, term 에서 모든 ensemble 을 조사한 후 전체 reinfo, term 에 대해 조사, 또는 전체 모델 조합에서 random 으로 조사)

    2. 1 의 n 개에 대하여 이전 기간의 복리 수익률을 구함

    3. 1 과 2 의 기하평균순으로 best m 을 선정

    4. 3 의 선정된 모델 중에서 앞서의 선정 history 를 참고하여 가장 많이 참조된 모델 선택

16. 2023 년 8 월 8 일 현재 진행 상황

15 일 마다 알고리즘트레이딩 2, 4, new 를 학습하는 것은 전과 동일하다. 모델의 선택 주기는 매 주말, 1 주일 마다 진행한다. 매 주말 마다 2 가지 서로 다른 방식으로 각 트레이딩 프로그램에서 모델 조합을 선택한다.

첫 번째 방식은 앞서 설명한 random select 된 1000 개의 모델 조합의 최근 1 년, 그리고 그 이전 기간의 복리 수익률의 1 년 기하평균을 기준으로 선택한다. 자세한 process 는 앞의 내용을 참조.

두 번째 방식은 적정 self-reflection 과 evaluation term(pred_term)으로 고정한 후 가능한 3 개 기본 모델의 모든 앙상블 조합 (1330 개, 또는 120 개)을 조사한 후 1 년 동안의 복리 수익률 기준으로 10 개를 선택하고 그 10 개 앙상블에 대하여 주어진 self-reflection, pred_term 범위에서 다시 최근 1 년, 그 이전의 복리 수익률에 대한 1 년 기하평균을 서로 곱하여 전체 수익률을 산출하여 best 5 개를 선택한다. 그 중에서 과거 참조가 많은 것이 선택된다.

첫 번째 방식에서는 최고의 수익률이 선택되지만 두 번째 방식에서는 최고 수익률이 아니더라도 과거 참조가 많은 것이 선택된다.

17. 현재 진행 상황에 대한 반성

2 가지 방식으로 동시에 진행 하는 것이 중복의 개념이 강할 수 있다. 물론 2 가지 방식에 선택되는 모델 조합이 서로 다른 경우가 대부분이지만 어는 방식의 모델 조합을 선택할 것인지에 대한 고민이 존재한다. 2 모델 중 최근 몇 일 동안의 수익률을

참조하기는 하지만 미래 상황에 대한 절대적 가치를 부여하지는 않을 것이다.

따라서 선별 과정을 다소 간소화 할 필요가 있을 듯싶다. 매 주마다 선별 작업을 하여 적용 모델을 바꾸어 주는 것은 성급한 것이 아닌 가 싶다. 15 일 마다 한 번씩 바꾸어 주되 중간에 수익률이 안 좋을 때는 그 이전 선택 모델로 잠시 대체하는 방법이 추천된다.

그리고 첫 번째 방식은 3 번의 수익률 조사 process 가 필요하지만 두 번째 방식은 2 번의 과정만이 필요하므로 시간적으로 유리할 것이다.

정리하자면 15 일 마다. 학습 process, 그리고 최근 1 년과, 그 이전 기간의 수익률 조사 process, 총 3 개의 프로세스가 3 개의 트레이딩 알고리즘에 각각 적용되어 3 개씩 모델이 선택되고 history 가 관리된다. 따라서 각 트레이딩 알고리즘 마다 시간 순으로 3 개의 모델 조합과 관련된 예측 프로그램들이 별도로 존재하게 되어 총 9 개의 예측 프로그램이 존재하게 된다. 이중 2 ~ 3 개의 프로그램이 선택되어 결과가 합성되어 최종 시그널과 진입 계약 수가 결정된다.

18.2023 년 8 월 11 일 현재 수익률 현황 및 분석

2022 년 1 월 20 일부터 2023 년 8 월 11 일까지의 수익률이 236.488%로 기록이 되었다. 그러나 이것은 단일 1 계약 기준으로서 실제로는 매 거래 시 계약 수가 1 ~ 4 계약으로 다양하다.

위의 그래프에서 보듯이 2023 년 1 월 ~ 4 월까지의 수익률이 마이너스를 기록하는 것으로 보이지만 실 거래 계약 수에 의한 증거금 대비로는 약 66%의 수익으로 계산된다.

즉, 손해 날 때는 적은 계약이 들어 갔고 이익이 날 때는 많은

계약이 진입했다는 것으로 이유를 들 수 있다.

19. 2023 년 8 월 18 일 중대한 실수 발견

모델의 prediction 값을 조정하는 make_reinfo.py 에서

전달 받은 input data 에 대한 즉석 target data 생성에서 중대한 오류 발견.

첫 번째 data 에 관한 target 값은 prediction term 만큼 경과한 봉수에 해당되는 시점의 지수 값을 참조하여 설정되는데 만약 현재 시점이 the first data + prediction term 이전 이라면 설정된 target 값은 현재 시점에서는 알 수 없는 미래 시점의 값을 참조해서 만드는 셈이 된다.

이것은 hit-ratio probability 을 계산하는데 있어 잘못 된 정보를 주게 됨으로 정확한 probability 값을 산출할 수 없게 된다.

따라서 실제 결과 보다 훨씬 좋은 수익률을 보이는 듯한 환상을 불러 일으킨다.

다행히 알고리즘 4, new 에서만 이런 오류가 발견되었고 알고리즘 2 는 정상 가동되어 진행해 왔다.

20. 새로운 selection 알고리즘의 적용

가능한 모든 모델 조합의 수익률 및 복리 수익률을 조사한다.

[모든 앙상블 조합, reinfo: [0.3, 0.4, 0.5], pred_term [10, 20, 30, 40] (알고리즘 4), [10, 20, 30] (알고리즘 2) [2, 3, 4, 5, 6, 7] (알고리즘 new)]

이렇게 하면 알고리즘 2 는 11970 개, 알고리즘 4 는 15960 개, 알고리즘 new 는 216 개의 가능한 모델 조합 수가 존재한다. 각 알고리즘 당 최소한 5000 개(알고리즘 new 는 전부)의 모델 조합을 조사해야 의미 있는 조사가 되리라 생각한다.

다음은 순서로 적용 모델 조합을 선택하는 과정이다.

1. 가능한 많은 모델 조합의 수익률(복리 포함)을 최근 6 개월 ~ 1 년 동안 조사하여 복리 수익률 기준 best 100 을 뽑는다.

2. 1 의 best 100 에 대해 그 이전 기간의 수익률을 조사한다. 1 과 2 의 복리 수익률의 곱이 높은 순으로 5 개를 선정한다.

3. 5 개 중 이전 참조 횟수를 고려하여 선정한다.

프로그램 upgrade

알고리즘트레이딩 4, new → 알고리즘트레이딩 4-1, new-1

<주 변경 내용>

target number 2 를 3 으로 수정 (0: 중립, 1:매도, 2:매수)

중복 모듈 하나로 통합

hit-ratio 를 계산하는 evaluation 기간을 유동적으로 조정

<모듈별 설명>

- data.py : 필요한 변수 모두를 config class 에 포함. 변수값 설정, Preprocessing 함수
- model.py : model creation, train, test, prediction 함수
- ensemble.py : 전체 앙상블 기간 (last_trains, start_times, end_times), 주어진 전체 기간의 앙상블 수익률 계산하는 main process, 주어진 단일 기간의 앙상블 prediction
- create_all_term_ensembles.py : 주어진 기간과 reinfo, term 에서 모든 앙상블의 수익률 조사
- all_term_ensembles.py : 주어진 앙상블 set 에서 주어진 기간의 reinfo 집합 [0.3, 0.4, 0.5]과 prediction term 집합 [10, 20, 30, 40] 의 모든 조합에 대한 수익률 조사
- all_term_ensembles2.py : 주어진 앙상블, reinfo, prediction term

에서

주어진 기간의 수익률 조사

- all_term_ensembles3.py : 주어진 앙상블, reinfo, prediction term 조합

에서 주어진 기간의 수익률 조사

- all_term_ensembles_random_reinfo_term.py : 주어진 기간에서 무작위

앙상블, reinfo, prediction term 의 수익률 조사

- After-train parameters

  학습된 모델을 적용할 때 모델의 원 예측 값 대신 강화된 예측 값으로 변경하게 되는데 이것을 reinforced prediction 이라 칭하겠다.

  reinforced prediction 의 방법은 다음과 같다.

  과거 일정 기간의 예측 결과값과 target 값을 비교하여 정확도를 계산한다. 정확도가 주어진 기준 이하인 경우 예측 값을 반대로 변경한다. (매수는 매도로 매도는 매수로 중립은 그대로)

  이 때 발생하는 parameter 들을 after-train parameter 들이라 칭하고 다음과 같은 것 들이 있다.

  1. Hit-ratio : 예측 값을 변경하게 되는 기준. 예측 정확도가 이 값 이하이면 예측 값 반대로 변경

  2. Pred_term : 예측 값들에 대한 target 값을 설정하기 위한

예측 term

3. Width : 예측 정확도를 구하기 위하여 조사하는 구간의 길이 (봉수)

4. Loss-cut : 예측에 따른 수익 계산시 적용되는 손절 rate

After-train parameter들을 최적화 하기

Bayesian optimization 적용 코드

```
def ensemble_predict(reinfo, term, loss_cut, width):

    conf.reinfo_th = reinfo
    conf.pred_term = int(term)
    conf.loss_cut = loss_cut
    conf.reinfo_width = int(width)

    conf.gubun = 0

    print('data processing start...')
    now = datetime.datetime.now()
    print(now)
    data.preprocessing(conf)
    print('data processing end...')
    r = ep.predict(conf)
    print(r)

    return(r)

def best_params():
    # 파라미터의 타입을 설정합니다.
    param_space = {
        "reinfo": (0, 1),
        "term": (1, 40),
        "loss_cut": (0.005, 1),
        "width": (10, 77),
    }

    # 베이지안 최적화를 수행합니다.
    bo = BayesianOptimization(
        f=ensemble_predict,
        pbounds=param_space,
        random_state=0,
    )
    bo.maximize(init_points=10, n_iter=10)
```

```
    return bo.max

if __name__ == "__main__":

    data.set_start_end_time(conf, '2023/09/01/09:00', '2023/09/08/15:00', '2023-08-31')
    data.set_path(conf)
    data.set_ensemble(conf, 0.4, ["15HL", "20P", "30C"])
    best = best_params()

    print(best)
```

- Best ensemble 선정 절차

  1. All_term_ensembles_random_reinfo_term.py 로 1000 개의 random ensemble, reinfo, pred_term, width 의 최근 1 년(올해가 23 년이면 22 년 또는 23 년 1 월부터 현재까지)의 수익률을 조사해서 best 10 의 reinfo, pred_term, width 의 평균을 구한다.

  2. Create_all_term_ensemble.py 로 1 의 reinforcing parameter 를 가진 앙상블을 랜덤으로 그 이전 기간을 포함해서 1000 개의 수익률을 조사하여 best 10 을 구한다

  3. All_term_emsembles.py 로 2 에서 선택된 앙상블에 대한 조사 기간 이후 현재까지의 수익률을 조사하여 그 이전 기간의 년 기하 평균과의 곱이 최상인 것을 선택한다.

4. 3 에서 선택된 앙상블에 대하여 최근 한달 동안의 수익률을 object 로 하는 베이시안 최적화를 실행하여 최적의 파라미터를 setting 한다.

* 수익률은 복리 수익률; 베이시안 최적화 대상 파라미터들은 reinfo_th, pred_term, width, loss_cut

- 알고리즘트레이딩 4-1 upgrade

  target 숫자를 2 에서 3 으로 (중립, 매수, 매도)

  DMI, Slowk 의 normalization 을 하지 않았던 것을 다른 변수들과 똑 같이 normalization (23 년 9 월 13 일)

- Best ensemble 선정 개략적 절차

  1. 최근 기간의 모든 reinforcing parameter 들을 포함한 random 앙상블 조합의 수익률을 조사한다.

  2. 2 에서 복리 수익률 기준 best n 의 reinforcing parameter 들의 평균을 구한다.

  3. 2 에서 구한 reinforcing parameter 로 1 의 기간을 제외한 전체 기간에서 전체 앙상블들에 대한 수익률 조사 best n 을 선정

4. 3 의 best n 의 reinforcing 앙상블 조합에 대한 최근 기간의 수익률 구해서 3 의 년 기한 평균의 값과의 곱이 최상인 것을 선택한다.

- 문제점 발견 및 개선

  알고리즘트레이딩 4, 4-1, new, new-1 에서 DMI 와 Slowk, D 관련 데이터들은 % 단위로 표시가 되는데 preprocessing 시 normalization 없이 그냥 사용해 왔다.

  문제점 : 다른 변수들은 normalization 을 통해 -2 ~ 2 사이의 값을 갖는데 앞서 언급한 변수들의 값은 1 ~ 100 사이의 값을 갖는다. 이러한 불균형이 불건전한 학습으로 이어진 것으로 추정됨.

  개선 : 우선 알고리즘트레이딩 4-1 에서 모든 변수들을 20 일 단위로 standard normalization 으로 동동하게 처리함. (23 년 9 월 13 일)

  * 추후 기타 알고리즘에도 적용할 예정임

- 학습 방법의 진화 – Incremental Learning

  지금까지의 모델 학습 방법은. . .

  1000 봉의 학습 구간을 랜덤으로 50%를 선택하여 100 번 반복해서
  학습하여 accuracy 기준으로 best 인 것을 선택하는데,
  매번 시도할 때마다 모델의 weight 을 초기화한다.

  이것을 incremental learning 으로 개선

  처음 시도할 때는 이전 월의 모델 weight 으로 시작하고 그 후로는
  학습된 모델을 계속 이어서 학습한다.

  일봉 기준의 거래 모델인 알고리즘 1D-1 (3 targets, qhedge PC 에 존재)
  에 적용중

- !!!!!!!! 모델 학습 시 ERROR 발견 – Deburgging 2023-09-23

  모델 학습을 100 번 반복 실행하여 평가 결과가 최상인 것을 선택하게 되는데 평가 방법에서 ERRor 발생

  알고리즘 version *-1 은 기존 version *에서 target 이 2 (매수, 매도)이었던 것을 target 3 (중립, 매도, 매수)로 변경. 따라서 모델 학습의 평가 방법도 변경 필요. 그러나 기존 방식대로 매수, 매도 일치 여부만 (target 3 기준으로는 중립, 매도 만) 평가. → 중립, 매수, 매도 3 개 시그널 일치 여부 평가로 변경

  알고리즘 1D-1 은 전체 모델 재 학습 중
  나머지 version 들은 최근 월 모델만 incremental learning 이 아닌 기존 방식으로 변경된 시그널 일치 여부 평가로 재 학습

  알고리즘 1D-1 실험결과 – 2023-09-24

  2017~2022 구간에서 reinfo_th 0.06, pred_term 24, width 41 로 random ensemble 들의 수익률 조사한 결과 incremental learning 이 다소 높은 수익률을 보였으나 2023-01-01 ~ 2-23-09-20 구간에서 매월 Bayesian 으로 선택된 파라미터들을 적용한 결과는 좋지 않았다.

- Incremental Learning 의 단점

  1. Overfitting 의 위험

  2. 최신 경향 보다 과거 경향에 비중이 높아 질 수 있다.

  3. 너무 빨리 정점(accuracy = 1)에 도달한다.

- The best selection of training
  (중립, 매도) accuracy vs. (중립, 매도, 매수) accuracy

  반복되는 training 과정에서 best training 을 선택하는 기준으로 (중립, 매도) 일치 여부의 validation accuracy 를 적용하였는데 (이것은 일종의 실수) 이것을 모든 시그널에 대한 일치 여부 accuracy 로 수정하여 재 학습한 결과

  알고리즘 4-1 과 알고리즘 new-1 의 재 학습 결과는 재 학습 전에 비해 향상되지 못한 것으로 (오히려 떨어지는 것으로) 나타났다.

  그 이유를 분석하자면. . .

  3 개의 시그널을 모두 일치 시키고자 하는 학습 방법은 오히려 모델의 정체성을 혼란스럽게 하는 경향이 있는 것 같다.

  따라서 중립과 매도 시그널만 즉 하향 방향성에만 집중해서 학습된 모델은 reinforcing 조정 과정에서 이점이 있는 것으로 생각된다.

  그리하여 기존의 (중립, 매도) 시그널 accuracy 만으로 validation 하는 학습을 고수한다.
-

Make_reinfo2.py

전체 시그널에 대한 정확도를 따지지 않고 각각의 시그널에 대한 예측 정확도를 따져서 기준치 이하이면 해당 시그널로 예측한 경우 중 가장 많은 실제 시그널로 바꾸어준다.

```
import numpy as np
import random
import pandas as pd

pred_term = 40
target_type = 'C'

th = 0.5

def make_train_data(df):

    # 고저점 예측 - 0: 정상  1: 고점 2: 저점 until pred_term
    target = []

    for i in range(len(df)):
        if i > len(df) - 1 - pred_term:
            if float(df.loc[i, '종가']) > float(df.loc[len(df) - 1, '종가']) * 1.009:
                target.append(1)
            elif float(df.loc[i, '종가']) < float(df.loc[len(df) - 1, '종가']) * 0.991:
                target.append(2)
            else:
                target.append(0)
        else:
            if float(df.loc[i, '종가']) > float(df.loc[i+pred_term, '종가']) * 1.009:
                target.append(1)
            elif float(df.loc[i, '종가']) < float(df.loc[i+pred_term, '종가']) * 0.991:
                target.append(2)
            else:
                target.append(0)

    return target

def reinfo(pred, pred_results, start_time, reinfo_width):
    global th

    df = pd.DataFrame(pred_results, columns=['date', 'results', '시가', '고가', '저가', '종가'])
    target = make_train_data(df)
    start_inex = df.loc[df['date'] >= start_time].index.min()

    new_pred = []
    for i in range(start_inex, len(pred)):
        n = i - pred_term
        cnt = np.array([[0, 0, 0], [0, 0, 0], [0, 0, 0]])
        for j in range(max(0, i-reinfo_width), i+1):
            if j <= n and pred[j] == target[j]:
                cnt[pred[j]][pred[j]] += 1
            if j > n and float(pred_results[i, 5]) - float(pred_results[j, 5]) > 0:
                cnt[pred[j]][2] += 1
            elif j > n and float(pred_results[i, 5]) - float(pred_results[j, 5]) < 0:
                cnt[pred[j]][1] += 1
            elif j > n and float(pred_results[i, 5]) - float(pred_results[j, 5]) == 0:
```

```
                cnt[pred[j]][0] += 1

        prob = np.zeros(shape=3)
        prob[0] = cnt[0, 0] / (cnt[0, :].sum() + 0.00001)
        prob[1] = cnt[1, 1] / (cnt[1, :].sum() + 0.00001)
        prob[2] = cnt[2, 2] / (cnt[2, :].sum() + 0.00001)

        #if th == 'r':
        #    th = random.random()
        if prob[pred[i]] < th:
            new_pred.append(np.argmax(cnt[pred[i], :]))
        else:
            new_pred.append(pred[i])
    return np.array(new_pred)
```

현재 적용되고 앙상블에 대하여 최근 1 년 동안의 profir_product * 0.1 / MDD 의 best 가 되는 reinforcing parameter 를 베이시안 최적화로 찾는다.

베이시안 최적화에 의해 찾은 파라미터로 전체 기간 동안의 best profit_product 와 MDD 를 보이는 앙상블을 찾는다.

- 모델 선택의 기본 규칙

  좋은 모델이란? 이전에도 복리 수익률과 MDD 가 좋았고 최근에도 좋은 모델. 최근 1 년에서 좋은 모델들을 찾아서 그 이전 기간의 복리 수익률과 MDD 를 조사해서 두 개 기간의 수익률의 곱을 MDD 의 합으로 나누어 최고인 것을 선택한다.

- 알고리즘 1D-1 생성
  10 월 24 일 알고리즘 1D 에서 target 이 2 개 였던 것을 3 개로 확장하여 업그레이드.

test 결과 어마어마한 결과가 나와서 의심. 역시 미래 데이터로 모델 학습한 것이 발각 됨

10 월 26 일 재 학습 중

- 일 봉 기준 예측 시스템에서 시그널 개수에 대한 고찰

  60 분봉으로 예측하는 시스템에서는 보통 15 일 마다 학습 모델을 갱신하는데 이 때 봉 개수는 15 * 7 = 105 개가 된다. 그러나 일 봉으로 예측하는 시스템에서는 한 달마다 학습 모델을 갱신하게 되어 봉 개수가 20 개 안 밖으로 상당히 적어지게 된다. 따라서 발생 시그널이 '매수', '매도'. '중립'으로 3 개를 설정하면 하나의 시그널당 평균 7 개 정도의 발생 건수가 생긴다. 따라서 evaluation 을 통해 reinforcing 하기에는 데이터가 충분하지 않아 miss information 을 유발하는 것으로 추측된다.

  좋은 performance 를 위해 일 봉 예측 시스템에서는 3 개의 시그널 대시 2 개의 시그널(매수, 매도)로 단순화 하는 것을 권장한다.

  현대 3 개의 시그널을 사용하는 일 봉 예측 시스템은 주식(카카오, 삼성)과 코스피 선물 일 예측(알고리즘 1D-1)이다.

  알고리즘 1D-1 과 알고리즘 1D(개 시그널 사용)의

PERFORMANCE 비교에 의해 알고리즘

1D-1 의 사용을 중지, 폐기한다.

- New version 추가 – 알고리즘트레이_new-1 대체

  알고리즘 5-1 – new version

  E-trend 의 강흥보 대표가 애용하는 3 개의 스토캐스틱 지표 (5, 3, 3), (10, 6, 6), (20, 12, 12)를 알고리즘 4-1 의 DMI+, DMI-, ADX, SlowK, SlowD 의 5 개 컬럼을 6 개로 대체한다.

  11 월 14 일부터 적용

  대체 이유는?

  알고리즘트레이_new-1 의 성능 및 성과가 좋지 않았음.

- 알고리즘트레이딩 2-2

  알고리즘트레이딩 2-1 으로부터 업그레이드

  - 저점 매수 시점 변경 : 현 시점으로부터 과거 pred_term 이전, 미래 pred_term 이후의 가격(또는 평균 가격) 보다 낮을 경유

  - 고점 매도 시점 변경 : 현 시점으로부터 과거 pred_term 이전, 미래 pred_term 이후의 가격(또는 평균 가격) 보다 높을 경유

  - 2-1 은 미래 시점의 가격만 비교

- 알고리즘트레이딩 2-3

  - 2-2 에서 30 분봉 데이터로 변경
  - training 시 평가를 매수, 매도 시그널 일치 여부로 변경 (기존에는 중립, 매도 일치 여부)

- 알고리즘트레이딩 version 5-3

version 5 의 특징

- ETrend 의 강흥보가 주로 언급하는 3 가지의 stccjastic 지표 데이터 (slow5 3, slow10,₩ 6, slow20 12)의 6 개 항목을 추가
- OHLC 및 거래랭, 미결제 와 그들의 파생 변수들을 합하여 총 89 개의 변수로서 분석

version -3 의 특징

- 30 분봉 데이터 사용

- 매도, 매수 시그널은 현 시점에서 과거와 미래의 주어진 term 중에서 고, 저점이 될 때 발생
- train 시 매수, 매도, 중립 모든 시그널들의 일치 확률을 가지고 학습의 능력치를 판단. 100 번 반복하여 best 선정

- make_reinfo2 만 사용
- 학습 모델 적용시 적중 여부를 따져서 예측 값을 reinforcing 하는 모듈에서 중립, 매수, 매도 모두 예측 정확도를 측정하여 기준 값 이하이면 최고 수치를 보이는 시그널로 변경

- 모델 type P 의 target 시그널 생성 방법 수정

  - 기존 : 현재 시점이 주어진 term 의 과거, 미래 시점보다 0.9% 상승한 경우 매도, 하락한 경우 매수 시그널 발생

  - 수정 : 현재 시점이 주어진 term 의 과거, 미래 시점보다 상승한 경우 매도, 하락한 경우 매수 시그널 발생

<target signal 생성 코드>

```
def make_train_data(df):

    # 고저점 예측 - 0: 정상  1: 고점 2: 저점 until pred_term
    target = []
    if target_type == 'P':
        for i in range(len(df)):
            if i > len(df) - 1 - pred_term:
                if df.loc[i, '종가'] > df.loc[len(df) - 1, '종가']:
                    target.append(1)
                elif df.loc[i, '종가'] < df.loc[len(df) - 1, '종가']:
                    target.append(2)
                else:
                    target.append(0)
            else:
                if df.loc[i, '종가'] > df.loc[i+pred_term, '종가']:
                    target.append(1)
                elif df.loc[i, '종가'] < df.loc[i+pred_term, '종가']:
                    target.append(2)
                else:
                    target.append(0)
    else:
        for i in range(len(df)):
            if i > len(df) - 1 - pred_term:
                if 0 > np.average(np.array(df.loc[i+1:len(df)-1, '종가'].values).astype(np.float) -
float(df.loc[i, '종가'])):
```

```
                    target.append(1)
                elif 0 < np.average(np.array(df.loc[i+1:len(df)-1, '종가'].values).astype(np.float)
- float(df.loc[i, '종가'])):
                    target.append(2)
                else:
                    target.append(0)
            else:
                if 0 > np.average(np.array(df.loc[i+1:i+pred_term-1,
'종가'].values).astype(np.float) - float(df.loc[i, '종가'])):
                    target.append(1)
                elif 0 < np.average(np.array(df.loc[i+1:i+pred_term-1,
'종가'].values).astype(np.float) - float(df.loc[i, '종가'])):
                    target.append(2)
                else:
                    target.append(0)

    return target
```

<반복 학습 시 prediction 정확도 평가 코드>

```
def train(model, conf):
    norm_df = pd.read_csv(conf.norm_df_path, encoding='euc-kr')

    train_end_time = datetime.datetime.strptime(conf.last_train,
"%Y-%m-%d").strftime("%Y/%m/%d/15:00")
    train_end_index = norm_df.loc[norm_df['date'] <= train_end_time].index.max() - conf.pred_term +
1

    if train_end_index-1000 < 0:
        train_df = norm_df.iloc[:train_end_index]
    else:
        train_df = norm_df.iloc[train_end_index-1000:train_end_index]

    # create train data
    train_df = train_df.drop('date', axis=1, inplace=False)
    train_data = train_df.values

    # downsampling, downsizing the data with target == 0 randomly
    drop_index = []
    r = (len(train_data) / (train_data[:, -1] == 0).sum() - 1) / 2
    for i in range(len(train_data)):
        if train_data[i, -1] == 0 and random.random() > r:
            drop_index.append(i)
    train_data = np.delete(train_data, drop_index, 0)

    pre_accu = 0
    repeat_cnt = 0
    best_profit = 0
    while repeat_cnt < conf.max_repeat_cnt:

        gc.collect()

        repeat_cnt += 1

        # 최근 데이터는 vaidation 데이터로 예약
```

```
        train_x = train_data[:, :conf.input_size]
        train_y = train_data[:, conf.input_size]

        # over_sampling
        # X_samp, y_samp = RandomOverSampler(random_state=0).fit_sample(train_x, train_y)
        # X_samp, y_samp = ADASYN(random_state=0, n_neighbors=5).fit_sample(train_x, train_y)
        #train_x, train_y = SMOTETomek(random_state=0).fit_resample(train_x, train_y)


        # train data 중 50%만 사용. . .
        #r = random.randrange(1, 10000)
        train_x, _, train_y, _ = train_test_split(train_x, train_y, train_size=conf.train_rate)#,
random_state=r)

        # valid data 10% 사용. . .
        train_x, valid_x, train_y, valid_y = train_test_split(train_x, train_y, test_size=0.1)


        # incremental learning 으로 model training
        #if repeat_cnt == 1:
        #    pre_last_train = data.get_pre_last_train(conf.last_train)
        #    model.load_weights(pre_last_train + "/30M_"+str(conf.pred_term) + conf.target_type +
"_best")
        #else:
        #    model.load_weights(conf.checkpoint_path_best)

        # target 재설정하여 model training
        model.load_weights(conf.checkpoint_path)
        early_stopping = tf.keras.callbacks.EarlyStopping(patience=2, verbose=0)
        model.fit(train_x, train_y, batch_size=conf.batch_size, epochs=conf.epochs, verbose=0,
callbacks=[early_stopping],
                  validation_data=(valid_x, valid_y))

        prediction = model.predict(valid_x).reshape(-1, conf.target_num)
        prediction = np.argmax(prediction, axis=1).reshape(-1)

        # calculate accuracy
        c = 0
        s = len(prediction)#list(prediction).count(2) + list(prediction).count(1)
        for i in range(len(prediction)):
            if valid_y[i] == prediction[i]:
                    c += 1

        if s == 0:
            accu = 0
        else:
            accu = c / s

        if repeat_cnt % 100 == 0:
            print("반복 횟수 : " + str(repeat_cnt) + " accuracy = " + str(accu))

        if accu > pre_accu:
            best_x = train_x
            best_y = train_y
            # model.fit(valid_x, valid_y, batch_size=batch_size, epochs=3, verbose=0)
            model.save_weights(conf.checkpoint_path_best)

            print("best accuracy " + str(accu))
```

```
            pre_accu = accu

    print("best accuracy " + str(pre_accu))
```

- Version 2, 4, 5 비교

  - version 2-# : 시가, 고가, 저가, 종가, 5, 20, 60, 120, 200 일 평균, 거래량, 미결제의 11 개 기본 항목에서 x 일 이전가, 수익률, 최고저가 등의 파생 변수들을 합하여 총 83 개의 input 변수들로 구성.

  - version 4-# : 시가, 고가, 저가, 종가, 거래량, 미결제의 6 개 기본 항목에서 x 일 이전가, 수익률, 이동평균, 최고저가 등의 파생 변수들과 DMI 변수 3 개, stochastic 변수 2 개를 합하여 총 88 개의 input 변수들로 구성.

  - version 5-# : 시가, 고가, 저가, 종가, 거래량, 미결제의 6 개 기본 항목에서 x 일 이전가, 수익률, 이동평균, 최고저가 등의 파생 변수들과 ETREND 강흥보의 6 개 stochastic 변수를 합하여 총 89 개의 input 변수들로 구성.

- Version -2, -3 에 대한 평가

  2023 년도에서 random ensemble, pred_term, reinfo, reinfo_width 으로 200 ~ 300 개 정도의 수익률 평균을 내보면 version -1 에 비해 떨어진다.

  2023 년 수익률 평균 비교

  version -1 : 약 2.0
  version -2 : 약 1.19
  version -3 : 약 1.5

  -2, -3 의 낮은 수익률은 과거 시점을 고려한 target 설정에 문제가 있다고 생각됨.

- 미래 시점 만을 고려한 target 설정으로 30 분봉 거래 시스템 고려.
  version -30M

  2023 년 1 월 1 일부터 12 월 1 일까지의 수익률과 MDD 를 200 ~ 300 개의 random 조합의 앙상블들의 60 분봉과 30 분봉에서 비교하면

	수익률	MDD
60 분봉		
2-1	1.99	0.375
4-1	1.91	0.39
5-1	1.72	0.44
30 분봉		
2-30M	2.0	0.629
4-30M	1.5	0.6
5-30M	1.97	0.578

**** 위의 실험 결과표에서 보듯이 30 분봉의 장점을 발견할 수가 없음. 따라서 일단 30 분봉은 보류하기로 함.

- 알고리즘트레이딩 2 와 알고리즘트레이딩 2-1 의 차이점

- target data 생성시 차이점

모델 타입 HL 과 C 에서 target 을 만들 때

알고리즘트레이딩 2 는 현재 시점에서 주어진 term 의 미래 시점까지 **최고 값 또는 최저 값과 비교**하여 매수, 매도 시그널을 결정. **2-1** 은 주어진 기간에서의 **최고 값 또는 최저 값과 비교**하는 것은 동일하나 예측 시그널을 결정하기 위한 reinforcing 시 별도의 모듈 make_reinfo2 가 있음.

알고리즘트레이딩 2 는 make_reinfo.py 에만 의존하여 reinforcing 하나 알고리즘트레이딩 2-1 은 make_reinfo 와 make_reinfo2 중에서 선택하게 되어 있음.

make_reinfo 와 make_reinfo2 의 차이는 target data 생성은 같으나 예측 시그널과 실제 시그널을 비교하는 알고리즘이 다르다는 것이다.

<make_reinfo 의 예측과 실제 시그널 비교 코드>

```
def reinfo(pred, pred_results, start_time, reinfo_width):
    global th

    df = pd.DataFrame(pred_results, columns=['date', 'results', '시가', '고가', '저가', '종가'])
    target = make_train_data(df)
    start_inex = df.loc[df['date'] >= start_time].index.min()

    new_pred = []
    for i in range(start_inex, len(pred)):
        n = i - pred_term
        cnt = 0
        for j in range(max(0, i-reinfo_width), i+1):
            if j <= n and pred[j] == target[j]:
                cnt += 1
            if j > n and float(pred_results[i, 5]) - float(pred_results[j, 5]) > 0:
```

```
                if pred[i] == 2:
                    cnt += 1
            elif j > n and float(pred_results[i, 5]) - float(pred_results[j, 5]) < 0:
                if pred[i] == 1:
                    cnt += 1
            elif j > n and float(pred_results[i, 5]) - float(pred_results[j, 5]) == 0:
                if pred[i] == 0:
                    cnt += 1

        prob = cnt / (min(i, reinfo_width) + 0.00001)

        #if th == 'r':
        #    th = random.random()
        if prob < th:
            if pred[i] == 1:
                new_pred.append(2)
            elif pred[i] == 2:
                new_pred.append(1)
            else:
                new_pred.append(0)
        else:
            new_pred.append(pred[i])
    return np.array(new_pred)
```

***** 전체 시그널의 일치 여부를 확률로 계산하여 기준치 이하이면

매수를 매도로 매도를 매수로 바꾸어준다.

<make_reinfo2 의 예측과 실제 시그널 비교 코드>

```
def reinfo(pred, pred_results, start_time, reinfo_width):
    global th

    df = pd.DataFrame(pred_results, columns=['date', 'results', '시가', '고가', '저가', '종가'])
    target = make_train_data(df)
    start_inex = df.loc[df['date'] >= start_time].index.min()

    new_pred = []
    for i in range(start_inex, len(pred)):
        n = i - pred_term
        cnt = np.array([[0, 0, 0], [0, 0, 0], [0, 0, 0]])
        for j in range(max(0, i-reinfo_width), i+1):
            if j <= n and pred[j] == target[j]:
                cnt[pred[j]][pred[j]] += 1
            if j > n and float(pred_results[i, 5]) - float(pred_results[j, 5]) > 0:
                cnt[pred[j]][2] += 1
            elif j > n and float(pred_results[i, 5]) - float(pred_results[j, 5]) < 0:
                cnt[pred[j]][1] += 1
```

```
            elif j > n and float(pred_results[i, 5]) - float(pred_results[j, 5]) == 0:
                cnt[pred[j]][0] += 1

        prob = np.zeros(shape=3)
        prob[0] = cnt[0, 0] / (cnt[0, :].sum() + 0.00001)
        prob[1] = cnt[1, 1] / (cnt[1, :].sum() + 0.00001)
        prob[2] = cnt[2, 2] / (cnt[2, :].sum() + 0.00001)

        #if th == 'r':
        #    th = random.random()
        if prob[pred[i]] < th:
            new_pred.append(np.argmax(cnt[pred[i], :]))
        else:
            new_pred.append(pred[i])
    return np.array(new_pred)
```

***** 개별 시그널들의 일치 여부를 각각 계산하여 기준치 이하이면 해당 예측 시그널의 최대 실제 시그널로 바꾸어준다. 예를 들어 매수로 예측한 경우, 과거 10 번의 매수로 예측한 결과를 실제 값과 비교하여 그 중 실제로 3 번이 매수, 5 번은 매도, 2 번은 중립이라고 하면 기준치가 0.5 이면 적중 확률이 0.3 이므로 기준치 이하가 되어 최대 시그널 값 매도로 바꾼다. 기준치가 0.2 이면 적중 확률이 기준치보다 크므로 그대로 매수를 유지한다.

- 또 하나의 차이점은

  알고리즘트레이딩 2-1 은 공통으로 사용되는 모듈 및 변수들을 data.py, model.py, ensemble_proc.py 에 넣어서 관리하지만 알고리즘트레이딩 2 는 각 프로그램마다 중복하여 모듈을 따로 갖는다

그리고 알고리즘트레이딩 2-1 은 reinforcing 시 평가 기간 reinfo_width 의 변수가 추가되어 reinforcing 이 된다.

- 최근 1 년 좋았던 앙상블 모델이 과거에도 좋은가? 라는 질문에 대한 분석

version	2-1	4-1	5-1
300 개 앙상블 모델의 과거 수익률 평균	200%	223%	47%
최근 수익률 상위 20 개 모델의 과거 수익률 평균	283%	278%	100%
최근 수익률 상위 10 개 모델의 과거 수익률 평균	278%	345%	71%

* 최근 수익률은 2023-01-01 ~ 2023-12-01 기간의 수익률

과거 수익률은 version 2-1 과 4-1 은 2017-01-01 ~ 2022-12-31 (2020 년 제외) 기간의 수익률, version 5-1 은 2021-01-01 ~ 2022-12-31 기간의 수익률

* 최근 수익률이 좋았던 앙상블 모델들이 버전과 기간에 따라 차이가 있기는 하지만 대략 50 ~ 120 프로 과거 수익률이 평균 수익률 보다 좋은 것으로 결과가 나왔다.

- 또 하나의 발견!!!!!!!!!!!!

  1. 최근 복리 수익률(1 년 정도) 상위 20 개 앙상블 모델들 중 과거 전체 수익률 (특정 구간 제외 예: 2020 년 코로나), MDD 를 고려하여 선택 .

     평가식 = 최근수익률 * 과거 수익률의 1 년 기하평균 /

     (최근, 과거 MDD 의 합)

  2. 과거 전체 복리 수익률 (특정 구간 제외 예: 2020 년 코로나), 상위 20 개 앙상블 모델들 중 최근 수익률(1 년 정도), MDD 를 고려하여 선택 .

     평가식 = 최근수익률 * 과거 수익률의 1 년 기하평균 /

     (최근, 과거 MDD 의 합)

  1 의 경우 상위 20 개 모델들의 과거 수익률 평균이 전체

평균 보다 상당한 수준으로 높게 나오지만 2 의 경우 최근 수익률 평균이 전체 평균과 비슷하다. 이는 **과거 수익률이 좋다고 최근 수익률이 좋다는 보장은 없다**는 의미이고 **최근 수익률이 좋은 모델은 과거에도 일반적으로 좋았다**는 것을 의미한다.

따라서 1 의 경우를 선택한다.

- Selection algorithm 의 건전성에 대한 고찰

현재 앞서 설명한 1 과 2 의 방법을 번갈아 가며 모의 투자를 9 월 16 일부터 12 월 5 일까지 진행했다.

총 투자금 5 억에 version 2-1, 4-1, 5-1 에 20 계약씩 투자해서 합산된 결과로 최종 주문이 나가게 하였다.

결과는 약 51,000,000 의 수익으로 총 투자금 대비 악 10%, 증거금 대비 약 12%의 수익률을 기록.

selection 알고리즘에 의한 앙상블 모델 선택이 아닌 random 선택이라면 어떻게 됐을까?

- version 2-1 은 -7%, version 4-1 은 -13%, version 5-1 은 +8%로서 합산하여 평균을 내면 -4%가 된다.

따라서 selection 알고리즘에 의한 모델 선택이 랜덤으로

모델을 선택하는 것 보다 더 좋은 결과를 보인다고 생각한다.

- 알고리즘 2 의 reinforcing

  version 2-1 과는 달리 reinforcing 하기 위한 evaluation term(reinfo_width)이 주어지지 않는다. 따라서 자동으로 주어진 result file 의 최대 데이터 건수로 설정되어 버린다. 즉 reifno_width = 무한대

- 코스피 선물 일봉 거래의 새로운 발견

- 지금까지 코스피 선물 일봉 거래는 매수, 매도 (중립 없음)가 바뀔 때 까지 무기한 보유하다가 시그널 변화가 생기면 청산하고 재 진입하는 것으로 되어 있었다.

  * 이 방법의 문제점은 계속 되는 overnight 으로 인해 손익의
    변동이 심하다는 것이다.
  * 높은 MDD 가 일반적 특징이라고 하겠다.
  * 3 개의 target (매수, 매도, 중립)으로 확장된 version
    알고리즘 1D-1 은 오히려 수익률을 더 악화 시켰다.

- 개선안 시도: overnight 하지 않고 전일 종가에서 예측한 결과로 금일 시가에 거래한 후 종가에 청산하는 방법으로 시도하였다.

  * 2021 년 ~ 2022 년 test 결과 기존 방식보다 월등한
    preformace 를 보여 주었다.
  * 기존 방식에서 2021 ~ 2022 년 수익률은 마이너스 였으나
    개선된 방식은 엄청난 플러스였다.

- 개선된 방식 여하를 떠나서 version 1D-1 은 1D 보다 나빴다. 그 이유는 target 을 생성하는 방식의 차이가

아닌가 싶다.

* version 1D 는 train 과 reinforcing 에 의한 모델 평가 시 생성

생성 방법이 같다.

* 그러나 version 1D-1 은 2 가지에서 서로 다른 방식을 취한다.

<Version 1D 의 target 생성 코드>

```
df = pd.read_csv(df_raw_path, encoding='euc-kr')
norm_df = pd.read_csv(df_pred_path, encoding='euc-kr')

# 고저점 예측 - 0: 정상  1: 고점 2: 저점 until pred_term, 단순 예측 - 0: 하락 1: 상승
if target_type == 'P':
    target = []
    for i in range(len(df)):
        if i > len(df) - 1 - pred_term:
            if df.loc[i, '종가'] > df.loc[len(df)-1, '종가']:
                target.append(0)
            else:
                target.append(1)
        else:
            if df.loc[i, '종가'] > df.loc[i+pred_term, '종가']:
                target.append(0)
            else:
                target.append(1)

    norm_df['target'] = target[19:]
else:
    target = []
    for i in range(len(df)):
        if i > len(df) - 1 - pred_term:
            if 0 > np.average(df.loc[i+1:len(df)-1, base1].values - df.loc[i, '종가']):
                target.append(0)
            else:
                target.append(1)
        else:
            if 0 > np.average(df.loc[i+1:i+pred_term-1, base1].values - df.loc[i, '종가']):
                target.append(0)
            else:
                target.append(1)
    norm_df['target'] = target[19:]
```

<version 1D-1 의 train 시 target 생성코드>

```
df = pd.read_csv(conf.df_raw_path, encoding='euc-kr')
norm_df = pd.read_csv(conf.df_pred_path, encoding='euc-kr')

# 고저점 예측 - 0: 정상  1: 고점 2: 저점 until pred_term, 단순 예측 - 0: 하락 1: 상승
if conf.target_type == 'P':
    target = []
    for i in range(len(df)):
        if i > len(df) - 1 - conf.pred_term:
            target.append(0)
        else:
            if df.loc[i, '종가'] > df.loc[i+conf.pred_term, '종가']*1.025:
                target.append(1)
            elif df.loc[i, '종가'] < df.loc[i+conf.pred_term, '종가']*0.975:
                target.append(2)
            else:
                target.append(0)
    df['target'] = target
    norm_df['target'] = target[19:]
else:
    target = []
    for i in range(len(df)):
        if i > len(df) - 1 - conf.pred_term:
            target.append(0)
        else:
            if df.loc[i, conf.base1] >= np.max(df.loc[i+1:i+conf.pred_term-1, conf.base1].values):
                target.append(1)
            elif df.loc[i, conf.base2] <= np.min(df.loc[i+1:i+conf.pred_term-1, conf.base2].values):
                target.append(2)
            else:
                target.append(0)
    df['target'] = target
    norm_df['target'] = target[19:]
```

<version 1D-1 의 reinforcing 모델 평가 시 target 생성코드>

```
target = []
if target_type == 'P':
    for i in range(len(df)):
```

```
        if i > len(df) - 1 - pred_term:
            if df.loc[i, '종가'] > df.loc[len(df) - 1, '종가']:
                target.append(1)
            elif df.loc[i, '종가'] < df.loc[len(df) - 1, '종가']:
                target.append(2)
            else:
                target.append(0)
        else:
            if df.loc[i, '종가'] > df.loc[i+pred_term, '종가']:
                target.append(1)
            elif df.loc[i, '종가'] < df.loc[i+pred_term, '종가']:
                target.append(2)
            else:
                target.append(0)
else:
    for i in range(len(df)):
        if i > len(df) - 1 - pred_term:
            if 0 > np.average(np.array(df.loc[i+1:len(df)-1, '종가'].values).astype(np.float) -
float(df.loc[i, '종가'])):
                target.append(1)
            elif 0 < np.average(np.array(df.loc[i+1:len(df)-1, '종가'].values).astype(np.float) -
float(df.loc[i, '종가'])):
                target.append(2)
            else:
                target.append(0)
        else:
            if 0 > np.average(np.array(df.loc[i+1:i+pred_term-1, '종가'].values).astype(np.float) -
float(df.loc[i, '종가'])):
                target.append(1)
            elif 0 < np.average(np.array(df.loc[i+1:i+pred_term-1, '종가'].values).astype(np.float)
- float(df.loc[i, '종가'])):
                target.append(2)
            else:
                target.append(0)

return target
```

- 요약 정리

version 1D 는 현재 시점에서 마지막 데이터 시점까지의 기간이 예측 term 보다 짧으면 마지막 시점까지의 기간을 예측 term 을 변경하여 target 을 생성

version 1D-1 은 train 시는 현재 시점에서 마지막 데이터 시점까지의 기간이 예측 term 보다 짧으면 무조건 target 을

0 으로 설정하고 reinforcing 모델 평가 시에는 version 1D 와 같다.

모델 타입 P 에 대한 target 생성에서, version 1D 는 현재 시점과 target 시점에서의 종가 차이를 따지고 version 1D-1 은 현재 시점이 target 시점의 2.5% 상승, 하락 여부에 따라 시그널이 결정된다.

모델 타입 C 와 HL 에 대한 target 생성에서, version 1D 는 현재 시점의 가격과 현재 시점 다음 시점부터 target 시점까지의 평균(C 는 종가 평균, HL 은 고가 평균 또는 저가 평균)을 비교하여 시그널을 결정하지만 version 1D-1 은 train 시에는 현재 시점과 target 시점의 고가의 최대 값(매도 시그널 발생시), 또는 저가의 최저 값(매수 시그널 발생시)과 비교하여 시그널이 결정되고 reinforcing 모델 평가 시는 version 1D 와 같이 종가의 평균과 비교하여 시그널을 결정한다.

Sorry!!!!!!!!!!!

잘못된 정보

수익률 계산 모듈 profit2 에서 매수 시그널을 매도 시그널로 잘 못 인식한 오류에서 나온 결과. 전일 종가 시그널에서 금일 시가로 거래하고 금일 종가로 청산하는 방법은 환상 이었음.

에이 좋다 말았네~~~~~~~~~

알고리즘 1D vs. 알고리즘 1D-1

알고리즘 1D-1 의 모델 pool 은 {1C, 1HL, 1P, 2C, 2HL, 2P, 3C, 3HL, 3P} 이다. 즉, 단기 예측 모델이다.

수익률 비교표를 보면

수익률 비교표를 보면

기간별 수익률과 MDD	알고리즘 1D		알고리즘 1D-1	
	수익률	MDD	수익률	MDD
2023-01-01 ~ 2023-12-15	115%	62%	-26%	-46%
2021 ~ 2022	97%	170%	-41%	113%

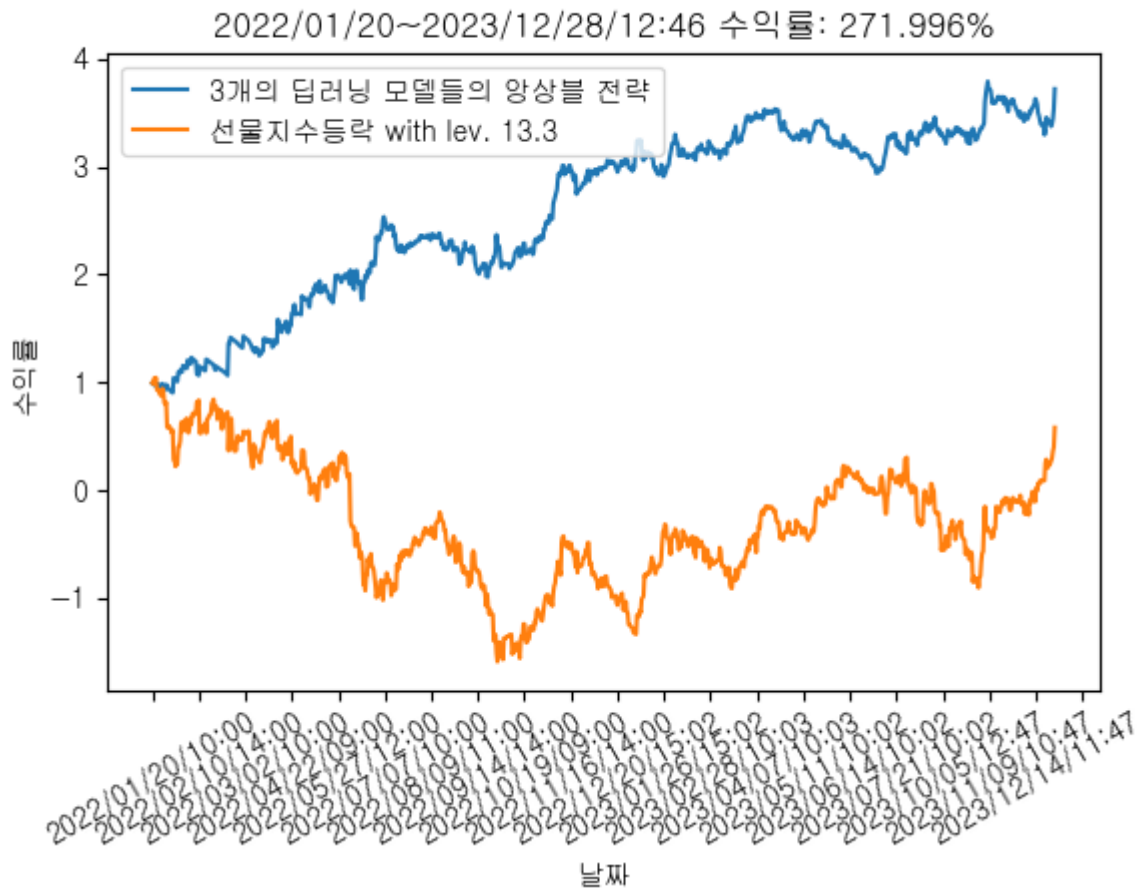
보는 바와 같이 알고리즘 1D-1 의 존재 이유는 없었다. 위의 표는 익일 시가 진입의 결과이나 당일 종가 진입의 결과는 더욱 더 참담했다.

결정했다.

알고리즘 1D-1 은 폐기!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!

- 2023 년 결산

  2022-01-01 ~ 2023-12-28 수익률 (증거금 대비, 증거금률 9.9%)



2023 년 요약: 이익 건수 119, 손실 건수 108, 승률 0.52422907

- 최적의 모델을 선택하는 방법.

  고려 대상인 모델 변수 :

  1. 앙상블 구성, 3 개의 개별 모델 TYPE (pred_term + type)
  2. 앙상블의 pred_term
  3. Self-reflection 기준값(reinfo_th)
  4. Evaluation term(reinfo_width)

  최근 1 년의 100 개의 랜덤 모델 중에서 product rate 기준 최상위 20 개 선택하여 그 이전 3 년간 수익률 조사 두 기간의 1 년 기준 기하 평균의 곱을 MDD 의 합으로 나눈 값이 최상인 모델 선택

  선택된 모델은 최근 학습 모델로서 실전에 적용. 수시로 Bayesian 을 이용하여 적용 앙상블의 reinfo_th, pred_term, reinfo_width 들의 최적의 값을 찾아 변경한다.

- Bayesian 활용 예

  알고리즘트레이딩 4-1 에서 2024-01-01 부터 선택된 모델은

  ensemble = ['15P', '20C', '25HL']

  pred_term = 5

  reinfo_th = 0.1

reinfo_width = 20

Bayesian 으로 01-01~01-03 의 기간 동안 최적의 변수를 search 해 보니..

pred_term = 35

reinfo_th = 0.389

reinfo_width = 75

였다.

2 개 모델을 01-04~01-05 이틀 동안 성과를 비교해 보니...

		original			bayesian 으로 변수 수정		
date	close	result	profit	fee	result	profit	fee
2024/01/04/09:00	352.65	1	0	0	1	0	0
2024/01/04/10:00	350.45	1	0	0	1	0	0
2024/01/04/11:00	349.5	1	0	0	1	0	0
2024/01/04/12:00	349.95	0	0	0	1	0	0
2024/01/04/13:00	350.25	0	0	0	1	0	0
2024/01/04/14:00	349.65	0	0	0	1	0	0
2024/01/04/15:00	350.3	0	587500	5272.125	1	587500	5272.125
2024/01/05/09:00	350.85	0	0	0	1	0	0
2024/01/05/10:00	349.8	1	0	0	1	0	0
2024/01/05/11:00	349.85	0	0	0	1	0	0
2024/01/05/12:00	350	2	-50000	5248.5	1	0	0
2024/01/05/13:00	348.95	1	-262500	5242.125	1	0	0
2024/01/05/14:00	349.2	1	0	0	1	0	0
2024/01/05/15:00	348.85	1	25000	5233.5	1	500000	5247.75

bayesian 으로 변수를 조정한 모델의 성과가 좋았다. 1 월 5 일

12 시 45 분에 원래 모델은 매수 시그널을 보냈는데 조정 모델은 매도 시그널을 보냄으로써 결과적으로 1 월 5 일 수익의 향상을 보였다.

기간별 수익률과 MDD	알고리즘 1D		알고리즘 1D-1	
	수익률	MDD	수익률	MDD
2023-01-01 ~ 2023-12-15	115%	62%	-26%	-46%
2021 ~ 2022	97%	170%	-41%	113%

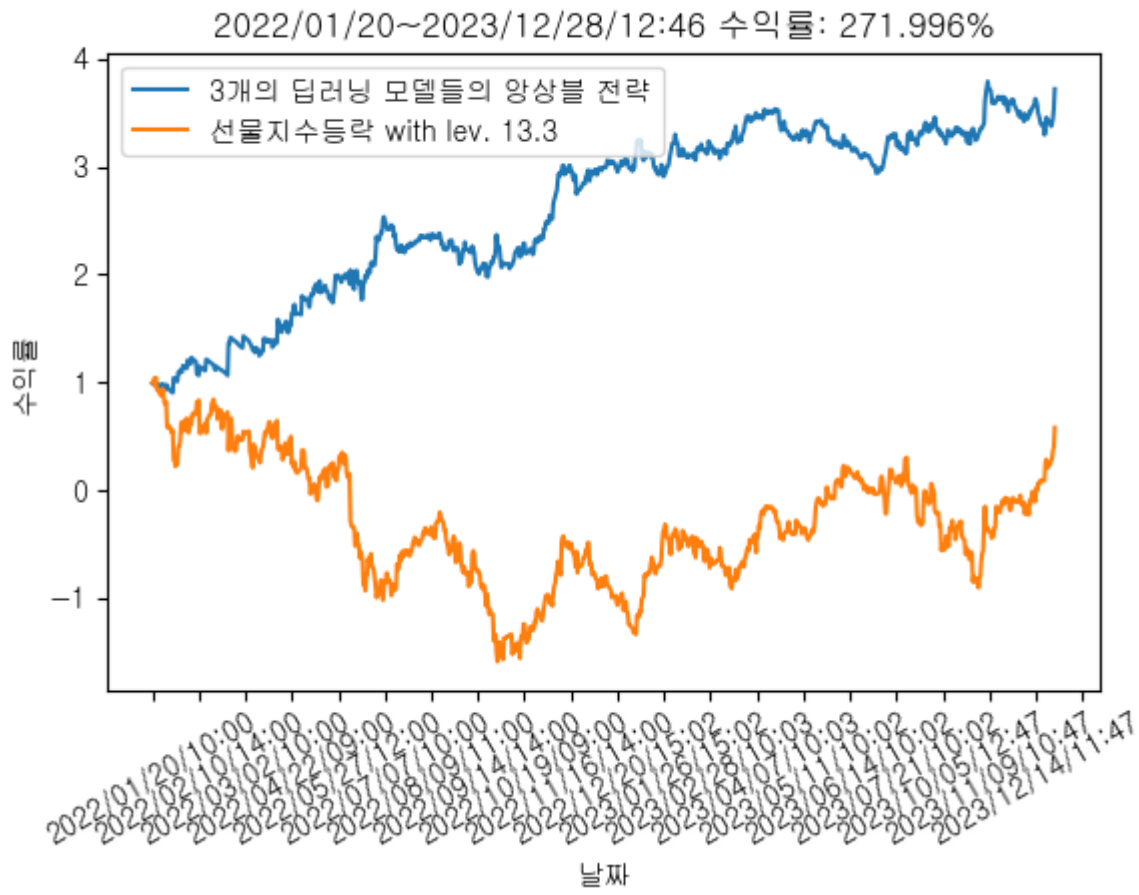
보는 바와 같이 알고리즘 1D-1 의 존재 이유는 없었다. 위의 표는 익일 시가 진입의 결과이나 당일 종가 진입의 결과는 더욱 더 참담했다.

결정했다.

알고리즘 1D-1 은 폐기!!

- 2023 년 결산

  2022-01-01 ~ 2023-12-28 수익률 (증거금 대비, 증거금률 9.9%)



2023 년 요약: 이익 건수 119, 손실 건수 108, 승률 0.52422907

- 최적의 모델을 선택하는 방법.

  고려 대상인 모델 변수 :

  1. 앙상블 구성, 3 개의 개별 모델 TYPE (pred_term + type)
  2. 앙상블의 pred_term
  3. Self-reflection 기준값(reinfo_th)
  4. Evaluation term(reinfo_width)

  최근 1 년의 100 개의 랜덤 모델 중에서 product rate 기준 최상위 20 개 선택하여 그 이전 3 년간 수익률 조사 두 기간의 1 년 기준 기하 평균의 곱을 MDD 의 합으로 나눈 값이 최상인 모델 선택

  선택된 모델은 최근 학습 모델로서 실전에 적용. 수시로 Bayesian 을 이용하여 적용 앙상블의 reinfo_th, pred_term, reinfo_width 들의 최적의 값을 찾아 변경한다.

- Bayesian 활용 예

  알고리즘트레이딩 4-1 에서 2024-01-01 부터 선택된 모델은

  ensemble = ['15P', '20C', '25HL']

  pred_term = 5

  reinfo_th = 0.1

reinfo_width = 20

Bayesian 으로 01-01~01-03 의 기간 동안 최적의 변수를 search 해 보니..

pred_term = 35

reinfo_th = 0.389

reinfo_width = 75

였다.

2 개 모델을 01-04~01-05 이틀 동안 성과를 비교해 보니...

		original			bayesian 으로 변수 수정		
date	close	result	profit	fee	result	profit	fee
2024/01/04/09:00	352.65	1	0	0	1	0	0
2024/01/04/10:00	350.45	1	0	0	1	0	0
2024/01/04/11:00	349.5	1	0	0	1	0	0
2024/01/04/12:00	349.95	0	0	0	1	0	0
2024/01/04/13:00	350.25	0	0	0	1	0	0
2024/01/04/14:00	349.65	0	0	0	1	0	0
2024/01/04/15:00	350.3	0	587500	5272.125	1	587500	5272.125
2024/01/05/09:00	350.85	0	0	0	1	0	0
2024/01/05/10:00	349.8	1	0	0	1	0	0
2024/01/05/11:00	349.85	0	0	0	1	0	0
2024/01/05/12:00	350	2	-50000	5248.5	1	0	0
2024/01/05/13:00	348.95	1	-262500	5242.125	1	0	0
2024/01/05/14:00	349.2	1	0	0	1	0	0
2024/01/05/15:00	348.85	1	25000	5233.5	1	500000	5247.75

bayesian 으로 변수를 조정한 모델의 성과가 좋았다. 1 월 5 일

12 시 45 분에 원래 모델은 매수 시그널을 보냈는데 조정 모델은 매도 시그널을 보냄으로써 결과적으로 1 월 5 일 수익의 향상을 보였다.