변동성 돌파전략 2

Contents

외부 데이터를 이용하여 윌리암스의 변동성 돌파전략을 개선해보겠습니다.

변동성 돌파전략은 시장이 좋을 때 활용하면 효과가 더 좋을 것 같습니다. 아무래도 상승장에서는 전일의 변동성을 돌파할 올라갈 확률이 높지 않을까요? 코스피 주가지수 데이터를 불러와서, 전일 코스피 주가지수의 수익율(종가 기준) 2%이상 발생한 경우만 매수를 하면 어떤 결과가 나오는지 테스트 해 보겠습니다. 매수일을 기준으로 2% 수익률이상으로 하면 더 좋을 것 같으나, 매수일의 종가 기준 수익율은 알 수 가 없기 때문에 전일을 기준으로 합니다. 먼저 지수데이터를 가져와서 종가 수익율을 계산합니다.

```
import FinanceDataReader as fdr
import pandas as pd

kospi_index = fdr.DataReader('KS11',  start='2021-01-03', end='2021-12-31')
kospi_index['kospi_return'] = kospi_index['Close']/kospi_index['Close'].shift(1)
kospi_index.to_pickle('kospi_index.pkl')
```

지수 데이터를 이용하면, 조건이 추가되므로 매수할 기회가 적어집니다. 지수데이터를 이용함으로써 예상수익율(일)이 0.3% 에서 1.6% 으로 올라갔습니다. 예상 최저수익율도 올라가서 리스크를 상당히 줄일 수 가 있습니다. 단, 매수 횟 수가 낮아 누적 수익율도 낮아졌습니다.

```
kospi_index = pd.read_pickle('kospi_index.pkl')
kospi_list = pd.read_pickle('kospi_list.pkl')


def avg_return(code, K, deci):

    stock = fdr.DataReader(code, start='2021-01-03', end='2021-12-31')
    stock_data = stock.merge(kospi_index['kospi_return'], left_index=True,
right_index=True, how='inner')
    stock_data['v'] = (stock_data['High'].shift(1) - stock_data['Low'].shift(1))*K
    stock_data['buy_price'] = stock_data['Open'] + stock_data['v']
    stock_data.dropna(inplace=True) # shift함수 이용으로 생긴 빈 셀 제거

    if deci == 1: # 지수 데이터를 이용한 경우
        stock_data['buy'] = (stock_data['High'] > stock_data['buy_price'])*
(stock_data['Low'] < stock_data['buy_price'])*(stock_data['kospi_return'].shift(1) >
1.02).astype(int)

    else: # 지수 데이터를 이용하지 않은 경우
        stock_data['buy'] = (stock_data['High'] > stock_data['buy_price'])*
(stock_data['Low'] < stock_data['buy_price']).astype(int)

    stock_data['return'] = stock_data['Open'].shift(-1)/stock_data['buy_price']

    n = len(stock_data[stock_data['buy']==1])
    r = stock_data[stock_data['buy']==1]['return'].mean()
    w = stock_data[stock_data['buy']==1]['return'].min()
    c = stock_data[stock_data['buy']==1]['return'].prod()
    return n, r, w, c


print('------------- 지수 데이터를 이용하지 않은 경우 ---------------')
symbol_list = []
name_list = []
obs_list = []
return_list = []
worst_list = []
cumul_list = []

for code, name in zip(kospi_list['code'],  kospi_list['name']):
    n, r, w, c = avg_return(code, 0.5, 0)

    if (r > 0) and (n > 0):  # 수익율 값이 존재하고, 최소한 한번 이상 거래일 존재해야 진
행
        symbol_list.append(code)
        name_list.append(name)
        obs_list.append(n)  # 매수 횟 수
        return_list.append(r)
        worst_list.append(w)
        cumul_list.append(c)

    else:
        continue

outcome_0 = pd.DataFrame({'symbol': symbol_list, 'name': name_list, 'num_obs':
obs_list, 'return': return_list, 'worst': worst_list, 'cumul': cumul_list})
print(outcome_0[['num_obs','return','worst','cumul']].describe())

print('\n')
print('------------- 지수 데이터를 이용한 경우 ---------------')
symbol_list = []
name_list = []
obs_list = []
return_list = []
worst_list = []
cumul_list = []

for code, name in zip(kospi_list['code'][:50],  kospi_list['name'][:50]):
    n, r, w, c = avg_return(code, 0.5, 1)

    if (r > 0) and (n > 0) :  # 수익율 값이 존재하고, 최소한 한번 이상 거래일 존재해야 진
행
        symbol_list.append(code)
        name_list.append(name)
        obs_list.append(n) # 매수 횟 수
        return_list.append(r)
        worst_list.append(w)
        cumul_list.append(c)

    else:
        continue

outcome_1 = pd.DataFrame({'symbol': symbol_list, 'name': name_list,  'num_obs':
obs_list, 'return': return_list, 'worst': worst_list, 'cumul': cumul_list})
print(outcome_1[['num_obs','return','worst','cumul']].describe())
```

```
------------- 지수 데이터를 이용하지 않은 경우 ---------------
          num_obs      return       worst       cumul
count  813.000000  813.000000  813.000000  813.000000
mean    92.398524    1.003090    0.901086    1.327235
std     15.477971    0.004127    0.143681    0.538553
min      2.000000    0.972844    0.000000    0.000000
25%     88.000000    1.000790    0.898420    1.043979
50%     94.000000    1.002838    0.931694    1.255082
75%    101.000000    1.005134    0.953390    1.514587
max    127.000000    1.017648    1.000000    5.318219


------------- 지수 데이터를 이용한 경우 ---------------
         num_obs     return      worst      cumul
count  49.000000  49.000000  49.000000  49.000000
mean    2.632653   1.016589   0.999325   1.040317
std     0.950743   0.019532   0.023266   0.044871
min     1.000000   0.975929   0.930901   0.927857
25%     2.000000   1.005605   0.984810   1.015540
50%     3.000000   1.011357   1.000000   1.029940
75%     3.000000   1.025377   1.010989   1.060224
max     4.000000   1.095283   1.066127   1.184608
```
