ARTIFICIAL INTELLIGENCE GENETIC ALGORITHM OF PORTFOLIO CONSTRUCTION

FRE 7043 CAPSTONE PROJECT REPORT

Submitted in Partial Fulfillment of
the Requirements for
the Degree of

MASTER OF SCIENCE
(Financial Engineering and Risk Management)

at the

NEW YORK UNIVERSITY
TANDON SCHOOL OF ENGINEERING

By

Jinfeng (Jeff) Hong

July 2020

University ID: N16342754
Net ID: jh6011

ABSTRACT

ARTIFICIAL INTELLIGENCE GENETIC ALGORITHM OF PORTFOLIO CONSTRUCTION

by

Jinfeng Hong

Advisor: Prof. Song Tang, Ph.D.

Submitted in Partial Fulfillment of the Requirement for

the Degree of MASTER OF SCIENCE

(Financial Engineering and Risk Management)

July 2020

Genetic algorithm (GA) is an adaptive heuristic search algorithm to portfolio optimization problem. It is a partial evolutionary algorithm that imitates the process of natural selection. This paper introduces how new offspring are reproduced from parents (old portfolios) by using a selection scheme, a crossover scheme, and a mutation scheme. The implementation of fitness function, back test, and probation test measures the performance of the selected portfolio. The final selected portfolios and results will be discussed and analyzed.

ACKNOWLEDGEMENT

I would like to express my deepest appreciation to my Professor Song Tang. My success in this capstone project would not have been possible without the support and nurturing of him. His high quality of teaching and heuristic education improved my problem-solving skills and critical and creative thinking. My progress of this project is inseparable from his great patient with which he helped me in solving problems and offered generous advice. When I completed the project and went back to finish writing nearly 30 pages papers, I am moved and surprised about my great achievements and progresses during these two months. I would like to extend my sincere thanks to New York University, Tandon School of Engineering for giving me this opportunity to study this advanced topic with one of the greatest professors in this school.

Contents

1. Introduction

The prediction of stock prices is a challenging task because prices are influenced by many reasons, related market news, counterparties' performances, politics, etc. Abilities to obtain information and processing information faster can help the company gain a firm foothold in the market and successfully invest in the market, which derives two approaches, fundamental analysis and technical analysis. Fundamental analysis mainly focuses on a company's financial statement to evaluate the company's fair value of business including their stock prices. While, technical analysis is a study of historical market data, economic behavior, and quantitative analysis to predict future market and understand the market sentiment behind price so that they can react swiftly. Both analysis methods must be combined with specific investment strategies in a certain time and market to help companies invest efficiently and effectively. The construction strategy of components in an investment portfolio is one of the most prominent applications. Nowadays, there are more companies using AI to achieve cost saving, low risk, and high returns. Among of most AI strategies, genetic algorithm (GA) is superior. It is better than conventional AI [1] because it also not limits to the cost function schemes, in other words, it would not break when there is a slightly change or in the presence of reasonable noise, which makes it robust. Due to its strong features and mechanisms, GA offers benefits to operate on a whole population of points and can be applied any kind of continuous or discrete optimization problems. Therefore, to explore its application in portfolio construction is essential and necessary in quantitative field.

1.1 Background

Our task is to use Artificial Intelligence Genetic Algorithm in *C++* with *sqlite3* to select and optimize a S&P500 investment portfolio. We use *Libcurl* and *Json* to process daily price, fundamental data, and index for S&P 500 composite stocks. Trading data are from 2010 to July 2020, in which data from January 2010 to December 2019 are used as training data, data from January 2020 to June 2020 are used for back testing, and data in July 2020 are used for probation testing. We need to construct 50 chromosomes (portfolios) from a set of genes (10 stocks for each) from investment parameters such as portfolio returns, risk, beta, yields, Sharpe ratio, Treynor measure and Jensen. Once we have an initial portfolio set, we use selection, crossover, and mutation schemes to create two new children from two parent chromosomes. Then we can replace parent portfolios if child portfolios have a better chromosome based on our designed fitness function. Continue the selection, crossover, and mutation cycle from generation to generation and evaluate portfolios in each generation with fitness function until the best portfolio is shown up or the max number of generations is reached.

2. Class Design

2.1 UML

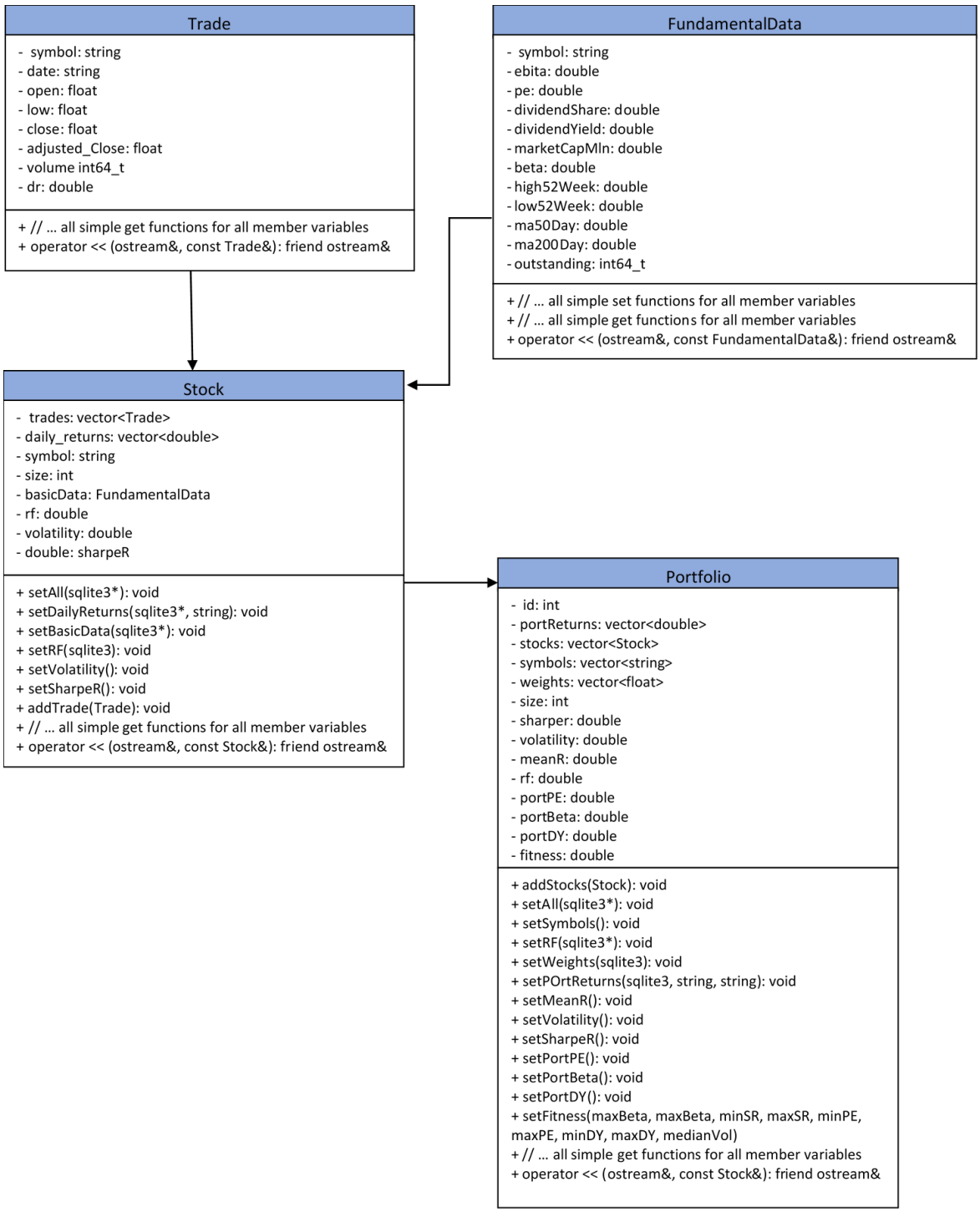

Figure 1

2.2 Designs Idea

There are four classes I'm using in my project stored in *Stock.h* header file. Stock class composites two classes, *Trade* and *FundamentalData*. Trade class objects are used to store daily trading data from Json object that will be discussed below. FundamentalData class objects store each stock's fundamental data. A Portfolio class object composites 10 Stock objects storing in a vector. Here are the design details.

2.2.1 Trade Class

The trade class is used for store data retrieving by URL and Jason. Its private elements are basic stock daily data. In this class, we have all get function for each private element and a left shift operator for print out each trade information.

2.2.2 FundamentalData Class

Even though there are many private elements, I only use P/E ratio, dividend yield, and market capitalization. Again, for each element we still have "get" and "set" functions. But here we have a function called *setAll()* which would retrieve data from database and update all private members. FundamentalData class is also a private member of Stock class so we can easily get a company's financial information. I will call FundamentalData's setAll() function in Stock class's setAll() function, thus, every time when we create a Stock object, it will have all information we need including volatility, sharpe ratio, daily_returns, and etc, which I will talk in the Stock class following.

2.2.3 Stock Class

Stock class is the most important part of the portfolio construction. I will calculate all private member first in the beginning of the program and then find out all max and min values of sharpe ratio, beta, inverse PE, and dividend yields, and median of volatility at stock level, so, later we can standardize portfolio's sharpe ratio, beta, invers P/E, and dividend yield for calculating fitness function. The Stock object has setAll() function as well, as mentioned before, that we use for setting all stocks only one time to get all private members' values. But setDailyReturns would be used many times in generating new portfolio. The default sql_select parameter is an empty

string. The reason is when the first time we call it in the beginning of the program, we can calculate stock's Sharpe ratio. After this step, every time when generating a portfolio, sql_select would not be an empty string, but a SQL statement to retrieving same trading dates data based on 10 stocks trading dates. I also retrieve the risk-free rate from US10Y table and convert it to daily risk-free rate from continuously annual yield. Then we can use the members, daily_returns and rf (risk-free rate) to calculate the stock's Sharpe ratio and volatility. Also, we will get all fundamental data mentioned above.

2.2.4 Portfolio Class

The portfolio class have some measurement private members that are weighted averaged. Portfolio class also has its setAll() function which would help us to set all portfolio level's factors. First, we need to decide the weights for each stock in a portfolio. I decide to set weights for each stock based on the industries they are in. For example, if A, B, C are three different industries in a portfolio, then sum up their market cap and calculate three industries weights proportionally. Next, I equally distribute industries' weights to stocks in each industry. To achieve this, I used only one SQL statement. By this method, we must update weights for each portfolio after crossover and mutation for calculating the fitness value. Then we calculate the portfolio returns. I used a SQL statement to find out all same trading dates for 10 stocks, thus, I can weight average every days' return and sum up them together to take average as my final portfolio return. To decide the portfolio risk-free rate, I used almost same SQL statement for lining up stock trading dates to find out the first trading dates yield in US10Y table. With the return the risk-free rate, we can calculate the portfolio volatility and Sharpe Ratio. Also, we update all private members in setAll() function by weighting corresponding 10 stocks' factors. In the setFitness function, I standardize 4 portfolio level's factors (portfolio Sharpe ratio, portfolio beta, portfolio dividend yield and portfolio inversed PE ratio) by using corresponding stock level's max and min values. Notice here, I use exponential expression for volatility because I want portfolio with volatility closing to the median to get higher score (max is 1). I assign 30% to portfolio beta, 30% to portfolio Sharpe ratio, 10% to inverse portfolio PE, 10% to portfolio dividend yield, and 15% to portfolio volatility.

3. Data sources, feeding, and populating

3.1 Data Sources:

We are using all S&P500 companies 10-year-daily trading data (total 505 companies), fundamental data for every stock, US 10 years treasury as our risk-free rate, SPY as a market reference. All data are downloading from *EOD Historical Data* website in Json Format.

3.2 Data Feeding and Populating:

There are two main tools to handle online data: one is *libcURL*, and another is *Json*. libcURL is described as a free client-side URL transfer library, supporting meanstream transfer protocol. While Json (JavaScript Object Notation) is a lightweight text-based data interchange format with straightforward syntax and is often used when data is sent from a server to a web page. To understand the processes of data feeding easier, you can image that *libcURL* is a bridge that connect between the website and your program, and Json is a container that can store data in Json format from retrieving process. Once using CURL, we store data into Json object in C++ saving as root which is like C++ vector for further operations. Even though the Json format uses a simple structure, but it is hard to read these text-based data. The solution is to find a professional website [5] to parse Json data and returns a structured and clean format. The figure 2 is the part of Json data that looks messy and unreadable, in which we cannot easily find values. After parsing Json data to the website namely, *jsonformatter*, the data is in organized and showing in the figure 3. For example, the figure shows where the Beta locates exactly with a specific set and column, so that we do not need to look for a needle in a haystack. Thanks for this website, we can use iterations to read Json objects, transform, and store them into primitive C++ data type variables.

962851,"SharesShortPriorMonth":34828293,"ShortRatio":0.96,"ShortPerce
ntOutstanding":0.01,"ShortPercentFloat":0.0081},"Technicals":
{"Beta":1.1821,"52WeekHigh":399.82,"52WeekLow":193.82,"50DayMA":366.5
848,"200DayMA":312.4363,"SharesShort":33962851,"SharesShortPriorMonth
":34828293,"ShortRatio":0.96,"ShortPercent":0.0081},"SplitsDividends"

Figure 2 (Before Parsing to Find Beta)

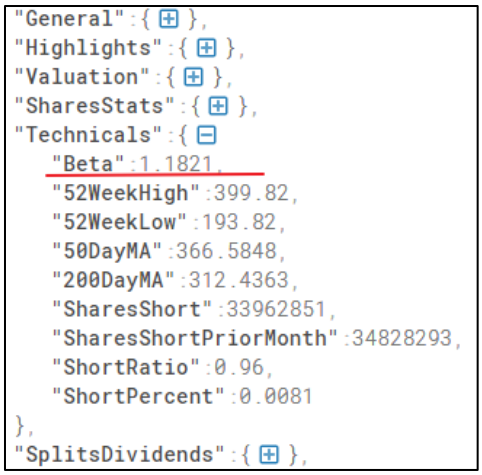

Figure 3 (After Parsing)

The most important part of data feeding is not how to retrieve data from web resource to C++ variables or objects, but how to populate data into a database highly efficiently. The management system we are using is SQLite because it is an open source database contained in C library and embedded into the end program, for which reason C++ can also handle this database efficiently. However, in my first time to create and populate all 505 S&P companies' 10 years daily trading data (total is about a million-daily data) by using *PopulateStockTable()* function, it costs me about 5 hours to finish all transaction. This, indeed, must be not acceptable. What I did was that I populate one day data of one stock every time reading it from the Json object called root. In SQLite, every time the *sqlite3_exec()* function is called, a transaction is implicitly opened. If a piece of data is inserted, the function is called once, and the transaction will be opened and closed repeatedly, which will increase the amount of IO and result in heavy burden of system operation. But, if we can open the transaction explicitly before inserting the data and then commit it once after inserting millions of data, it will greatly improve the IO efficiency, and then increase the data insertion speed. This new method applied into *FasterInsertion_Stocks()* function does finish the insertion in 5 minutes, which is a great progress compared with 5 hours. However, it also has shortage that we cannot applied multi-threading techniques due to the thread-safe setting of SQLite and cache space limitation of a transaction. For example, if there are thousands of companies' data, the program may not populate all data into the table without notification. To deal with big data, multi-thread techniques are essential and necessary. As we know, most of database systems are thread-safe in default, which makes the multi-threading difficult or impossible. Thanks to our

professor, Song Tang, for his generous help, our populating process have made another progress. When opening the database, the program must set "threading mode" by using the following codes, so that I can create 5 threads object to call *MultiThreadRetrive()* function that in this way, each thread deals with 101 companies' data and together finish insertion task in few minutes. Overall, the last method is preferred and help program become more robust, efficient, and flexible.

4. AI Strategies

AI Strategies are the heart of the project, and it consists of four main operations namely, fitness operation, crossover operation, mutation operation, and selection operation. A fitness operation measures the quality of the produced solution (optimal portfolio). A portfolio would be assigned a fitness value and compared to the benchmark value to tell the genetic algorithm whether the portfolio is the optimal one or not. The crossover operation produces new chromosomes (portfolios) by combining existing chromosomes. Each time, based on the designer's purpose, a crossover operation takes parts of solution encodings from two existing portfolios and combines them into one or two new portfolios. In my strategy, it chooses to produce two new portfolios to save the procedure time. Before a genetic algorithm finishes the production of new chromosomes, after it performs a crossover operation successfully, it performs a mutation operation. A mutation operation makes random, but small, changes to a portfolio. This prevents the falling of all solutions into a local optimum and extends the search space of the algorithm. A portfolio is consisting of 10 different stocks, and few of its stocks may be changed to new stocks not existing in that portfolio from stocks dataset. Last, the selection operation is the procedure to choose paired parents before the crossover function. My strategy uses roulette wheel selection instead of the ranking selection because generally, roulette wheel selection would give us a better prediction. The introduction order is from the most important to the least import: Fitness operation → Crossover operation → Mutation operation → Selection operation. The program procedure is shown in figure 4:

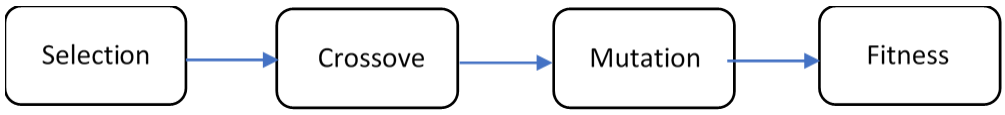

Figure 4

4.1 Strategies Flowchart

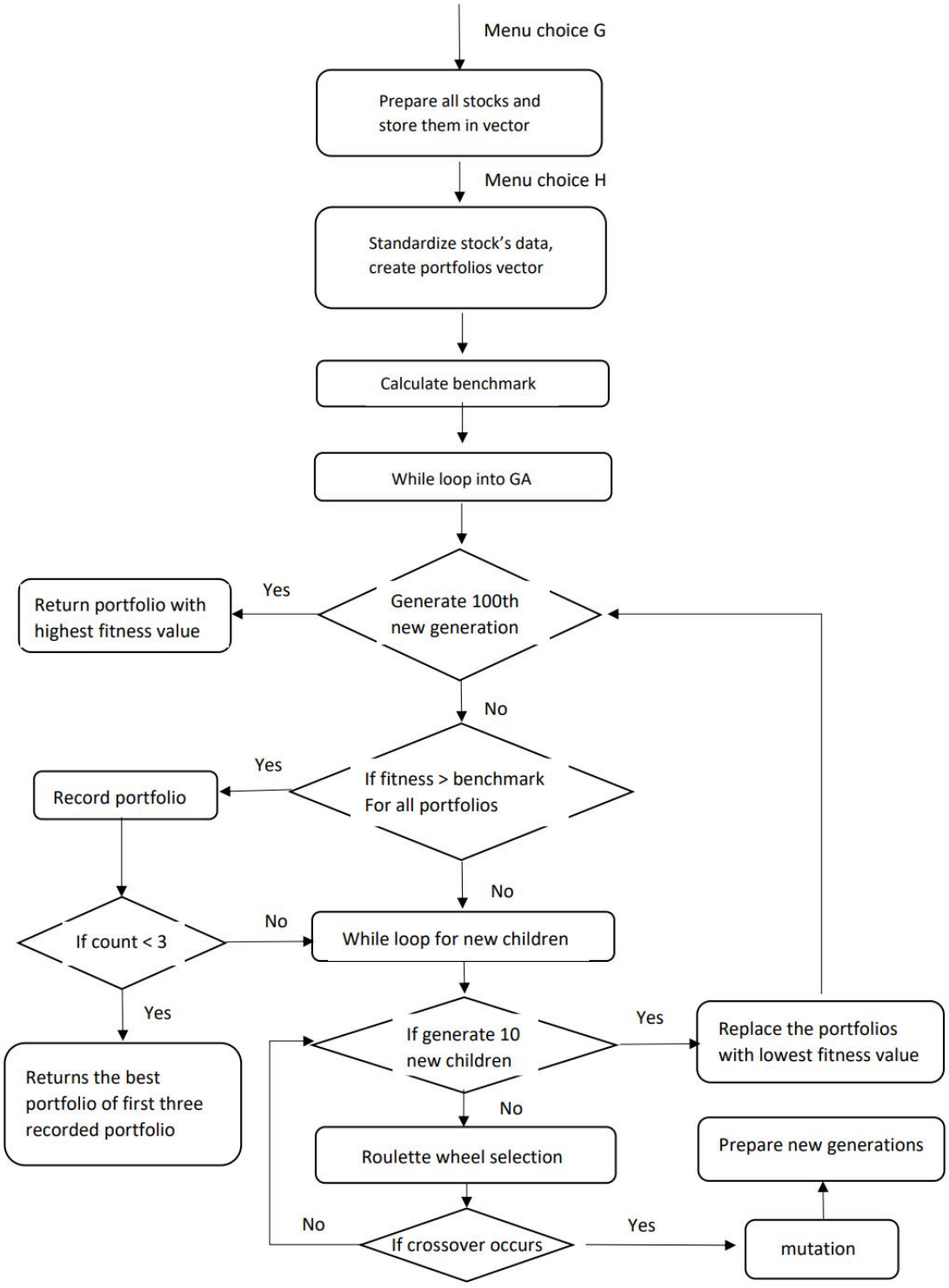

Figure 5

4.2 Fitness operations

4.2.1 Components and Reasons

After doing research, I decided to use P/E ratio, Beta, Sharpe Ratio, Volatility, and Dividend Yield as components in my fitness function with user input weights.

In reference [2], the authors using Sharpe Ratio as main metrics to analyzing their return values, which gives me the point that Sharpe Ratio could be an important indicator of strategy efficiency because this would give us the idea of the relationship between risk and portfolio returns. In other words, it measures the excess returns per systematic risk. People also seek a higher return with lower risk no matter whom they are, risk aversions or risk lovers. Therefore, we could assign large weights on this indicator in the fitness function to award portfolio which match to that idea.

Beta is also another important factor when construct portfolio. In respect of choosing beta as our part of fitness function, reference [3] points out,

> *It is easy to see that β_p measures portfolio volatility relative to the benchmark index or the capital market. Indeed, if a portfolio is well chosen such that returns of the benchmark index and portfolio are highly correlated, then β_p efficiently approximates the volatility ratio of the portfolio to the benchmark index and hence measures sensitivity of the portfolio to market fluctuation.*

This reflects the trend between market and stock. The paper mainly focusses on how portfolio beta with optimal weights can be applied into GA model and shows us that the beta plays an important role on their portfolio, which brings back a quite competitive profit. It also proves that beta can be an effective tool in a stable market where it is bullish or bearish. Therefore, I put beta into my fitness. Another point is in this project, we are more likely to do passive investment in the long run. Therefore, we aim to match a given market index and align with the market.

The P/E ratio is a measure not only of last year's performance, but also a measure of the markets expected performance of the company. P/E ratio is normally best use in stock screening, as a portfolio aspect, it can have the same effect and help us to construct the fitness functions to some extends. However, P/E ratio has its limitation and we should be careful when using it. Generally, people think low P/E is better, but which is a wrong concept. The reason for that is "*Valuations and growth rates of companies may often vary wildly between sectors due both to the differing ways companies earn money and to the differing timelines during which companies earn*

that money." [4]. Therefore, due to its limitations, we can assign small weights, such as 10% to it and keep it in our portfolio.

Warren Buffett says, "To be successful at business, you have to understand the underlying financial values of the business." This might be an encouragement for us to implement dividend yield combinations in our portfolio construction. A research paper [6] called, *Dividend Yield Combinations,* reports the combinations of six factors namely Dividend Yield, Value, Momentum, Quality, Growth and Dividend Growth. All combinations have outperformed single dividend yield strategy. Although the paper does not point out what the value factors are used, dividend yield and P/E ratio may still give us an unexpected output in our strategy because both reflect somehow a company's growth and health. In my portfolio, technical factors are protagonists and fundamental factors are supporting roles, so dividend yield and P/E ratio will get small weights.

Volatility are essential factor in technical analysis and should be also included in the portfolio construction. It measures the risks of an investment: normally high risk may have high returns; low risk may have low return. In 2020, circuit breaker mechanism was triggered 4 times to halt the market followed by another increase, which is thrilling roller coaster. I am not a risk aversion or preference, so in my portfolio, I take median volatility among all stocks as my benchmark factor.

4.2.2 Scoring the Fitness Value

First, we need to decide the weights for each stock in a portfolio. I decide to set weights for each stock based on the industries they are in. For example, if A, B, C are three different industries in a portfolio, then I can sum up their market cap and calculate three industries weights proportionally. Then, I equally distribute industries' weights to stocks in each industry. To achieve this, I used only one SQL statement. By this method, we have to update weights for each portfolio after crossover and mutation for calculating the fitness value.

Next, to calculate the Sharpe Ratio, we need to line up stocks trading dates in a portfolio to make sure they have the exact same trading dates. I also used one SQL statement to get the data from database and at the same time to get the first trading date's treasury yield as a risk-free rate from US 10-years treasury yield table. Notice that we need to convert this continuous annual yield to daily yield to keep the consistency of inputs data in the fitness function. We also need the

volatility calculated from the formula, $\sigma_P = \sqrt{\frac{\sum(r_i - r_{mean})^2}{n}}$. Once we have risk-free rate and portfolio returns, we can calculate the Sharpe Ratio.

Last but not least, the portfolio P/E ratio, Beta, and Dividend Yield are getting from fundamental data by weighted averages. However, in general, the lower the portfolio P/E, the better the performance. We should inverse the portfolio P/E ratio first and normalize all three factors (P/E, Sharpe Ratio, beta, dividend yield) by function $x_{std} = \frac{x_i - x_{min}}{x_{max} - x_{min}}$, where the min and max values are from 50 portfolio.

The fitness value could be constructed by weighting the scaled Sharpe Ratio, scaled Beta, and scaled inversed P/E ratio, transformed Volatility, and scaled Dividend Yield. The weights are manually input and sum up to one. First, we need to scale all factors except volatility to be less than or equal to 1 (P/E ratio is inversed ahead of this step) and transformed volatility is calculated by exponential expression to ensure a portfolio with volatility closer to median getting a higher score. All minimum and maximum values are from stock's level, so we only one standardizing rule to do scaling process in all generations. Following is the fitness function.

$$fitness = w_1 \frac{SR - minSR}{maxSR - minSR} + w_2 \frac{Beta - minBeta}{maxBeta - minBeta} + w_3 \frac{1/PE - \min\ \_inv\_PE}{\max\ \_inv\_PE - \min\ \_inv\_PE} + w_4 \frac{DY - minDY}{maxDY - minDY} + w_5 \exp(-|volatility - medianVol|)$$

I set a benchmark/stop sign to finish the GA construction that either the fitness reach above the benchmark three times or 100 generations has been reproduced, then choose the highest score as our final portfolio shown in above flowchart. The benchmark is market fitness value, which is from weighted scaled percentiles of all factors plus 15% directly from volatility:

$$fitness = w_1 \frac{PctSR - minSR}{maxSR - minSR} + w_2 \frac{PctBeta - minBeta}{maxBeta - minBeta} + w_3 \frac{Pct\left(\frac{1}{PE}\right) - \min\ \_inv\_PE}{\max\ \_inv\_PE - \min\ \_inv\_PE} + w_4 \frac{PctDY - minDY}{maxDY - minDY} + 15\%$$

4.3 Issues and Improvement of Fitness Function

4.3.1 Issues

My old model does not beat the market all the time maybe because of the following reasons:

1) The weights in fitness function are not a good fit with portfolio.
2) Some factors may not good fit with fitness model.
3) The benchmark should be redesign: for example, I used 95 percentiles data from factors except volatility before; I could adjust it to higher or lower level in either way.
4) It can only output one result that's fitness value overpasses the benchmark without considering other potentially better portfolio.

4.3.2 Improvements

To improve the model, first of all, I changed part of the codes so that I can choose the best of the first three portfolios whose fitness values greater than benchmark. Then I kept changing the weights in fitness function with benchmark. Each change brought 20 or more test results; therefore, I can have a general idea of which weights should be used in the fitness. After testing, I found that my fitness function weights would be one of followings as followings:

$$fitness = 0.37 * Beta + 0.35 * Sharpe\ Ratio + 0.03 * PE\ Ratio + 0.1 * Dividend\ Yield + 0.15 * Volatiltiy$$

4.4 Crossover

I applied one-point-crossover but there are 3 unique improvements to help increase the GA's training efficiency. First, I improve the randomness of the crossover. A simple one-point-crossover only exchanges the later parts of two chromosome but keeps previous parts unchanged. This brings up a potential issue that if 50 portfolios have a large number of bad stocks in their later parts, in which case no matter how GA does crossover, it is hard to get final portfolio in few generations, and even if you get a portfolio after 100 generations, it is highly likely that its back and probation tests would not outperform. What we can do as shown in figure 6 is to shuffle two offspring in the beginning of the crossover so that we have equally chance to switch stocks between two portfolios.

Second, I improve the uniqueness of the crossover results. To ensure an offspring has 10 different stocks' symbols, we need to check if the switching stock of one portfolio already exist in non-switching part of another portfolio. For example, on Figure 6, the first red block is not successfully switching because the stock B is already in portfolio 1's previous part; then, the second red block is not successfully switching because the first red block is not switching resulting in that the stock B is still in portfolio 2.

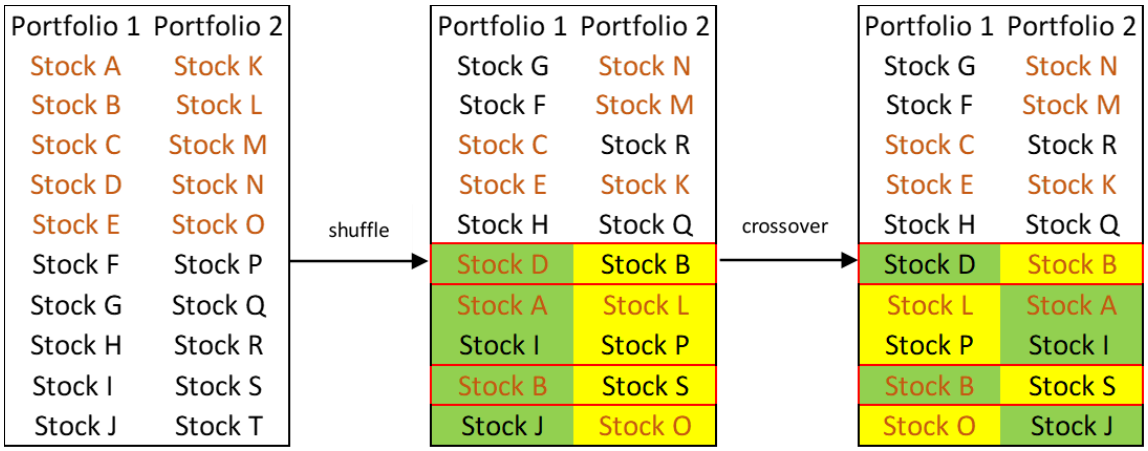

Figure 6

Third, the uniqueness of portfolios is improved. The Crossover() function will either return 1 or -1 to decide whether mutation and new children would be recorded. If crossover does happen, and at least two stocks switching successfully, then it returns 1 and Mutation() function and following process will go on, otherwise start a new while loop. In this mechanism, all 50 portfolios are unique without any repetitions.

4.5 Mutation

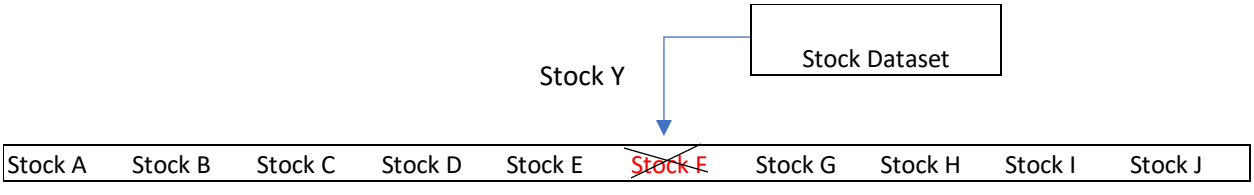

Figure 7

Mutation will happen if random number is less than mutation rate set as 3%. For each stock, if needed to be mutated, choose a stock from symbols set and check if it already exists in the offspring: if yes, no mutation happens; if not, replace the old one. As figure 7 shown, the Stock F will be substituted to Stock Y chosen from stock dataset, and Stock Y is different from other 9 stocks in the portfolio.

4.6 Selection

The total fitness value is getting before the while loop of production of new portfolios. When traversing 50 portfolios to find a portfolio whose fitness value is greater than benchmark, the total fitness value is calculated. Then in the selection function, it will traverse all fitness values from portfolios and accumulate relative fitness values. If the accumulative value is greater than the generated random number, selection stops the accumulation at that point as a stock position in the stock's dataset.

5. Back Testing

Before the back testing, each fitness settings were tested at least 20 times. I picked up the three best portfolios with 96%-percentiles-benchmark that the fitness value is 0.688554. In the back testing, I will compare the cumulative returns of portfolio against SPY (market)'s one starting from January 2020 to the end of June 2020, then record portfolio's fitness value and all its stocks members for probation test later.

a) Portfolio 1:

tickers	"EVRG", "DHR", "AVB", "NVDA", "IQV", "NOC", "BK", "BFB", "TDG", "ALK"
Fitness value	0.699978
SPY cumulative returns	-0.00757 to -0.04104
Portfolio cumulative returns	-0.00721 to 0.170332

Table 1

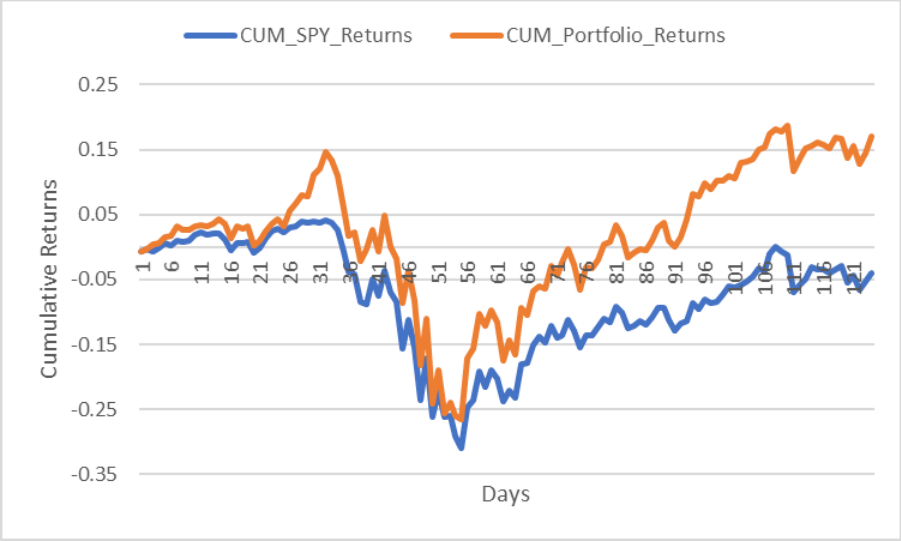

Figure 8

symbol	name	sector
BF.B	Brown-Forman Corp.	Consumer Staples
BK	The Bank of New York Mellon	Financials
DHR	Danaher Corp.	Health Care
IQV	IQVIA Holdings Inc.	Health Care
ALK	Alaska Air Group Inc	Industrials
NOC	Northrop Grumman	Industrials
TDG	TransDigm Group	Industrials
NVDA	Nvidia Corporation	Information Technology
AVB	AvalonBay Communities	Real Estate
EVRG	Evergy	Utilities

Figure 9

b) Portfolio 2:

tickers	"ABC", "ALGN", "DHR", "ABMD", "BXP", "CDW", "LEN", "LRCX", "ADBE", "HOG"
Fitness value	0.701344
SPY cumulative returns	-0.00757 to -0.04104
Portfolio cumulative returns	-0.00955 to 0.108691

Table 2

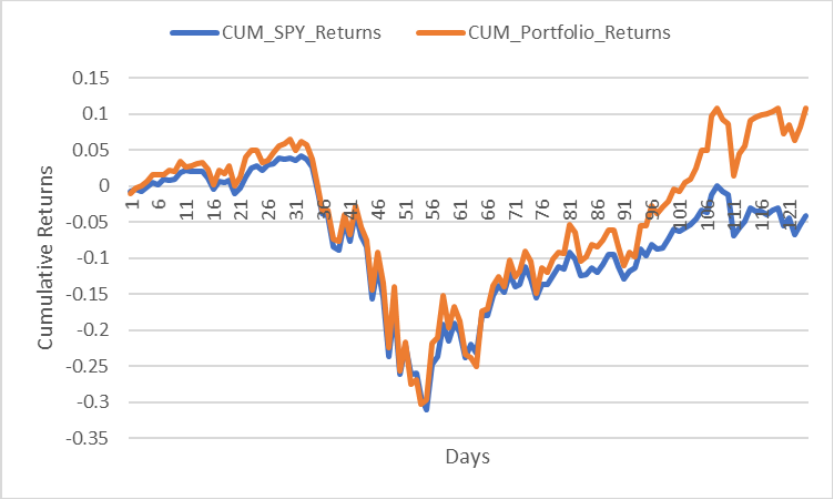

Figure 10

symbol	name	sector
HOG	Harley-Davidson	Consumer Discretionary
LEN	Lennar Corp.	Consumer Discretionary
ABC	AmerisourceBergen Corp	Health Care
ABMD	ABIOMED Inc	Health Care
ALGN	Align Technology	Health Care
DHR	Danaher Corp.	Health Care
ADBE	Adobe Inc.	Information Technology
CDW	CDW	Information Technology
LRCX	Lam Research	Information Technology
BXP	Boston Properties	Real Estate

Figure 11

c) Portfolio 3:

tickers	"MET", "DLTR", "SIVB", "NEE", "NOW", "ZTS", "EA", "TTWO", "CNC", "FLT"
Fitness value	0.694374
SPY cumulative returns	-0.00757 to -0.04104
Portfolio cumulative returns	-0.0000617 to 0.062919

Table 3

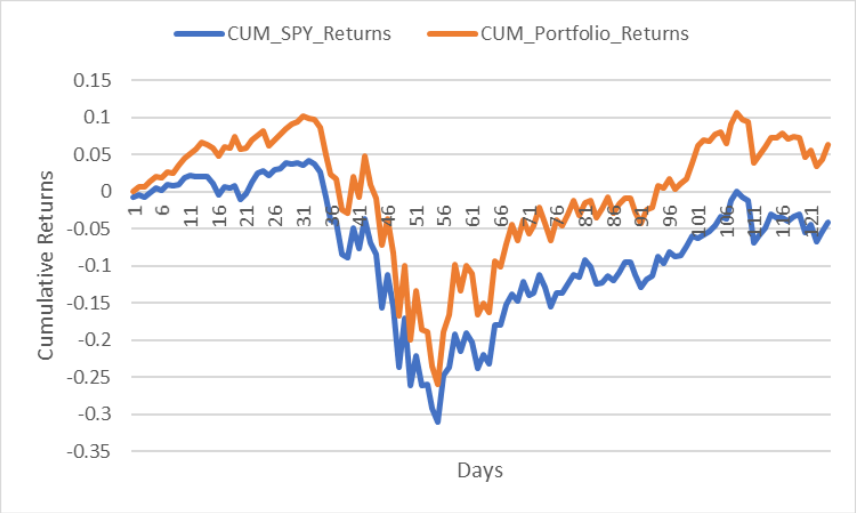

Figure 12

symbol	name	sector
EA	Electronic Arts	Communication Services
TTWO	Take-Two Interactive	Communication Services
DLTR	Dollar Tree	Consumer Discretionary
MET	MetLife Inc.	Financials
SIVB	SVB Financial	Financials
CNC	Centene Corporation	Health Care
ZTS	Zoetis	Health Care
FLT	FleetCor Technologies Inc	Information Technology
NOW	ServiceNow	Information Technology
NEE	NextEra Energy	Utilities

Figure 13

Analysis: I plotted the cumulative returns of portfolio and SPY (market reference). As you see, these fitness settings beat the market (SPY). However, the models do not always outperform in such a good way. Sometimes, trends have the first previous period underperforming, but later period outperforming and keeps the upward trends; sometimes, they are overlapping the market trending line, which means they do not beat or lose in the market. It is not always good to mimic the market without outperforming. To further analysis the performance of the GA, we need probation test on whole July trading dates.

6. Probation Testing

After back testing, the program will start the probation test. Again, it compares the cumulative returns of portfolios against the SPY's one using all July 2020's trading data. The probation test will print out Beta, Sharpe Ratio, Volatility, Cumulative Returns of SPY and Portfolio for comparison purpose between them. It will also print out the P/E ratio and Dividend Yield so that we can compare the performance between different portfolios.

a) Portfolio 1:

tickers	"EVRG", "DHR", "AVB", "NVDA", "IQV", "NOC", "BK", "BFB", "TDG", "ALK"	
Comparison	**SPY**	**Portfolio**
Beta	1	1.14439
Sharpe Ratio	0.29426	0.338577
Volatility	0.00816147	0.0112855
Cumulative Returns	0.00550696 to 0.0515265	0.00810186 to 0.0825563
P/E Ratio		57.7704
Dividend Yield		0.0193968

Table 4

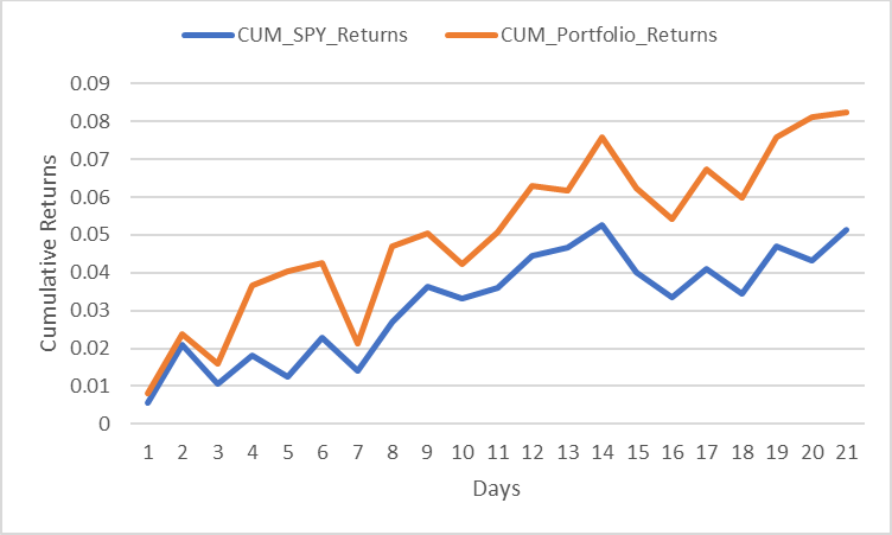

Figure 14

b) Portfolio 2:

tickers	“ABC”, “ALGN”, “DHR”, “ABMD”, “BXP”, “CDW”, “LEN”, “LRCX”, “ADBE”, “HOG”	
Comparison	**SPY**	**Portfolio**
Beta	1	1.1445
Sharpe Ratio	0.29426	0.294479
Volatility	0.00816147	0.0130452
Cumulative Returns	0.00550696 to 0.0515265	0.0154493 to 0.0825329
P/E Ratio		31.6653
Dividend Yield		0.00864878

Table 5

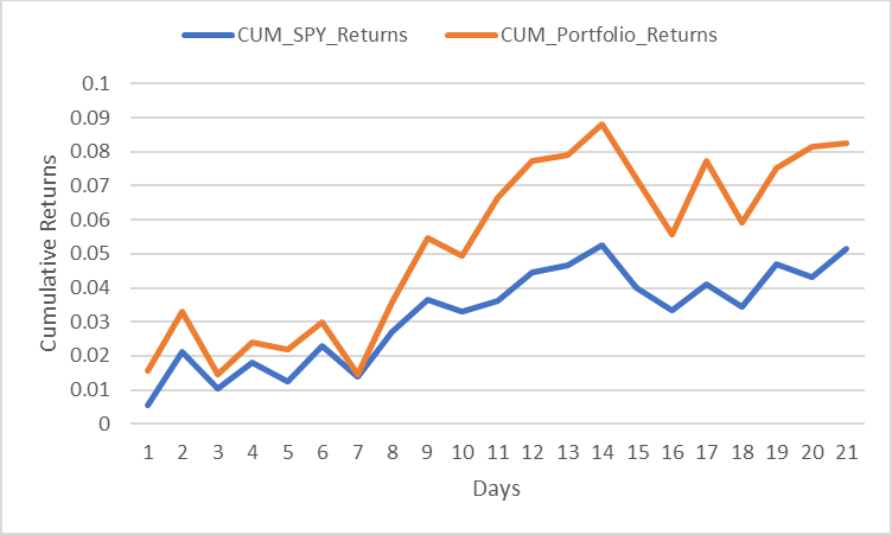

Figure 15

c) Portfolio 3:

tickers	"MET", "DLTR", "SIVB", "NEE", "NOW", "ZTS", "EA", "TTWO", "CNC", "FLT"	
Comparison	**SPY**	**Portfolio**
Beta	1	0.853772
Sharpe Ratio	0.29426	0.427257
Volatility	0.00816147	0.00847585
Cumulative returns	0.00550696 to 0.0515265	0.00218975 to 0.07867
P/E Ratio		39.3072
Dividend Yield		0.00950236

Table 6

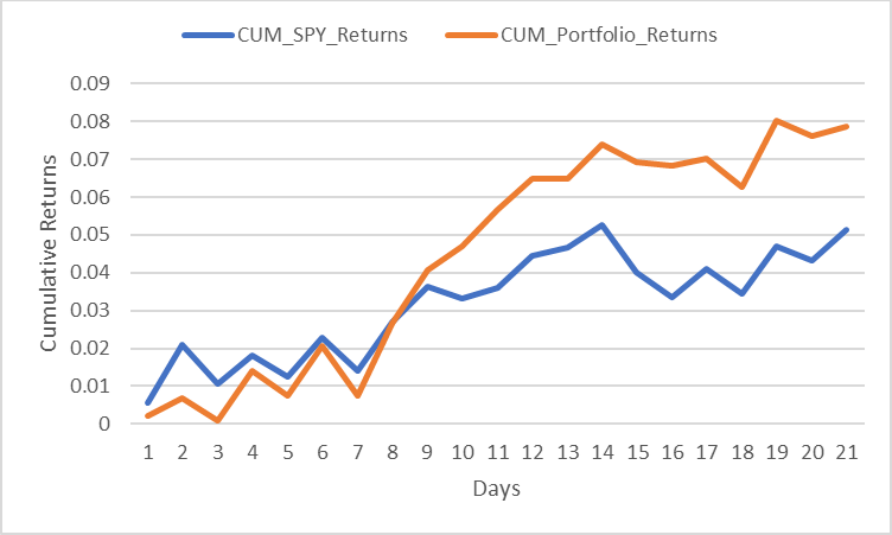

Figure 16

Analysis: Overall, all three portfolios beat the market. Comparing to SPY, I found out that portfolios' Beta are close to 1, which means they have same trend as SPY, and their volatility also close to SPY's volatility indicating that they have same risk as market; in other words, that they may not have high risk with high returns. Besides, their Sharpe Ratios are greater than market. It suggests Sharpe Ratios is a key factor to decide whether a portfolio from my GA would beat the market or not. Therefore, I guess that, if Beta and Sharpe Ratio both greater than market in my strategy, program have a high likelihood to output a portfolio beating the market. While P/E ratio and Dividend Yield are auxiliary factors in respect of fundamental analysis to help us find which companies have high potential returns. From the comparisons of P/E ratio and Dividend Yield, we can have a simple conclusion that portfolio 1 is better than other two portfolios because its higher P/E ratio and Dividend Yield.

7. Conclusion and Future Work

Using genetic algorithm to help investor to construct portfolio looks promising. The best portfolio is the first one whose cumulative return is 10 times higher than its beginning and the performance of the strategy is acceptable. However, it does not always guarantee to beat the market in probation test though may beat the market in the back testing. There are many aspects we can consider in the future regarding to improve the optimal portfolio and efficiency of the program. The first future improvement, for example can be the stocks' weights in a portfolio. In my settings, the weights are created from portfolio's industries' portions. We can use weights from efficient frontier, thus, we will have the minimal variance with optimal returns. Another improvement can be the fundamental data of each stocks. The fundamental data on *EOD Historical Data* are not updated every day, so that my program may be influenced by the lagged effect. If we have newest data, the program can be more accurate and practical. Next, to find the best combination of fitness's factors, it is possible to design a more artificial intelligent function that can do the job for us automatically in a short time. In my who project, I manually adjusted factors' weights in the fitness function and then tested each combination at least 20 times. This is a time-consuming work and results are not always outperforming the market. Also, I can even change some factors or add more factors to check whether they help in selecting optimal portfolio and increase the passing rate both in the back and probation tests. Finally, the mutation function can be more complexity to choose stocks in higher ranks that are defined manually by stock screening operation that would be another huge topic beyond this project.

References

[1] Yadav, R., & Ahmad, W., *Benchmark Function Optimization using Genetic Algorithm*

[2] Iskrich, D., & Grigoriev, D. *Generating Long-Term Trading System Rules Using GA Based on Analyzing Historical Data*

[3] Oh, J.K., Kim, Y.T., Min, SH., & Lee, Y.H., *Portfolio algorithm based on portfolio beta using genetic algorithm*

[4] Hayes, A., *Price-to-Earnings Ratio – P/E Ratio*, Retrieved March 17, 2020 from https://www.investopedia.com/terms/p/price-earningsratio.asp

[5] Retrieved August 3, 2020, from https://jsonformatter.curiousconcept.com/

[6] Rabener, N., *DIVIDEND YIELD COMBINATIONS*, Retrieved August 3, 2020 from https://www.factorresearch.com/research-dividend-yield-combinations

APPENDIX