Machine Learning Engineer Nanodegree

Lin Yu, Maria

May 23^{rd}, 2019

1 Definition

1.1 Project Overview

Stock investment can be one of the ways to manage one's asset. Technical analysis is sometimes used in financial markets to assist traders make buying and selling decisions [1]. Many technical analysis trading rules are deterministic trading policies. [2] uses genetic algorithm to find technical trading rule. [3] studies evolutionary algorithms in optimization of technical rules for automated stock trading. [4] proposed a stock trading system based on optimized technical analysis parameters for creating buy-sell points using genetic algorithms. [5] studies the selection of the optimal trading model for stock investment in different industries. [6] describes the optimization of trading strategies.

The optimization of trading rule using genetic algorithm or evolutionary algorithms belongs to policy-based method, which is a branch of Machine Learning. Policy-based methods try to directly optimize for the optimal policy which is an important branch for domains with continuous action spaces [7]. There are studies focus on how to find a trading strategy via Reinforcement Learning (RL) [8] or using Deep-Q learning for automatic trading algorithm [9]. But in this study we will focus on the policy-based method using Generic Algorithm that directly search for the optimal parameters of a deterministic policy.

Yahoo Finance's stock history data [10] will be used in this study. The reason to choose Yahoo Finance data is because it is free and available for public to assess. The performance of algorithm will be evaluated using different stocks.

The purpose of the study is to see the difference between using an agent with optimized policy to manage one's asset with buy-and-hold strategy, or manage one's asset with an agent with unoptimized policy.

1.2 Problem Statement

The goal is to create a usable tool with iPython Notebook on Macbook Pro; the tasks involved are the following:

1. Download and preprocess the Yahoo Finance data.
2. Choose and train an agent that is able to manage the trading such that it maximize the initial investment.
    - Choose a trading policy
    - Choose a trading agent
    - Define how to split training and testing data. Here in this study, I use past one year data (April 2017- April 2018) for training and the current one year data (April 2018 - April 2019) for testing. I will repeat this study for using April 2014-April 2015 data as training

data and April 2015- April 2016 as testing and so on for 4 years. I will also repeat the study for another stock for 4 years.

- Optimize the policy hyper parameters using the training data
- Use the trading agent with the trading policy with optimized hyper parameter to conduct trading for the testing data.

3. Compare the Agent's performance on trading with the optimized hyper parameters for the select stock (AAPL) with respect to the buy-and-hold strategy and see whether optimizing hyper parameter of a trading policy is able to help achieving better gain for the testing data.

The intention of the solution is to create a tool to help one make decision on managing his/her own asset and to show that with optimized hyper parameters, the trading agent use the same trading policy, it may be able to perform better than buy-and-hold.

1.3 Metrics

1.3.1 Return of Investment

A very simple investment return is defined similar to interest that banks give per year. The investment return in this project is defined as gain in percentage with respect to his/her initial investment.

$$Return\ of\ Investment = \frac{Assets\ Value}{Initial\ Investment} \tag{1}$$

1.3.2 Sharpe Ratio

Definition of the Sharpe Ratio is given by [11]. The Sharpe ratio was developed by Nobel laureate William F. Sharpe and is used to help investors understand the return of an investment compared to its risk. The ratio is the average return earned in excess of the risk-free rate per unit of volatility or total risk.

Subtracting the risk-free rate from the mean return allows an investor to better isolate the profits associated with risk-taking activities. Generally, the greater the value of the Sharpe ratio, the more attractive the risk-adjusted return. Sharpe Ratio is defined as:

$$Sharpe\ Ratio = \frac{R_p - R_f}{\sigma_p} \tag{2}$$

where, R_p is the return of portfolio, R_f is risk-free rate, σ_p is standard deviation of the portfolio. Here we use $R_f = 2.5\%$ as risk-free rate through out this study as this is the highest risk free deposit rate in Singapore. We didn't use the benchmark as the risk free rate as most common people don't regard stock. buy-and-hold strategy as risk-free.

1.3.3 Sortino Ratio

The Sortino ratio was designed to replace Sharpe ratio as a measure of risk-adjusted return [12]. It is defined as:

$$Sortino\ Ratio = \frac{R_p - R_f}{\sigma_d} \tag{3}$$

where, R_p is the return of portfolio, R_f is risk-free rate, σ_d is standard deviation of the down side of the portfolio.

In this study, I will use Return of Investment (ROI) as the cost function to optimize the trading policy parameter and we will evaluate the result with both Sharpe Ratio and Sortino Ratio. The

reason for choosing purely ROI as the cost function this is that if ROI is optimized with a buying low and selling high strategy, then the Sharpe Ratio and Sortino Ratio shall be automatically taken care of. We will evaluate both metrics to verify this hypothesis.

When I choose the trading policy and trading agent, I did consider the processing time, where the processing time includes the time for the policy to give a suggestion and the trading agent to give ROI. This is because if it takes very long time for the trading agent to give a ROI for each buying for selling step, then the optimization process will be very long as optimization always use the performance metric ROI as a guidance to optimize the hyper parameter of a trading policy.

The time for a policy to give a suggestion is also critical as as asset management can be a time consuming task if the program takes very long time to give a suggestion. This may cause user to lose patient on using it.

2 Analysis

2.1 Data Exploration

I decided to get data from Yahoo Finance for stock prices. It provides historical data that can be downloaded as csv files. I choose AAPL as the first example to study as this is one of the most popular stock that has large transaction volume.

The historical data fetch from Yahoo Finance contains, **Date**, **Open**, **High**, **Low**, **Close**, **Adjusted Close** and **Volume**. The difference between Adjusted Close and Close are described in [14]. The reason that the **Adjusted Close** is chosen is because it accounts for all corporate actions such as stock splits, dividends/distributions and rights offerings.

Below compares AAPL with GOOG, the stock used in [18]. Fig. 1 shows a snap shot of AAPL and GOOG data from Yahoo Finance. Fig. 2 shows the chart of AAPL and GOOG **Adj Close** Price for the past one year. It can be observed from the chart that the two stocks might be highly correlated. Fig. 3 shows the data statistics of AAPL and GOOG.

It can be observed from Fig. 3 that there are 253 total samples for all columns of both GOOG and AAPL data.

2.2 Exploratory Visualization

Since both GOOG and AAPL are components of SPY Index [16], to understand the data correlation better, we downloaded two more data from Yahoo Finance, SPY and GLD. We may expect that both GOOG and AAPL are positively correlated with SPY but may not have such relationship with GLD. Fig. 4 shows the SPY and GLD chart for latest one year data. Fig. 5 are the scatter plots that show the correlation between GOOG, AAPL with SPY and correlation between AAPL and GOOG and AAPL and GLD.

Date	High	Low	Open	Close	Volume	Adj Close
2018-04-23	166.919998	164.089996	166.830002	165.240005	36515500.0	162.761597
2018-04-24	166.330002	161.220001	165.669998	162.940002	33692000.0	160.496109
2018-04-25	165.419998	162.410004	162.619995	163.649994	28382100.0	161.195450
2018-04-26	165.729996	163.369995	164.119995	164.220001	27963000.0	161.756912
2018-04-27	164.330002	160.630005	164.000000	162.320007	35655800.0	159.885406

Date	High	Low	Open	Close	Volume	Adj Close
2018-04-23	1082.719971	1060.699951	1077.859985	1067.449951	2341300	1067.449951
2018-04-24	1057.000000	1010.590027	1052.000000	1019.979980	4760300	1019.979980
2018-04-25	1032.489990	1015.309998	1025.520020	1021.179993	2391100	1021.179993
2018-04-26	1047.979980	1018.190002	1029.510010	1040.040039	2079500	1040.040039
2018-04-27	1049.500000	1025.589966	1046.000000	1030.050049	1619800	1030.050049

Figure 1: A snap shot of raw data. Left: AAPL, Right: GOOG

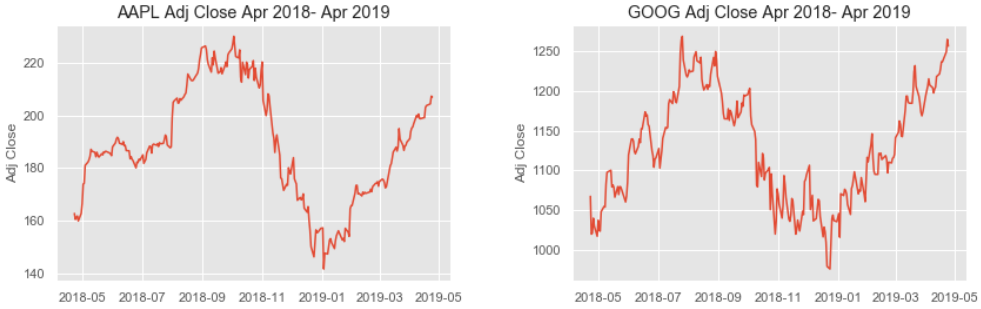

Figure 2: Adj Close Price chart for data between April 2018 and April 2019. . Left: AAPL, Right: GOOG

	High	Low	Open	Close	Volume	Adj Close
count	253.000000	253.000000	253.000000	253.000000	2.530000e+02	253.000000
mean	191.921819	188.207786	190.050040	190.163320	3.212594e+07	188.892235
std	21.728724	21.454309	21.567157	21.575702	1.419415e+07	21.272061
min	145.720001	142.000000	143.979996	142.190002	1.251390e+07	141.582779
25%	175.000000	172.350006	173.869995	174.240005	2.252550e+07	173.495911
50%	190.160004	187.529999	189.009995	188.720001	2.838210e+07	187.782913
75%	209.250000	206.089996	207.360001	207.529999	3.835890e+07	206.540436
max	233.470001	229.779999	230.779999	232.070007	9.624670e+07	230.275482

	High	Low	Open	Close	Volume	Adj Close
count	253.000000	253.000000	253.000000	253.000000	2.530000e+02	253.000000
mean	1140.314657	1117.528862	1128.812510	1129.299879	1.588594e+06	1129.299879
std	65.982735	70.334942	68.649881	68.736806	6.484036e+05	68.736806
min	1003.539978	970.109985	973.900024	976.219971	6.790000e+05	976.219971
25%	1083.974976	1060.680054	1072.939941	1071.469971	1.199300e+06	1071.469971
50%	1132.170044	1111.010010	1123.140015	1121.280029	1.443200e+06	1121.280029
75%	1197.880005	1181.000000	1189.390015	1186.959961	1.848700e+06	1186.959961
max	1273.890015	1255.000000	1271.000000	1268.329956	4.760300e+06	1268.329956

Figure 3: Data statistics. Left: AAPL, Right: GOOG

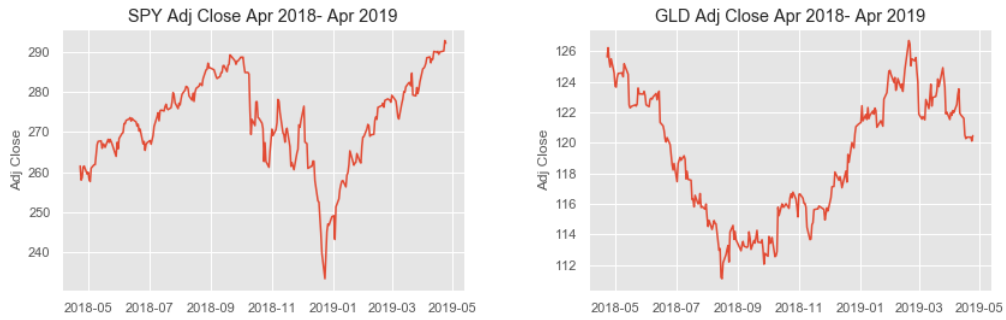

Figure 4: Adj Close Price chart for data between April 2018 and April 2019. Left: SPY, Right: GLD

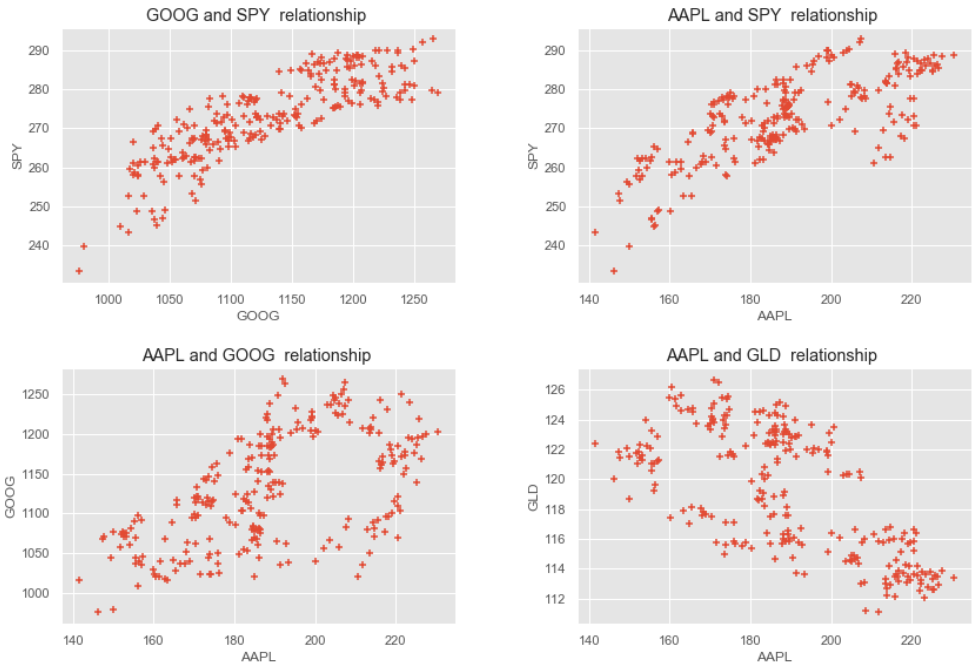

Figure 5: Data correlation: GOOG vs SPY, AAPL vs SPY, AAPL vs GOOG and AAPL vs GLD

The above analysis shows that if one would like to diversify his/her investment risk, he/she may choose GLD as another investment tool which is not highly correlated to AAPL.

Since in this study, I will only focus trading using one stock as an example, and the policy I chose is only use the **Adj Close** column. Let's take a look at the correlations between its own columns. Finally, 6 shows the correlation between AAPL dataset's columns. It can be observed that the **Open**, **Close**, **High** and **Low** are highly correlated with **Adj Close** while **Volume** is not really very correlated with the **Adj Close** column. This means if we choose a policy that uses any price related information, they gives similar information, while a policy using volume information may perform very differently as the information in volume and in price are not very correlated.

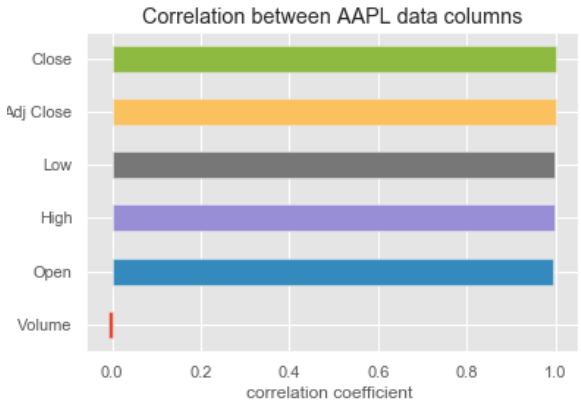

Figure 6: Data correlation: AAPL columns with respect to the Adj Close Price.

2.3 Algorithms and Techniques

2.3.1 Problem Formulation

We model the problem as an investment episode. Each episode last for one year. The trading agent is given an initial amount of cash to invest. At the end of the episode the agent may hold cash or shares or both. Figure 7 shows the state space and action space. The state space include the cash holding, the number of share and the asset value (cash plus share value). The action space includes three possible actions: buy x number of shares, sell x number of shares and do nothing.

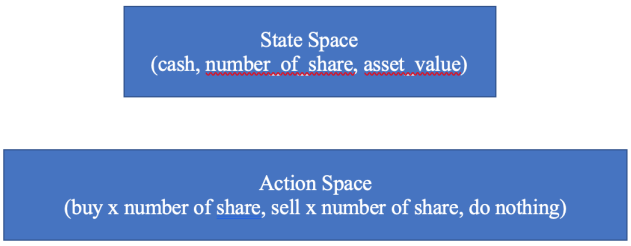

Figure 7: The state space and action space.

2.3.2 Objective Function

The objective function can be a state value, can be average state value, average action value or a function of reward. The best true objective function is a function of reward.

$$J(\theta) = E_\pi[R(\tau)] \tag{4}$$

Here, $\tau = S_0, A_0, R_1, S_1 ....$ in which S is state, A is action, and R is reward and π, the trading policy $\pi(S, A, \theta)$, is a function of state, action and policy parameters, θ. Therefore, J is a function of θ.

2.4 Policy and Optimization Pseudo Code

Here in this study we focus on deterministic policy:

$$\pi : s \rightarrow a \tag{5}$$

The pseudo code for the optimization of the the $J(\theta)$ is given below:

Algorithm

Generate initial population of θs arbitrarily

for the episode for each θ in the generation:

evaluate $J(\theta)$

θ = best θ of the generation

While $(iteration < max\ iteration)$

Breed a new generation of θs from the best θ

for each θ: evaluate $J(\theta)$

Select the best θ that gives the best $J(\theta)$

Usually the termination of the optimization can be controlled by maximum iteration or when $\Delta J(\theta) < threshold$. Here we choose maximum iteration to better control the simulation time.

2.5 Genetic Algorithm

I chose Generic Algorithm (GA) to optimize the policy parameters. FIG. 8 illustrates the general steps taken in executing a GA [17].

A GA is a metaheuristic inspired by the process of natural selection that belongs to the larger class of evolutionary algorithms (EA). Genetic algorithms are commonly used to generate high-quality solutions to optimization and search problems by relying on bio-inspired operators such as mutation, crossover and selection. [19] John Holland introduced genetic algorithms in 1960 based on the concept of Darwin's theory of evolution; afterwards, his student David E. Goldberg extended GA in 1989 [20].

In a genetic algorithm, a population of candidate solutions (called individuals, creatures, or phenotypes) to an optimization problem is evolved toward better solutions. Each candidate solution has a set of properties (its chromosomes or genotype) which can be mutated and altered; traditionally, solutions are represented in binary as strings of 0s and 1s, but other encodings are also possible [21].

The evolution usually starts from a population of randomly generated individuals, and is an iterative process, with the population in each iteration called a generation. In each generation, the fitness of every individual in the population is evaluated; the fitness is usually the value of the objective function in the optimization problem being solved. The more fit individuals are stochastically selected from the current population, and each individual's genome is modified (recombined and possibly randomly mutated) to form a new generation. The new generation of candidate solutions is then used in the next iteration of the algorithm. Commonly, the algorithm terminates when either a maximum number of generations has been produced, or a satisfactory fitness level has been reached for the population.

Fig. 8 shows the GA diagram. It includes the following steps:

1. Input data is read
2. an initial population of M genotypes is generated.
3. A fitness function is applied to determine the fitness of each genotype..

4. A new generation of genotypes is bred by using reproduction, crossover and mutation.
5. The fitness of each genotype is again evaluated.
6. The M most fit genotypes are selected from the new generation.
7. A determination is made as to whether the best genotype resulting from the previous Selection is Satisfactory.
8. If the best genotype is not satisfactory, Steps 4 to 7 are repeated.
9. If the best genotype is satisfactory, the best genotype is displayed.

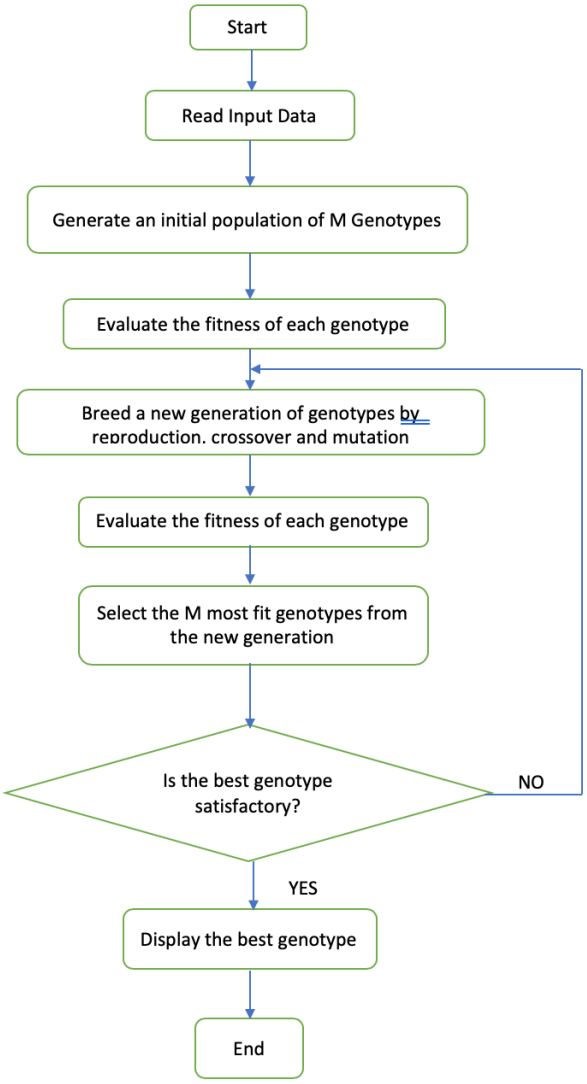

Figure 8: GA diagram.

A typical genetic algorithm requires:

1. a genetic representation of the solution domain,
2. a fitness function to evaluate the solution domain.

2.6 Benchmark

I choose the investment return with buy-and-hold strategy as benchmark strategy. Investment return for buy-and-hold will calculated as buying the maximum number of share with the initial investment of $10,000$ at the price of the beginning ten day's average price and selling the number of share bought at the price of the ending ten day's average price. The ROI, Sharpe Ratio, Sortino Ratio of the benchmark buy-and-hold strategy is tabulated below for training data and testing data.

Training data: Benchmark buy-and hold strategy performance

	ROI	Sharpe Ratio	Sortino Ratio
Buy and Hold	20.69%	6.68%	5.97%

Testing data: Benchmark buy-and hold strategy performance

	ROI	Sharpe Ratio	Sortino Ratio
Buy and Hold	20.96%	5.87%	7.78%

3 Methodology

3.1 Data Preprocessing

The preprocessing of the data is pretty simple for the study. It is mainly load the AAPL data from Yahoo Finance and get the previous year data as training data and get the next one year data as testing data, drops the not suitable data with dropna() function. It can be observed from the data statistics in Fig. 3 that the min, max values all are in reasonable range of values. The training and testing data chart in Fig 9 also looks reasonable. All data seems to be valid and no outliers need to be replaced.

```
# get last 1 year data from Yahoo Finance for training
start = dt.datetime(2017,4,24)
end = dt.datetime(2018, 4,24)
df = web.DataReader('AAPL','yahoo', start,end)
X_train = df
```

```
# get current 1 year data from Yahoo Finance for testing
start = dt.datetime(2018,4,24)
end = dt.datetime(2019, 4,24)
df = web.DataReader('AAPL','yahoo', start,end)
X_test = df
```

The logic behind splitting training and testing data is that we can only use past data for training as in practical situation no body knows the future stock price. Sine we can only using past data for training, in order to test the performance with unseen data, we reserved the latest one year data from April 2018 - April 2019 for testing. Therefore we can only use say the most recent data besides the reserved testing data, which would be April 2017 to April 2018. Here we consider investment

period for training and testing are the same, 1 year. Fig.9 shows the training data and the testing data for AAPL.

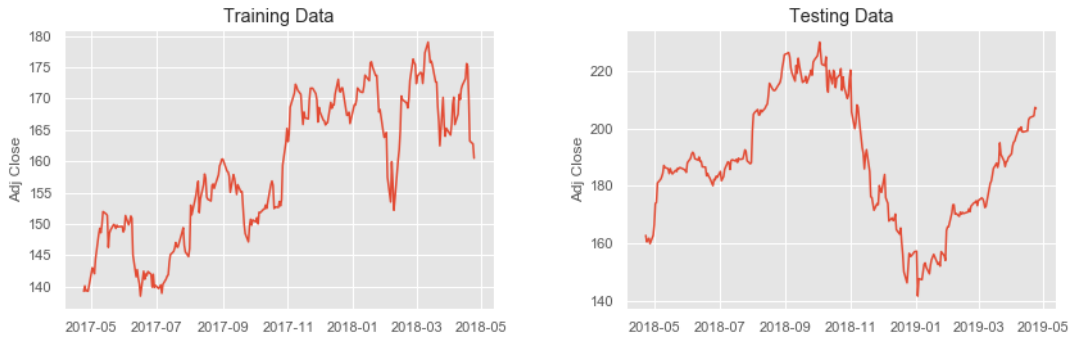

Figure 9: Training and testing data in this study.

Fig. 10 shows the training data and testing data information. Both training data and testing data consists of 253 non-null float data. There is no missing data exist in the training and testing data.

```
X_test.info()
```

```
<class 'pandas.core.frame.DataFrame'>
DatetimeIndex: 253 entries, 2018-04-23 to 2019-04-24
Data columns (total 6 columns):
High          253 non-null float64
Low           253 non-null float64
Open          253 non-null float64
Close         253 non-null float64
Volume        253 non-null float64
Adj Close     253 non-null float64
dtypes: float64(6)
memory usage: 13.8 KB
```

```
X_train.info()
```

```
<class 'pandas.core.frame.DataFrame'>
DatetimeIndex: 253 entries, 2017-04-24 to 2018-04-24
Data columns (total 6 columns):
High          253 non-null float64
Low           253 non-null float64
Open          253 non-null float64
Close         253 non-null float64
Volume        253 non-null float64
Adj Close     253 non-null float64
dtypes: float64(6)
memory usage: 13.8 KB
```

Figure 10: Training and testing data information.

Since we are not going to use any neural network which requires normalization or scaling due to the activation function that saturate input and limit the maximum input that pass through the activation function to be the maximum value of the activation function, we are not going to do any scaling or normalization in this study. As any scaling will requires us un-do the scaling if we would like to see the chart in origin price range.

3.2 Implementation

[18] Huseinzol summarized 23 trading policies in his maxGitHub repository. Among all trading policies, I chose abcd strategy which gives relatively good ROI and short processing time. It only uses **Adj Close** Price to generate buying and selling trading signals. The trading policy has two parameters, one is moving average period (ma) and another is skip_loop: the number of trading days to skip when considering the critical a, b,, c, d reference points on moving average curve. This is a typical deterministic policy where policy $\pi s \rightarrow a$ determines buy sell or do nothing solely based on states. I will explain this in detailed in the Implementation Section.

I chose this abcd trading policy is because it has good return for the default GOOG data with its default setting. It's processing time is also short which means that the optimization of it hyper parameters doesn't need a. lot of computation power. It seems to me that the policy is a buy low and sell high type of policy. The policy and the agent have a few hyper parameters only. Therefore, the optimization of its hyper parameters won't be very complex.

The abcd Trading Policy

The policy of trading gives buying and selling signals based on a predefined policy [18]. The abcd trading policy detects a pattern in the moving average (ma) of **Adj Close** price. The moving average window size is parameter *ma*.

In this policy, there are four monitoring points(trading day), a, b, c, and d, spaced at minimum *skip_loop* trading days apart from each other. Monitoring point a is a beginning monitoring point of the moving average of Adj Close price and b is monitoring point after a with a higher moving average value then a, i.e. ma[b]¿ ma[a]. and c is a monitoring point after b with a ma value between $ma[a]$ and $ma[b]$, i.e., ma[a]¡ma[c]¡ma[b]. d is a monitoring point after c and have a moving average value higher than b, i.e. ma[d]¿ ma[b].

The trading signal such as buy and sell is issued based on the patterns found in the moving average of the Adj Close price. The trading policy scan though all the trading days in the investment period, and create two sets: ac_set and bd_set.

If a trading day is in ac_set and not in bd_set, then a buy signal is issued.

If a trading day is in bd_set and not in ac_set, then a sell signal is issued.

The implementation of this abcd trading policy is given by [18]. The python implementation is illustrated below.

```
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
```

Fig. 11 shows a snap shot of the detected patten array x_np of the policy and Fig. 12 shows the first detected abcd patten. It can be seen that a and c are has lower ma values than b and d. The policy tried to detect the turning point so that the agent can buy low and sell high.

```
x_np
```

```
array([[  2,   6,  10,  18],
       [  2,   6,  10,  22],
       [  2,   6,  10,  26],
       ...,
       [144, 160, 172, 176],
       [144, 164, 172, 176],
       [147, 151, 159, 175]])
```

Figure 11: abcd patten on ma['Adj Close'] of AAPL.

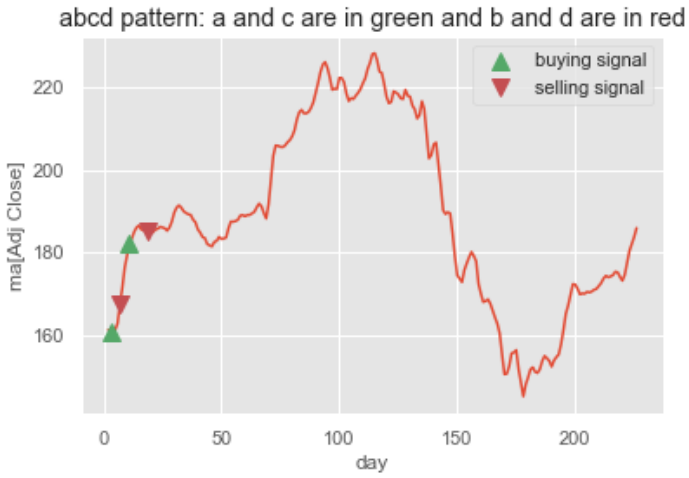

Figure 12: abcd patten on ma['Adj Close'] of AAPL.

The Trading Agent

A trading agent uses the above trading policy to buy and sell share. The trading agent is a function that records states_buy, states_sell, total_gain, investment ROI, current cash and asset value. The input is the Adj Close price, the policy output (buying and selling signal), the maximum number of shares that the agent is allowed to buy and sell for each trading day.

```
states_buy, states_sell, total_gains, invest, states_money, states_asset =
    buy_stock(df.Close, signal, max_buy,max_sell)
```

As we can see that the trading policy and the trading agent consists of the following hyper parameters:

- skip_loop: the number of sample to be skipped before next buy and sell action.
- ma: moving average period
- max_buy: the maximum share to be purchased in a single buy action

- max_sell: the maximum share to be sold in a single sell action

3.3 Refinement

3.3.1 Generic Algorithm Implementation

In this genetic algorithm implementation, a genetic representation of optimization of the hyper-parameter is to take the policy and trading parameters as the genotype to be optimized. Here we generate the next generation of hyper parameters by generating parameter values close to the best set of hyper-parameters of previous generation. This represents minor alternation of genotype over generation from the best among the previous generation. A fitness function here is the investment return generated by the buying_stock agent for evaluating the fitness of the policy hyper-parameters.

In the present implementation, the inputs for the GA include the following policy parameters: param =[skip_loop, ma, max_buy and max_sell]. To further simplify the optimization, in this study, I force max_buy = max_sell.

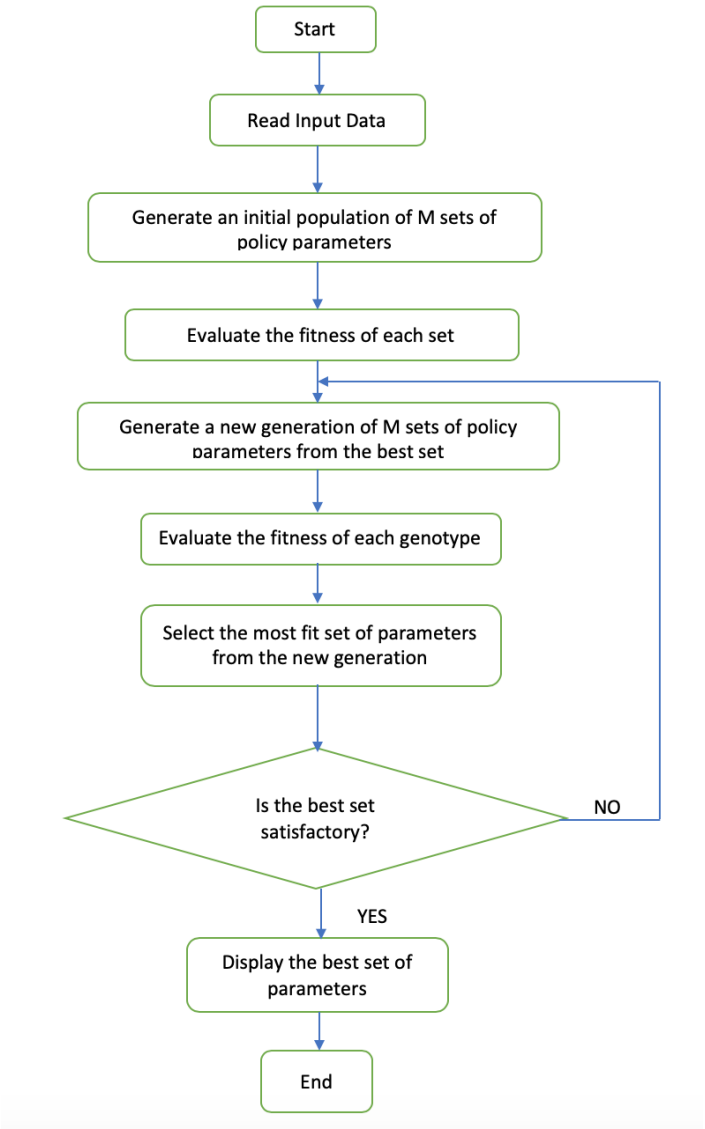

Figure 13: GA implementation in this study.

Fig 13 illustrates the GA implementation of this study. The input data here is the Adj Close price of the stock selected, which is AAPL here. The initial population of the M sets of policy parameters are arbitrarily selected which shouldn't be very critical. The fitness of each set of hyper parameters are evaluated using the buying_stock agent's ROI. Among the first generation of parameter sets, the best set of parameter that gives the best ROI is selected and a new generation of M sets of policy parameters is generated by altering the parameters around its neighbourhoods by a random gap. The code for generating the next generation of parameters and the GA implementation are listed below.

```
def gen_list(best_param):
    skip_loop, ma, max_buy, max_sell = best_param
    gap = randint(1,10)
    if (skip_loop -gap)>0 and ma-gap> 0 and max_buy-5>0:
```

```
        skip_loop_list = [skip_loop-gap, skip_loop, skip_loop+gap]
        ma_list =[ma-gap, ma, ma+gap]
        max_list = [max_buy-5*gap, max_buy, max_buy+5*gap]
    else:
        skip_loop_list = [skip_loop, skip_loop+gap, skip_loop+2*gap ]
        ma_list =[ma, ma+gap, ma+2*gap]
        max_list = [max_buy, max_buy+5*gap, max_buy+10*gap]
    return skip_loop_list, ma_list, max_list, max_list
\label{listing:gen_list}
```

```
best_J =[]
i=0
J=[]
param_vec = []

for iter in np.arange(1):
    if iter == 0:
        skip_loop_list = [3, 5, 7]
        ma_list = [3, 7, 13]
        max_list = [10, 50, 100]
        best_param =[]
        for skip_loop in skip_loop_list:
            for ma in ma_list:
                for max_buy, max_sell in zip (max_list, max_list):
                    signal = abcd(df_last_1yr['Adj Close'], skip_loop = skip_loop, ma = ma)
                    states_buy, states_sell, total_gains, invest, states_money, states_asset
                        = buy_stock(df_last_1yr['Adj Close'], signal,max_buy, max_sell)
                    J.append(invest)
                    param =[skip_loop, ma, max_buy, max_sell]
                    param_vec.append(param)
                    best_J.append(max(J))
                    i=i+1
        maxpos = J.index(max(J))
        best_param = param_vec[maxpos]
    else:
        skip_loop, ma, max_buy, max_sell = gen_list(best_param)
        for skip_loop in skip_loop_list:
            for ma in ma_list:
                for max_buy, max_sell in zip (max_list, max_list):
                    signal = abcd(df_last_1yr['Adj Close'], skip_loop = skip_loop, ma = ma)
                    states_buy, states_sell, total_gains, invest, states_money, states_asset
                        = buy_stock(df_last_1yr['Adj Close'], signal,max_buy, max_sell)
                    J.append(invest)
                    param =[skip_loop, ma, max_buy, max_sell]
                    param_vec.append(param)
                    best_J.append(max(J))
                    i=i+1
        maxpos = J.index(max(J))
        best_param = param_vec[maxpos]
```

4 Results

4.1 Model Evaluation and Validation

Fig. 14 shows the best ROI versus iterations. It can be observed that the best ROI improves as the algorithm converges.

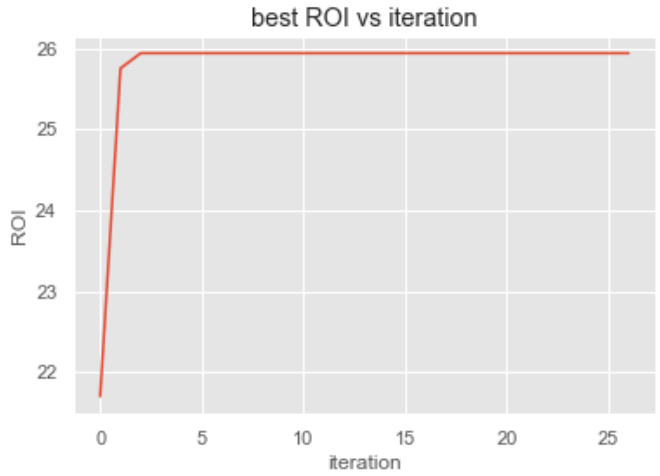

Figure 14: Best ROI vs iterations.

The best set of Hyper parameters is found to be ma=3, skip_loop=3, max_buy = max_sell =100. The best ROI for the training data

4.1.1 Training Performance

Fig.15 shows the training performance of the trading agent with optimized Hyper parameters for the policy and trading agent. The optimized setting is ma=3, skip_loop=3, and max_buy=max_sell =100.

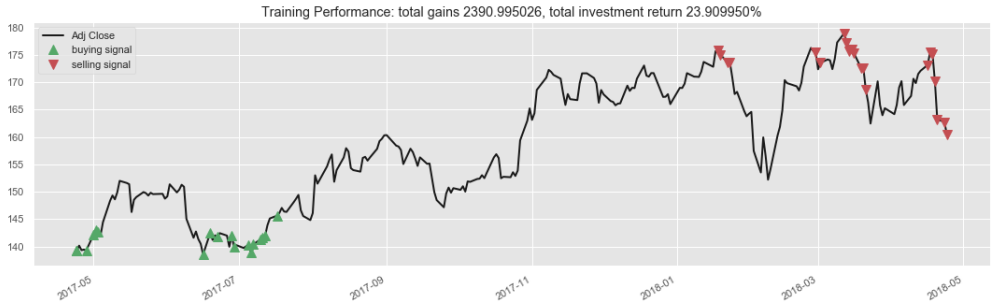

Figure 15: Adj Close Price chart for data between April 2017 and April 2018 with trading signals and investment return with optimized setting.

4.1.2 Testing Performance

Fig. 16 shows the testing performance of the trading agent with optimized Hyper parameters for the policy and trading agent.

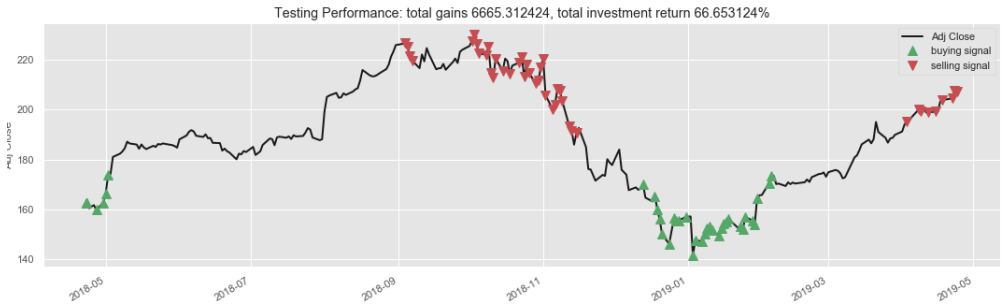

Figure 16: Adj Close Price chart for data between April 2018 and April 2019 with trading signals and investment return with optimized setting.

4.2 Justification

4.2.1 Compare the Training and Testing Performance with Benchmark

Table 1a and 1b compare the training and testing performance of optimized hyper parameters trading policy with the default setting, respectively. It can be observed that the ROI is best with the optimized set of hyper parameters for both the training data and the testing data. We have also evaluated the Sharpe Ratio and Sortino Ratio for users references. It can be observed from the results that the assumption that if we optimize ROI, then Sharpe Ratio and Sortino ratio will automatically be taken care is not true. The Sortino ratio of the default setting performs better than the optimized setting. This shows that the three metrics are different for different investors. Sharpe Ratio is good for investors to choose investments with reduced risk/variations. Sortino ratio provides a performance measure for downside risk. ROI is purely based on return. If one is keen on maximizing his/her investment return, ROI is the best performance index.

Table 1a. Compare the Training Performance with Benchmark

	ROI	Sharpe Ratio	Sortino Ratio
Buy and Hold	20.69%	6.68%	5.97%
abcd Default setting	5.46%	4.59%	7.03%
abcd Optimized setting	26.20%	8.79%	13.46%

Note: Stock: AAPL, Training Data, Date: Apr 2017- Apr 2018

Table 1b. Compare the Testing Performance with Benchmark

	ROI	Sharpe Ratio	Sortino Ratio
Buy and Hold	20.96%	5.87%	7.78%
abcd Default setting	20.61%	15.41%	31.58%
abcd Optimized setting	68.94%	14.93%	24.27%

Note: Stock: AAPL, Testing Data, Date: Apr 2018- Apr 2019

We repeated the above comparison study for using the AAPL historical starting at 2014-4-24 to 2015-4-23 as training data and 2015-4-24 to 2016-4-23 data as testing data and so on until using 2017-4-24 to 2018-4-23 as training data and 2018-4-24 to 2019-4-23 as testing data. For four training and testing pairs, the average gain comparing our proposed method with buy-and-hold strategy is summarized in Table 2a for stock AAPL. We also repeated such study for SPY, and summarized the result in Table 2b. In both tables, Avg all refers to average gain of our proposed method with respect to the buy-and-hold strategy for both training data and testing data; Avg Train refers to average gain for training data and Avg Test refers to average gain for testing data. It can be observed that our proposed method outperforms the benchmark buy-and-hold strategy on average in terms of ROI, Sharpe Ratio and Sortino Ratio.

Table 2a. Average Gain of the proposed method over buy-and hold for stock AAPL.

	ROI Gain (%)	Sharpe Gain (%)	Sortino Gain (%)
Avg all	13.27	4.12	8.34
Avg Train	15.57	6.10	9.49
Avg Test	10.98	2.14	7.20

Note: Stock AAPL, Date: Apr 2014 to Apr 2019

Table 2b. Average Gain of the proposed method over buy-and hold for SPY.

	ROI Gain (%)	Sharpe Gain (%)	Sortino Gain (%)
Avg all	7.12	3.97	10.17
Avg Train	8.86	4.74	10.54
Avg Test	5.37	3.21	9.80

Note: Stock SPY, Date: Apr 2014 to Apr 2019

5 Conclusion

5.1 Free-Form Visualization

Fig. 17 shows the cash holding comparison between default setting and optimized Hyper parameter setting for testing data. It can be observed that after optimization the algorithm use the initial investment more efficiently than the default setting.

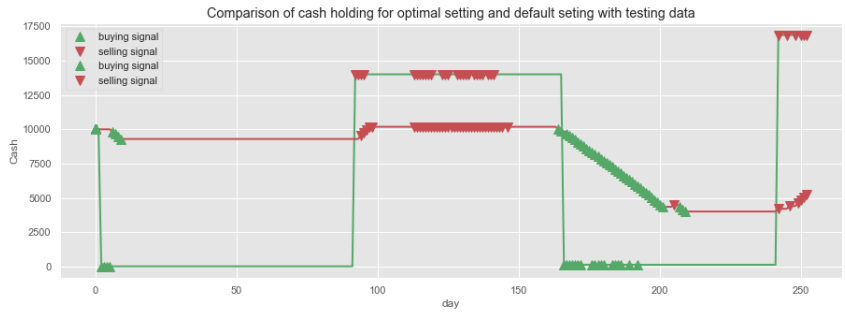

Figure 17: Cash holding over the trading period resulting from trading with default setting (red) and with optimized setting (green)

Fig. 18 shows the asset owned versus trading days resulting from trading with default setting, trading with optimized hyper parameters and the benchmark buy-and-hold strategy. It can be seen that the agent is able to perform better with the optimized hyper parameter with abcd trading policy.

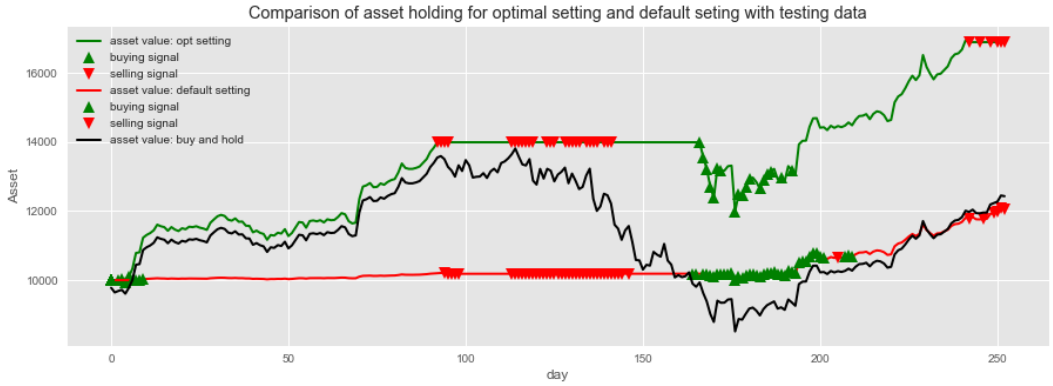

Figure 18: Asset holding over the trading period resulting from trading with default setting (red), with optimized setting (green) and with buy-and-hold strategy (black).

5.2 Reflection

The problem that I studied is on how to manage one's asset through stock or property.

The entire process of the project includes

1. Selection of investment such as stock or property

2. Selection of data source
3. Selection of trading policy
4. Understand the critical policy parameters to be optimize
5. Choose an optimization algorithm
6. Implement the optimization algorithm
7. Split the data into training data and testing data
8. Select a benchmarking criteria
9. Evaluate the performance against benchmarking criteria
10. Summarize the results

The most difficult aspect and the most interesting aspects of the project include:

- Selection of trading policy
- Optimization of trading policy

The final model and solution fit my expectation for the problem. The optimized policy works better than the buy-and-hold strategy.

5.3 Improvement

The iPython Notebook developed is for optimizing Hyper parameter of a trading policy and used the optimized trading policy for trading. It produces trading buy and sell signals.

One very useful improvement is to implement a user friendly user interface for users to choose the stock that they are interested in and generate buy/sell trading signals for the stock they choose.

Further improvements can be including more trading policies and compare their optimized performance. One may also construct ensemble trading policies.

References

[1] Wee Mien Cheung et al, "A Fuzzy Logic Based Trading System", https://pdfs.semanticscholar.org/4cfd/0ab4a3bcd883a742a0d4706746859d075fb5.pdf.

[2] TIan Yu et al, "Using Genetic Programming with Lambda abstraction to find the Technical Trading Rules", https://www.researchgate.net/profile/Shu-heng_Chen/publication.

[3] Harish K Subramanian, "Evolutionary Algorithms in Optimization of Technical Rules for Automated Stock Trading", http://www.cs.utexas.edu/ pstone/Courses/395Tfall06/resources/Thesis-harish.pdf.

[4] Sezer OB, Ozbayoglu M, Dogdu E. "A Deep Neural-Network Based Stock Trading System Based on Evolutionary Optimized Technical Analysis Parameters", Procedia Computer Science. 2017; 114:473–480.

[5] Dongdong Lv et al, "Selection of the optimal trading model for stock investment in different industries", https://journals.plos.org/plosone/article?id=10.1371/journal.pone.0212137, February 13, 2019.

[6] Pardo R. The Evaluation and Optimization of Trading Strategies. 2nd ed. Hoboken: John Wiley & Sons; 2008.

[7] Udacity, "Udacity Machine Learning Engineer Nanodegree Program, Part II, Lesson 18: Policy-Based Methods", http://www.udacity.com.

[8] Pierpaolo G. Necchi, "Reinforcement Learning For Automated Trading", https://pdfs.semanticscholar.org/f94b/ea4efc52acf695131c3128294a0f039629a8.pdf

[9] Wang et al., "Deep Q-trading", CSLT TECHNICAL REPORT-20160036 [Sunday 8th January, 2017], http://cslt.riit.tsinghua.edu.cn/mediawiki/images/5/5f/Dtq.pdf

[10] Yahoo Finance, https://finance.yahoo.com.

[11] Investopedia, https://www.investopedia.com/terms/s/sharperatio.asp, 2019.

[12] Investopedia, https://www.investopedia.com/terms/s/sortinoratio.asp 2019.

[13] Singapore Gov Website, https://data.gov.sg/dataset/private-residential-property-price-index-by-type-of-property., Singapore, 2019.

[14] Singapore Gov Website, https://www.ura.gov.sg/Corporate/Property/Property-Data, 2019.

[15] Investopedia, https://www.investopedia.com/terms/a/adjusted_closing_price.asp.

[16] Yahoo, https://finance.yahoo.com/quote/SPY/holdings/.

[17] Joseph Thomas O'Neil, "Method for utilizing a Generic Algorithm to provide constraint-based routing of packets in a communication network." United States Patent No. US 6,912,587, Jun 28 (2006).

[18] Huseinzo, https://github.com/huseinzol05/Stock-Prediction-Models/tree/master/agent.

[19] Mitchell, Melanie. An Introduction to Genetic Algorithms. Cambridge, MA: MIT Press, 1996.

[20] Sadeghi, Javad; Sadeghi, Saeid; Niaki, Seyed Taghi Akhavan, "Optimizing a hybrid vendor-managed inventory and transportation problem with fuzzy demand: An improved particle swarm optimization algorithm". Information Sciences. 272: 126–144, 10 July 2014..

[21] Whitley, Darrell, "A genetic algorithm tutorial" (PDF). Statistics and Computing. 4 (2): 65–85, 1994.

[22] S. Azodolmolky *et al.*, Experimental demonstration of an impairment aware network planning and operation tool for transparent/translucent optical networks," *J. Lightw. Technol.*, vol. 29, no. 4, pp. 439–448, Sep. 2011.