# 23.abcd-strategy-agent

September 29, 2021

[1]:
```
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
import seaborn as sns
sns.set()
```

[2]:
```
df = pd.read_csv('../dataset/GOOG-year.csv')
df.head()
```

[2]:

| | Date | Open | High | Low | Close | Adj Close | Volume |
|---|---|---|---|---|---|---|---|
| 0 | 2016-11-02 | 778.200012 | 781.650024 | 763.450012 | 768.700012 | 768.700012 | 1872400 |
| 1 | 2016-11-03 | 767.250000 | 769.950012 | 759.030029 | 762.130005 | 762.130005 | 1943200 |
| 2 | 2016-11-04 | 750.659973 | 770.359985 | 750.560974 | 762.020020 | 762.020020 | 2134800 |
| 3 | 2016-11-07 | 774.500000 | 785.190002 | 772.549988 | 782.520020 | 782.520020 | 1585100 |
| 4 | 2016-11-08 | 783.400024 | 795.632996 | 780.190002 | 790.510010 | 790.510010 | 1350800 |

[14]:
```
def abcd(trend, skip_loop = 4, ma = 7):
    ma = pd.Series(trend).rolling(ma).mean().values
    x = []
    for a in range(ma.shape[0]):
        for b in range(a, ma.shape[0], skip_loop):
            for c in range(b, ma.shape[0], skip_loop):
                for d in range(c, ma.shape[0], skip_loop):
                    if ma[b] > ma[a] and \
                    (ma[c] < ma[b] and ma[c] > ma[a]) \
                    and ma[d] > ma[b]:
                        x.append([a,b,c,d])
    x_np = np.array(x)
    ac = x_np[:,0].tolist() + x_np[:,2].tolist()
    bd = x_np[:,1].tolist() + x_np[:,3].tolist()
```

```
    ac_set = set(ac)
    bd_set = set(bd)
    signal = np.zeros(len(trend))
    buy = list(ac_set - bd_set)
    sell = list(list(bd_set - ac_set))
    signal[buy] = 1.0
    signal[sell] = -1.0
    return signal
```

[15]:
```
%%time
signal = abcd(df['Close'])
```

```
CPU times: user 1.08 s, sys: 8 ms, total: 1.09 s
Wall time: 1.09 s
```

[16]:
```
def buy_stock(
    real_movement,
    signal,
    initial_money = 10000,
    max_buy = 1,
    max_sell = 1,
):
    """
    real_movement = actual movement in the real world
    delay = how much interval you want to delay to change our decision from buy␣
 ↪to sell, vice versa
    initial_state = 1 is buy, 0 is sell
    initial_money = 10000, ignore what kind of currency
    max_buy = max quantity for share to buy
    max_sell = max quantity for share to sell
    """
    starting_money = initial_money
    states_sell = []
    states_buy = []
    states_money = []
    current_inventory = 0

    def buy(i, initial_money, current_inventory):
        shares = initial_money // real_movement[i]
        if shares < 1:
            print(
                'day %d: total balances %f, not enough money to buy a unit␣
 ↪price %f'
                % (i, initial_money, real_movement[i])
            )
        else:
            if shares > max_buy:
```

```
                buy_units = max_buy
            else:
                buy_units = shares
            initial_money -= buy_units * real_movement[i]
            current_inventory += buy_units
            print(
                'day %d: buy %d units at price %f, total balance %f'
                % (i, buy_units, buy_units * real_movement[i], initial_money)
            )
            states_buy.append(0)
        return initial_money, current_inventory

    for i in range(real_movement.shape[0]):
        state = signal[i]
        if state == 1:
            initial_money, current_inventory = buy(
                i, initial_money, current_inventory
            )
            states_buy.append(i)
        elif state == -1:
            if current_inventory == 0:
                    print('day %d: cannot sell anything, inventory 0' % (i))
            else:
                if current_inventory > max_sell:
                    sell_units = max_sell
                else:
                    sell_units = current_inventory
                current_inventory -= sell_units
                total_sell = sell_units * real_movement[i]
                initial_money += total_sell
                try:
                    invest = (
                        (real_movement[i] - real_movement[states_buy[-1]])
                        / real_movement[states_buy[-1]]
                    ) * 100
                except:
                    invest = 0
                print(
                    'day %d, sell %d units at price %f, investment %f %%, total␣
↪balance %f,'
                    % (i, sell_units, total_sell, invest, initial_money)
                )
            states_sell.append(i)
        states_money.append(initial_money)

    invest = ((initial_money - starting_money) / starting_money) * 100
    total_gains = initial_money - starting_money
```

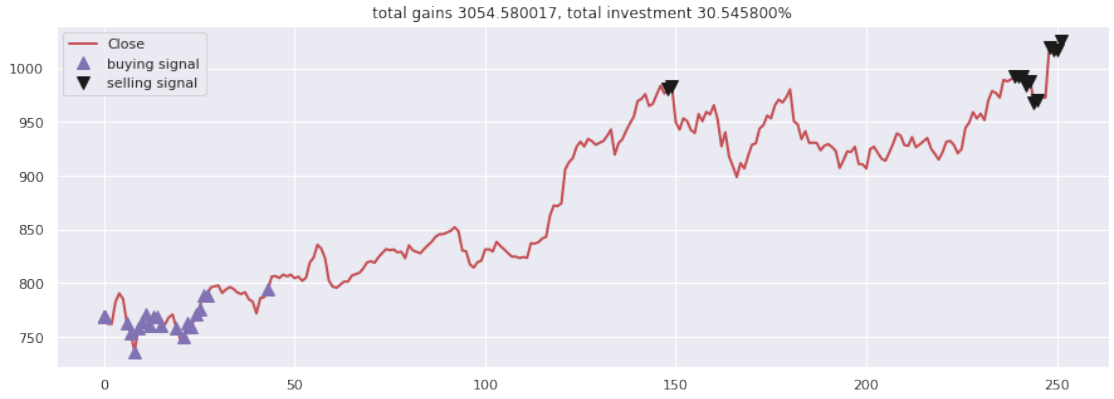
```
    return states_buy, states_sell, total_gains, invest, states_money
```

[17]:
```
states_buy, states_sell, total_gains, invest, states_money = buy_stock(df.
↪Close, signal)
```

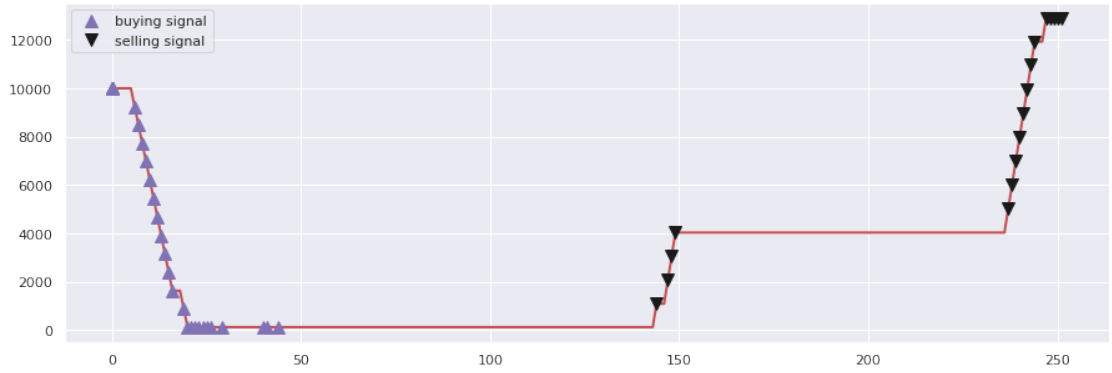
```
day 6: buy 1 units at price 762.559998, total balance 9237.440002
day 7: buy 1 units at price 754.020020, total balance 8483.419982
day 8: buy 1 units at price 736.080017, total balance 7747.339965
day 9: buy 1 units at price 758.489990, total balance 6988.849975
day 10: buy 1 units at price 764.479980, total balance 6224.369995
day 11: buy 1 units at price 771.229980, total balance 5453.140015
day 12: buy 1 units at price 760.539978, total balance 4692.600037
day 13: buy 1 units at price 769.200012, total balance 3923.400025
day 14: buy 1 units at price 768.270020, total balance 3155.130005
day 15: buy 1 units at price 760.989990, total balance 2394.140015
day 19: buy 1 units at price 758.039978, total balance 1636.100037
day 21: buy 1 units at price 750.500000, total balance 885.600037
day 22: buy 1 units at price 762.520020, total balance 123.080017
day 23: total balances 123.080017, not enough money to buy a unit price
759.109985
day 24: total balances 123.080017, not enough money to buy a unit price
771.190002
day 25: total balances 123.080017, not enough money to buy a unit price
776.419983
day 26: total balances 123.080017, not enough money to buy a unit price
789.289978
day 27: total balances 123.080017, not enough money to buy a unit price
789.270020
day 43: total balances 123.080017, not enough money to buy a unit price
794.020020
day 148, sell 1 units at price 980.940002, investment 23.540966 %, total balance
1104.020019,
day 149, sell 1 units at price 983.409973, investment 23.852038 %, total balance
2087.429992,
day 239, sell 1 units at price 992.000000, investment 24.933878 %, total balance
3079.429992,
day 240, sell 1 units at price 992.179993, investment 24.956546 %, total balance
4071.609985,
day 241, sell 1 units at price 992.809998, investment 25.035890 %, total balance
5064.419983,
day 242, sell 1 units at price 984.450012, investment 23.983021 %, total balance
6048.869995,
day 243, sell 1 units at price 988.200012, investment 24.455302 %, total balance
7037.070007,
day 244, sell 1 units at price 968.450012, investment 21.967959 %, total balance
8005.520019,
day 245, sell 1 units at price 970.539978, investment 22.231172 %, total balance
```

```
8976.059997,
day 248, sell 1 units at price 1019.270020, investment 28.368302 %, total
balance 9995.330017,
day 249, sell 1 units at price 1017.109985, investment 28.096264 %, total
balance 11012.440002,
day 250, sell 1 units at price 1016.640015, investment 28.037076 %, total
balance 12029.080017,
day 251, sell 1 units at price 1025.500000, investment 29.152915 %, total
balance 13054.580017,
```

[18]:
```
close = df['Close']
fig = plt.figure(figsize = (15,5))
plt.plot(close, color='r', lw=2.)
plt.plot(close, '^', markersize=10, color='m', label = 'buying signal',␣
 ↪markevery = states_buy)
plt.plot(close, 'v', markersize=10, color='k', label = 'selling signal',␣
 ↪markevery = states_sell)
plt.title('total gains %f, total investment %f%%'%(total_gains, invest))
plt.legend()
plt.show()
```




[8]:
```
fig = plt.figure(figsize = (15,5))
plt.plot(states_money, color='r', lw=2.)
plt.plot(states_money, '^', markersize=10, color='m', label = 'buying signal',␣
 ↪markevery = states_buy)
plt.plot(states_money, 'v', markersize=10, color='k', label = 'selling signal',␣
 ↪markevery = states_sell)
plt.legend()
plt.show()
```


buying signal
selling signal
12000
10000
8000
6000
4000
2000
0
0
50
100
150
200
250

[ ]: