Case study: Portfolio selection by Genetic Algorithm

Introduction

When investors build their portfolio, it is not only how much money they can make from the portfolio, but also how high of risk they should take to make the money. So, most of the investors would use Sharpe ratio to evaluate a portfolio because Sharpe ratio considers both return and risk. The formula of Sharpe ratio:

Sharpe Ratio = (portfolio return – risk free rate) / risk

Portfolio return: the average return rate of the portfolio in certain period (weekly, monthly, yearly, etc.).
Risk free rate: the cost of money in that period. Usually investors would use the treasury bond yield to be the risk free rate.
Risk: the volatility of the portfolio return rate.

It is obviously that investors would prefer high return by taking low risk. So, depending on the formula, the higher Sharpe ratio means the better portfolio.

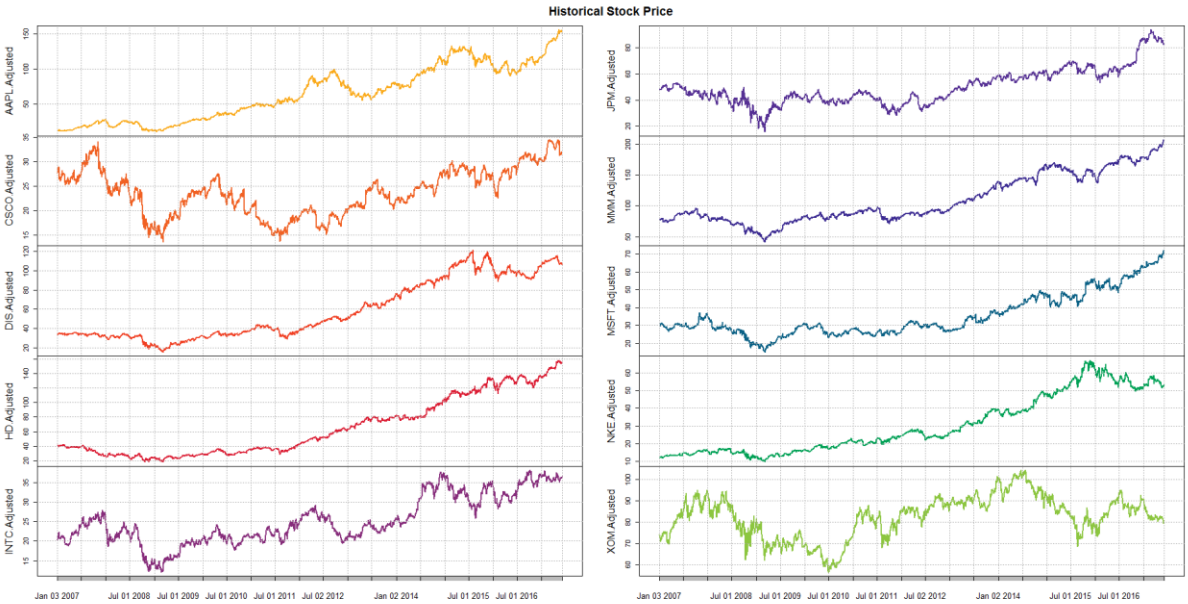
From previous parts, we know that Genetic Algorithm can be used to search optimized results. In this case, we are trying to use Genetic Algorithm to find a portfolio with high Sharpe ratio.

Data Description:

We picked 10 stocks from Dow Jones Index components. They are:

- Apple
- Cisco
- Disney
- Home Depot
- Intel
- JPMorgan Chase
- 3M
- Microsoft
- Nike
- Exxon Mobil

We use the R package of quantmod (a package for quantitative finance) to download the historical data.



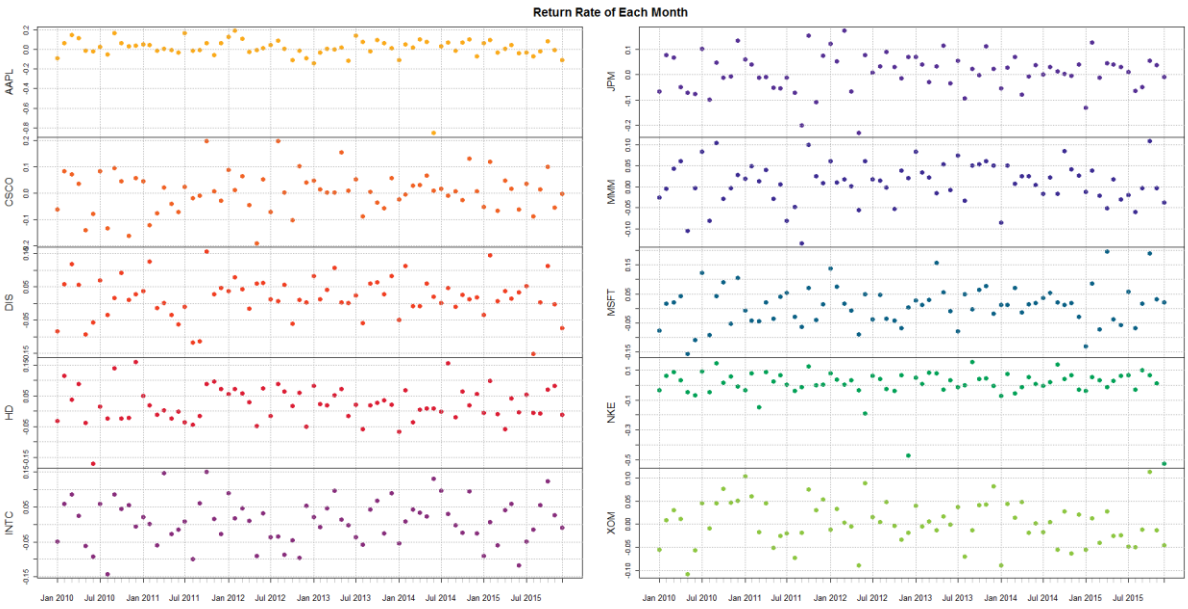
Historical daily adjusted price (because some stocks have dividends) of the 10 stocks

In the following parts, we would use the historical price (adjusted price) data from 2010 to 2015 of these 10 stocks as training data to build a portfolio, then use the stock price data of 2016 as testing data to see how the portfolio performs.

In the Sharpe ratio formula, we need to know risk free rate. We would use one month treasury bond yield rate on May 28, 2017 (the day we build the portfolio) which is 0.00728.

Data Transformation

In Sharpe ratio, we need the average return in certain period. In this case, we choose 1 month as the time period. The return rate of each month of the 10 stocks:

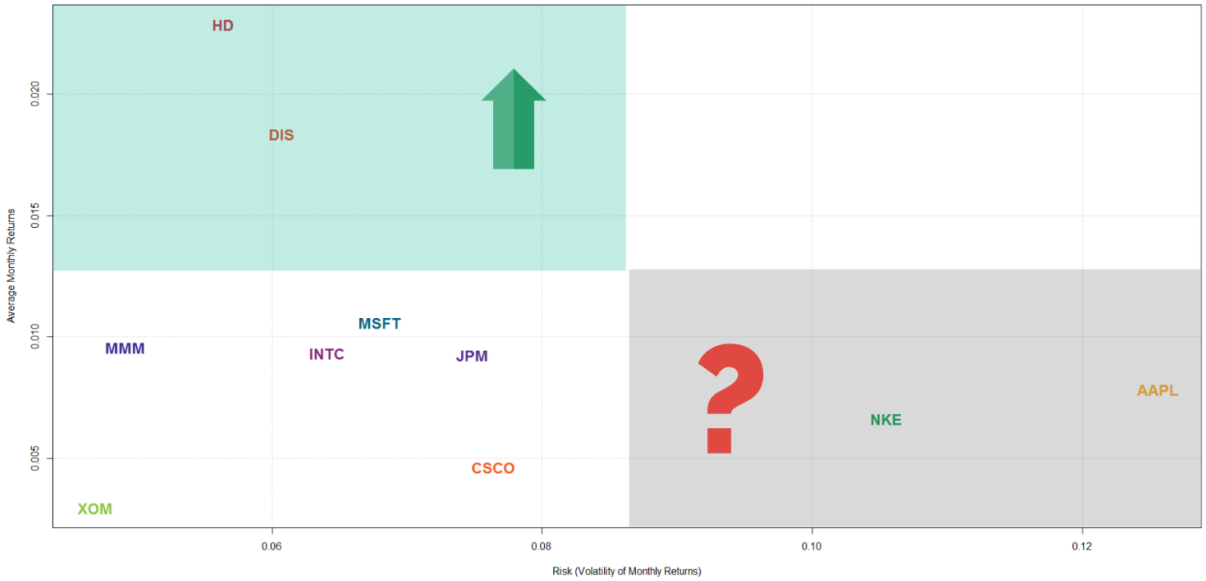


We also compute the average monthly return and return volatilities for each stock.

	AAPL	CSCO	DIS	HD	INTC	JPM	MMM	MSFT	NKE	XOM
Average Monthly Return	0.007824	0.004642	0.018357	0.022880	0.009322	0.009249	0.009570	0.010596	0.006622	0.002946
Monthly Return Standard Deviation	0.125636	0.076379	0.060741	0.056388	0.064094	0.074833	0.049181	0.067998	0.105460	0.046942

Data analysis

We put the return and risk (volatilities) of each stock in a single chart.



Depending on the “high return + low risk” rule, we probably need to assign more weights on the stocks like HD (Home Depot) and DIS (Disneyland), and less weights on stock like NKE(Nike) and AAPL(Apple). In the next part, we would use genetic algorithm to find the best weight for each stocks.

Experimental Results

To use Genetic Algorithm to find a portfolio with high Sharpe ratio, we need to design a good fitness function. In the fitness function, we use adjusted Sharpe ratio:

Adjusted Sharpe ratio = Sharpe ratio - risk adjustment

We create variable, risk adjustment, and subtract it from Sharpe ratio. Because, we think the risk adjustment can help us to find a portfolio with more stable historical performance, which might also bring more stable future performance (more predictable performance).
Since the average monthly return of the 10 stocks is 0.1, so, We choose 0.1^2 * 10% as a base line of risk:

- If the risk is less than 10%* 0.1^2 = 0.001:
  riskAdjustment = 0
- If the risk is more than 10%* 0.1^2 = 0.001:
  riskAdjustment = risk*1000
- If the risk is more than 20%* 0.1^2 = 0.002:
  riskAdjustment = risk*1000 + (risk-0.002)*1000

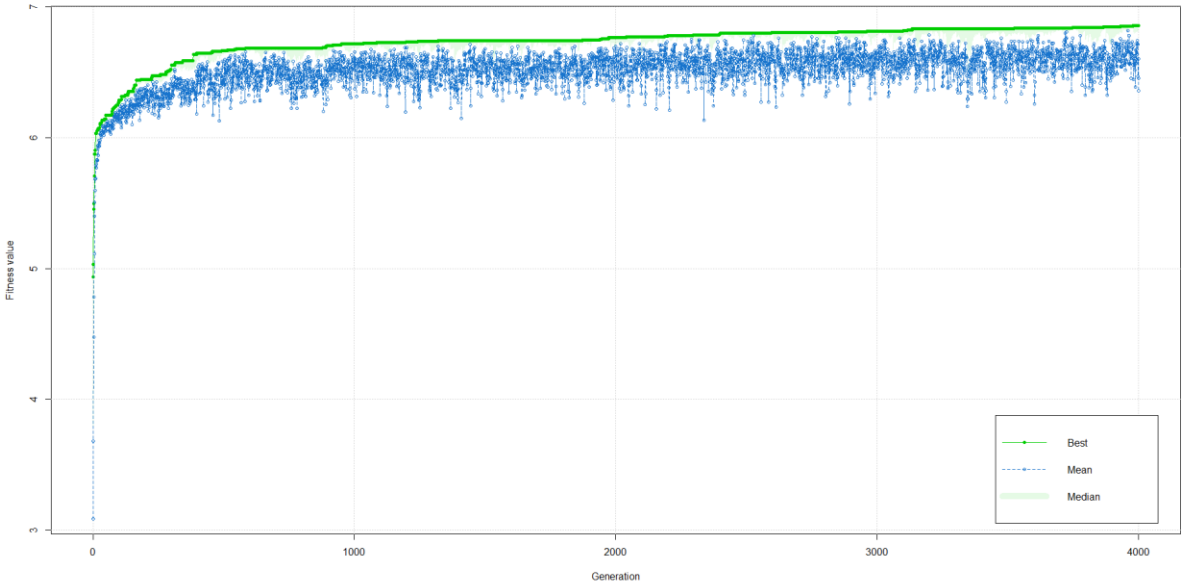
The more risk, the more punishment on Sharpe ratio. So, if a portfolio has too much risk, unless its sharpe ratio is really high, it would not be chosen.

After setting up fitness function. We need to decide the parameters:

- generation
- probability of mutation
- probability of crossover

Since we are trying to find the best solution of the fitness function (highest sharpe ratio). It is not the same with what we normally did: model building + predicting. From the subset of the training data, it is impossible to find the best solution of fitness function for the whole training data set. Which means cross-validation validation is not appropriate for parameter tuning. It is better to write a function to do directly parameter tuning by comparing the GA@solution of different parameters.

We start with number of generation. For the value of generation, the more generation, the higher accuracy of searching and longer running time. We run the GA with default value for all parameters, except setting maxiter = 4000 (it means 4000 generations):



From the model performance chart, we can see that, when generation is more than around 1000, the result of the algorithm only improved less than (6.8-6.5)/6.5 = 4.6%. But the computation time increased 4000/1000 -1 = 300%.
The result from 1000 generation, which is around 6.6 sharpe ratio (after risk adjustment) is already great enough.
Considering both the runtime and accuracy, we would use 1000 as the value of maxiter (generation).

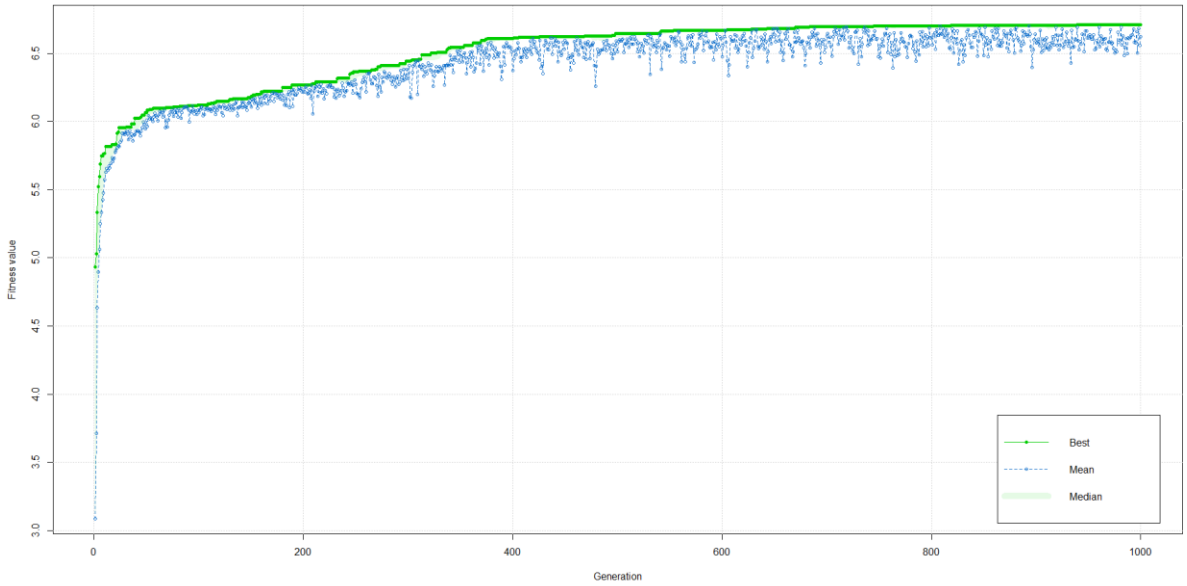
Mostly, researchers use crossover rate on level 0.7-0.9 and mutation on 0.05-0.2. ("Parameter tuning for configuring and analyzing evolutionary algorithms",A.E. Eiben, S.K. Smit1) So, we would like to search the best value in these 2 ranges.

We use nest loop to test all possible combination of crossover rate (0.7-0.9, increased 0.01 each time) and mutation rate (0.05-0.2, increased 0.01 each time) when generation is 1000. The best values are:

crossover rate = 0.7

mutation rate = 0.06

The GA model performance when maxiter=1000, crossover= 0.7, mutation = 0.06 :



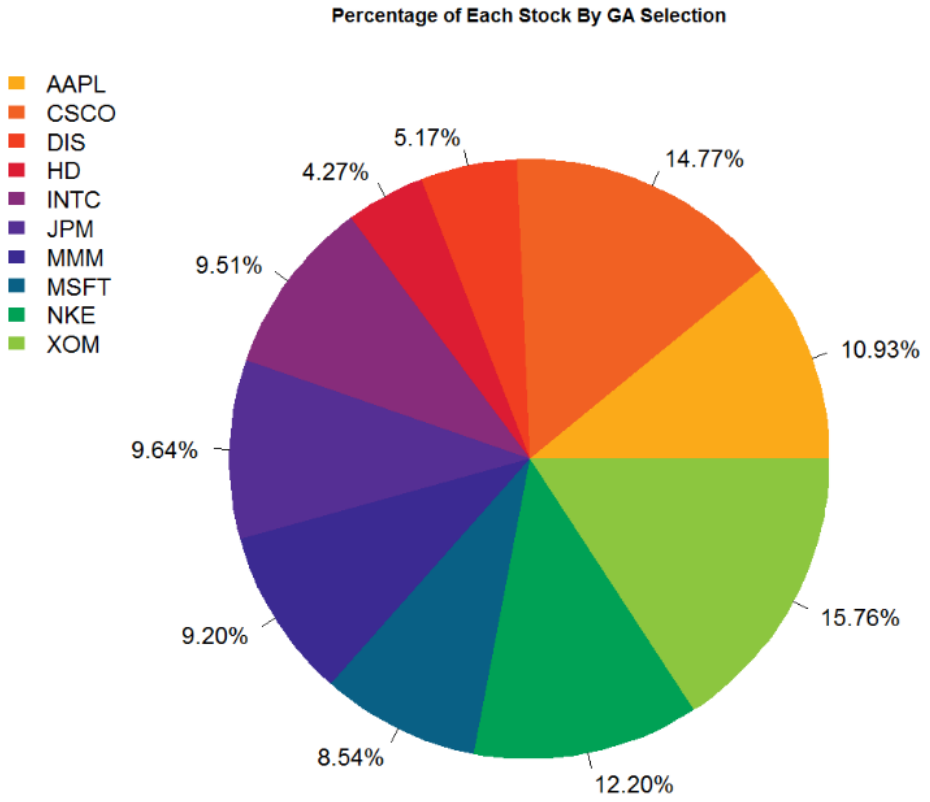
We get the best portfolio with Adjusted Sharpe ratio = 6.70821. The weights of each stock:

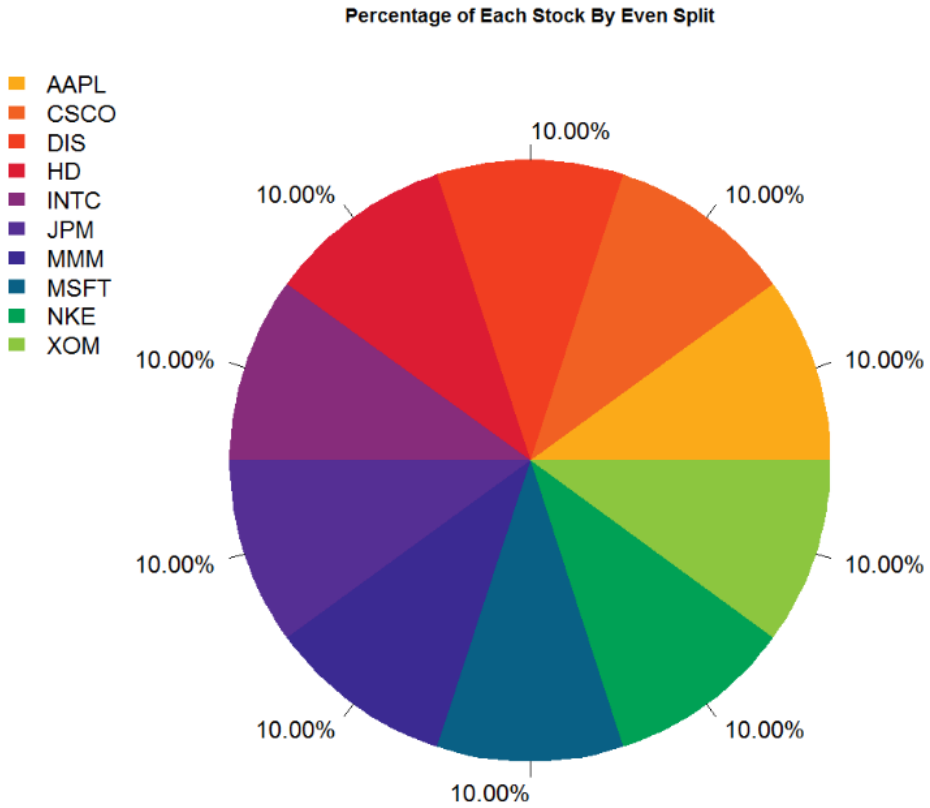
	AAPL	CSCO	DIS	HD	INTC	JPM	MMM	MSFT	NKE	XOM
Weight	10.92%	15.15%	5.30%	4.31%	9.50%	9.52%	9.37%	8.58%	12.32%	15.02%

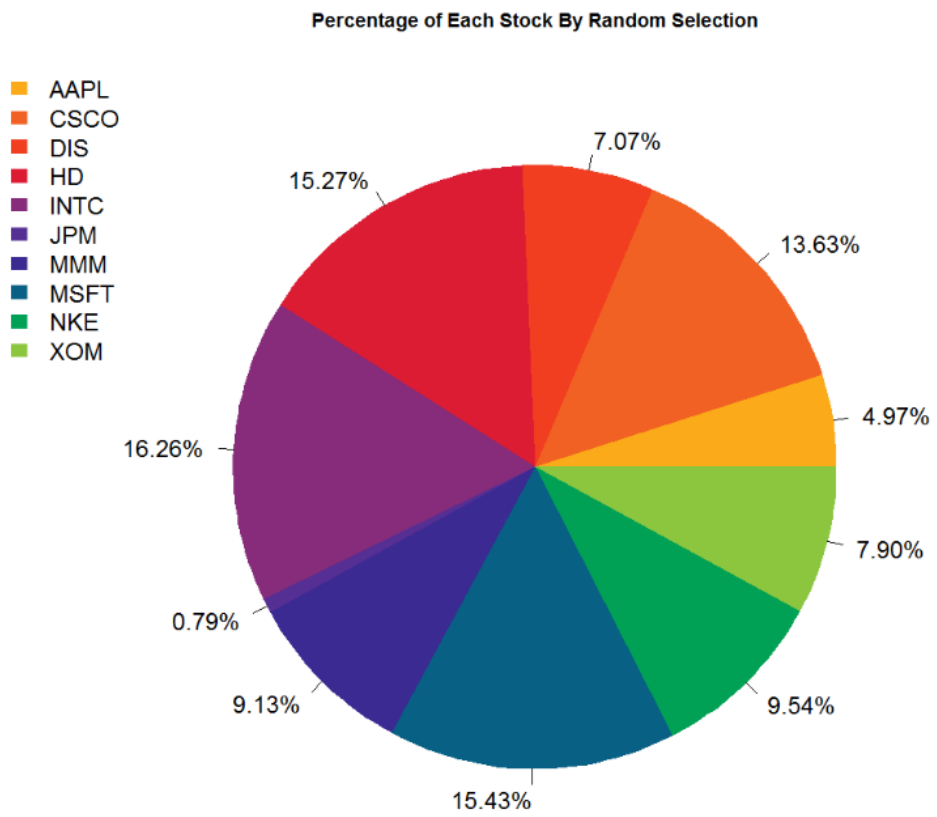
We would analysis this portfolio in the next part.

Experimental Analysis

To evaluate how good (bad) is the portfolio built by genetic algorithm selection, we compare the 3 portfolio, they are built by random selection, equal weight and GA selection:

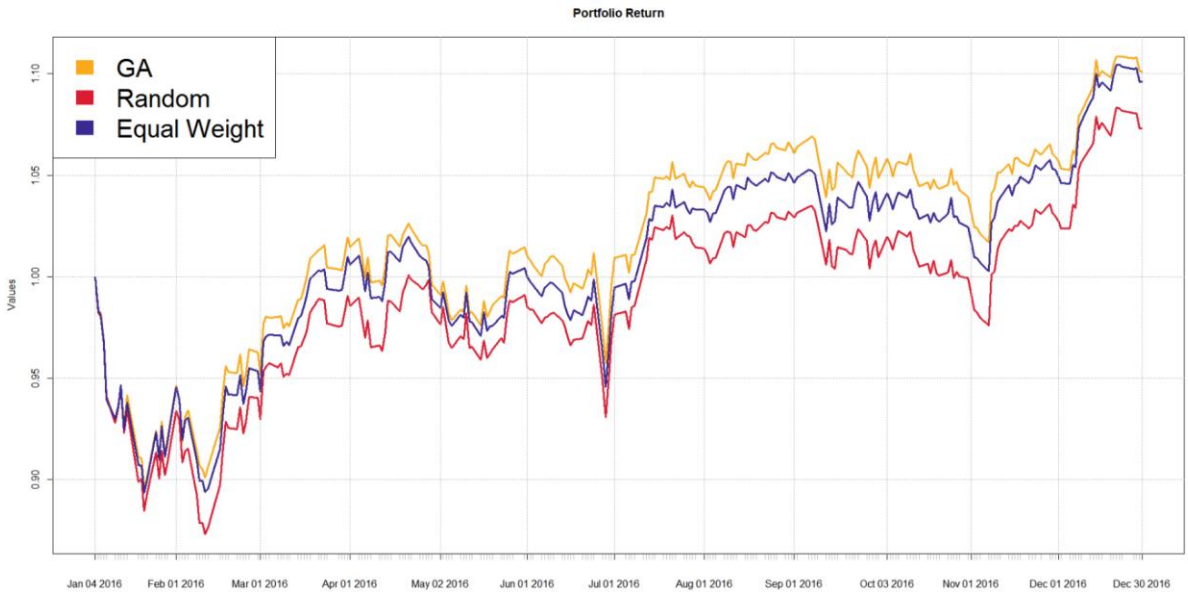

Percentage of Each Stock By GA Selection
AAPL
CSCO
DIS
HD
INTC
JPM
MMM
MSFT
NKE
XOM
5.17%
14.77%
4.27%
9.51%
10.93%
9.64%
15.76%
9.20%
8.54%
12.20%


Percentage of Each Stock By Even Split
AAPL
CSCO
DIS
HD
INTC
JPM
MMM
MSFT
NKE
XOM
10.00%
10.00%
10.00%
10.00%
10.00%
10.00%
10.00%
10.00%
10.00%



From the table below, we can see the performance of each portfolios. In the training data, the genetic algorithm returns best Sharpe ratio, and lowest risk. In the testing data, the GA portfolio still get the best sharpe ratio, but the equal weight portfolio has the lowest risk. Random selection portfolio performs the worst in both training and testing data. Random selection is all about luck. It is possible another random selection of portfolio can return a great sharpe ratio. If an investor prefer to bet on this kind of selection, in English, probably this behavior should be called gambling, not investing. It is not a smart choice.

	2010 - 2015		2016	
	Sharpe	Risk	Sharpe	Risk
GA	6.708	0.000159	0.647	0.001250
Even	4.813	0.000607	0.390	0.001090
Random	3.418	0.000959	-0.988	0.001386



From the portfolio return graph above, it shows that the GA portfolio has the highest return. So, high Sharpe ratio + high return, every investor would love it.

Conclusion

In this project, we discuss what is genetic selection, what is genetic algorithm and how to use genetic algorithm in Python and R. In summary, we think GA has some advanteges and disadvanteges:

Pro:

1. Genetic algorithm can help to find optimized result with much better solution than random selection.
2. Genetic algorithm requires less runtime than greedy algorithm (test all possible combination).
3. Genetic algorithm can be used in many kinds of optimized result searching problem, such as parameter tuning for another models, solution searching for some mathematical problem.

Con

1. It is not very easy to learn how to use the algorithm. Unlike the popular algorithms such as decision tree and KNN, GA not requires suitable parameters setting up, both also fitness function designing. And for some complex problem, the fitness function is not easy to design.
2. To find optimal solution for complex high dimensional problem, run time of GA could be still very long though it is better than greedy algorithm.
3. When GA is used to find the optimized result, it does not search all of the possible results. It only widens its searching area by crossover and mutation. So, GA can not guarantee to return the exactly best solution. For most of time, it only return a relative best solution which is very close to the true best solution.

By knowing the pros and cons, researchers can have more overall idea to decide whether to use GA or not in different situation.

From the case study, we confirm that Genetic algorithm is able to find a good portfolio with high sharpe ratio. No doubt, much better than random selection. On the other hand, genetic algorithm can not 100% guarantee the portfolio always keep the same sharpe ratio in the future as it is now. So, it could be a tool, but, should not be the only one tool to decide the weights of a portfolio. Probably investors should also consider some other fundamental indicators of each stock, such as PEG, PE.

In further study, we might design different kinds of risk adjustment in the fitness function, and test if there exists a risk adjustment which could return a portfolio better than we have gotten. What is more, we would also seek for other factors rather than historical Sharpe ratio to decide the weights of a portfolio, for example, fundamental factors (PE, PB, PEG) and Technical indicators (MACD, Bollinger Band).

Appendix

Case study codes:

```
#install.packages("devtools")
library(devtools)
dev_mode(on=T)
#install.packages("quantmod")     #if can't install the package , try install it from github (run line 5)
#install_github("joshuaulrich/quantmod")
library(quantmod)
library(timeSeries)

#download data
stockList <- c("AAPL","CSCO","DIS","HD","INTC","JPM","MMM","MSFT","NKE","XOM")
getSymbols(stockList, src = "yahoo")

#prepare historical stock data and charts
historyData <-
cbind(AAPL[,6],CSCO[,6],DIS[,6],HD[,6],INTC[,6],JPM[,6],MMM[,6],MSFT[,6],NKE[,6],XOM[,6])
plot(as.timeSeries(historyData), at = "chic", col
= .colorwheelPalette(10),lwd=2,xlab=c(1,2,3,4,5,6,7,8,9,0))
title("Historical Stock Price")

#prepare train data
returns.train <- lapply(stockList, function(s)
  monthlyReturn(eval(parse(text = s)),subset = "2010::2015"))
```

```
returns.train <- do.call(cbind,returns.train)
colnames(returns.train) <- stockList

#install.packages("timeSeries")
#plot stock returns of each month (train data)
library(timeSeries)
plot(as.timeSeries(returns.train), at = "chic", col = .colorwheelPalette(10),lwd=6,type="p")
title("Return Rate of Each Month")

n <- ncol(returns.train)
R.train <- colMeans(returns.train) # average monthly returns
R.train
c.train <- cov(returns.train) # monthly returns covariance matrix
s.train <- colSds(returns.train) # monthly returns Standard deviation
s.train

plot(s.train, R.train, type="n", panel.first = grid(),xlab = "Risk (Volatility of Monthly Returns)", ylab
= "Average Monthly Returns")
text(s.train, R.train, names(R.train), col = .colorwheelPalette(10), font = 2,cex=1.5)



#prepare testing data
#random selection data
set.seed(1234)
x <- runif(10)
w.random <- x/sum(x)
w.random

w.even = c(0.1,0.1,0.1,0.1,0.1,0.1,0.1,0.1,0.1,0.1)
w.even

returns.t <- lapply(stockList, function(s)
  monthlyReturn(eval(parse(text = s)),subset = "2016"))
returns.test <- do.call(cbind,returns.t)
colnames(returns.test) <- stockList
R.test <- colMeans(returns.test) # average monthly returns
c.test <- cov(returns.test) # covariance matrix of monthly returns
s.test <- sqrt(diag(c.test)) # volatility of monthly returns

#install.packages("ggplot2")
library(ggplot2)
```

```
percent <- function(x, digits = 2, format = "f") {
  paste0(formatC(100 * x, format = format, digits = digits), "%")
}

pie(w.random, labels = percent(round(w.random,digit=4)), main ="Percentage of Each Stock By Random Selection",col = .colorwheelPalette(10),lty=0,cex =1.5 )
legend("topleft", stockList, cex = 1.5, fill = .colorwheelPalette(10),border = "white",box.col="white")

pie(w.even, labels = percent(round(w.even,digit=4)), main ="Percentage of Each Stock By equal weight",col = .colorwheelPalette(10),lty=0,cex =1.5)
legend("topleft", stockList, cex = 1.5, fill = .colorwheelPalette(10),border = "white",box.col="white")



#what we need is a high sharpe ratio portfolio. So, use the GA to search the highest sharpe ratio portfolio
weights <- function(w) # normalised weights
{ drop(w/sum(w)) }

PortfolioReturn <- function(w,R) # Portfolio return
{ sum(weights(w)*R) }

sharpe <- function (w,R){
  (PortfolioReturn(w,R)-rfr)/sd(weights(w)*R)
}

p.risk <- function(w,R){
  sd(weights(w)*R)
}

#since we focus on monthly return, so we choose 1 month T-Bill yield today (May 28, 2017) as the risk free rate
rfr=0.00728

#We create a variable: risk adjustment. Because, we think the risk adjustment can help us to find
#a portfolio with more stable historical performance, which might also bring more stable future performance (more predictable performance).
#Since the average monthly return of the 10 stocks is 0.1, so, We choose 0.1^2*10% as a base line of risk.
#If the risk is more than 10%* 0.1^2 = 0.001, we reduce the sharpe ratio by risk*1000
#If the risk is more than 20%* 0.1^2 = 0.002, we reduce the sharpe ratio again, by (risk-0.002)*1000
```

```
#The more risk, the more punishment. So, if a portfolio has too much risk, unless its sharpe ratio is really high, it would not be chosen.
fitness <- function(w)
{
  riskAdjust=0
  if (p.risk(w,R.train)>0.001) riskAdjust=riskAdjust+(p.risk(w,R.train))*1000
  if (p.risk(w,R.train)>0.002) riskAdjust=riskAdjust+(p.risk(w,R.train)-0.002)*1000
  sharpe(w,R.train)- riskAdjust
}


#parameter tuning: 1. probability of mutation  2.probability of crossover  3.the generation

#Since we are trying to find the best solution of the fitness function (highest sharpe ratio).
#It is not what we normally did: model building + predicting.
#From the subset of the training data, it is impossible to find the best solution of fitness function for the whole training data set.
#Which means cross-validation validation is not appropriate for parameter tuning.
#It is better to write a function to do directly parameter tuning by comparing the GA@solution of different parameters.

#Before tuning the probability of mutation and probability of crossover, we need to think about the trade-off between runtime and accuracy.
#The more generation, the higher computation time. And of course, the result might be more close to best solution.
#We started by 4000 generation (maxiter = 4000), and default value for probability of mutation and probability of crossover

#install.packages("GA")
library("GA")
set.seed(1234)

GA.default <- ga(type = "real-valued", fitness = fitness,min = rep(0,n), max = rep(1,n), names = stockList,maxiter = 4000)
summary(GA.default)
plot(GA.default)

#we can see that, when generation is more than around 1000, the result of the algorithm only improved less than (6.8-6.5)/6.5 = 4.6%.
#But the computation time increased 4000/1000 -1 = 300%
#The result from 1000 generation, which is around 6.6 sharpe ratio (after risk adjustment) is already great enough.
```

#Considering both the runtime and accuracy, we would use 1000 as the value of maxiter(generation).

#parameter tuning: 1. probability of mutation 2.probability of crossover

#Mostly, researchers use crossover rate on level 0.7-0.9 and mutation on 0.05-0.2 ("Parameter tuning for configuring and analyzing evolutionary algorithms",A.E. Eiben, S.K. Smit1)

#So, we would like to search the best value in these 2 ranges.

```
pcrossover = 0.7#0.7 to 9;  increased 0.01 each time
pmutation = 0.05 #0.05 to 0.2; increased 0.01 each time
bestFit=0
pc=0
pm=0

while (pcrossover <=0.9)
{
  while (pmutation <=0.2){
    set.seed(1234)
    GA <- ga(type = "real-valued", fitness = fitness,min = rep(0,n), max = rep(1,n), names = stockList, pcrossover = pcrossover, pmutation =pmutation,maxiter = 1000)
    pmutation = pmutation + 0.01
    if (GA@fitnessValue > bestFit) {
      bestFit = GA@fitnessValue
      pc=pcrossover
      pm=pmutation
    }
  }
  pcrossover = pcrossover + 0.01
}

pc
pm

set.seed(1234)
GA <- ga(type = "real-valued", fitness = fitness,min = rep(0,n), max = rep(1,n), names = stockList,pcrossover = pc, pmutation =pm,maxiter = 1000)
summary(GA)
plot(GA)

w <- weights(GA@solution)
w
```

```
#plot weights of GA portfolio
pie(w, labels = percent(round(w,digit=4)), main ="Percentage of Each Stock By GA Selection",col = .colorwheelPalette(10),lty=0, cex = 1.5)
legend("topleft", stockList, cex = 1.5, fill = .colorwheelPalette(10),border = "white",box.col="white")

#portfolio performance: Sharpe and risk
performances<- function(R){
  port.per.tr = cbind(sharpe(w.even,R),sharpe(w.random,R),sharpe(w,R))
  colnames(port.per.tr) <- c("even.Sharpe","random.Sharpe","GA.Sharpe")
  print(port.per.tr)

  port.rsk.tr = cbind(p.risk(w.even,R),p.risk(w.random,R),p.risk(w,R))
  colnames(port.rsk.tr) <- c("even.rsk","random.rsk","GA.rsk")
  print(port.rsk.tr)
}
performances(R.train)
performances(R.test)



#Plot 3 portfolio return
daily2016 <- lapply(stockList, function(s)
  dailyReturn(eval(parse(text = s)),subset = "2016"))
daily2016 <- do.call(cbind,daily2016)
colnames(daily2016) <- stockList

daily2016$port=1
daily2016$port.random=1
daily2016$port.even=1
daily2016[,1:10]=daily2016[,1:10]+1

for(i in 2:nrow(daily2016)){
  daily2016$port[i]=daily2016$port[i-1]*sum(w*as.numeric(daily2016[i-1,1:10]))
  daily2016$port.random[i]=daily2016$port.random[i-1]*sum(w.random*as.numeric(daily2016[i-1,1:10]))
  daily2016$port.even[i]=daily2016$port.even[i-1]*sum(w.even*as.numeric(daily2016[i-1,1:10]))
}

plot(as.timeSeries(daily2016[,11:13]), plot.type="single",at = "chic", col = .colorwheelPalette(4),lwd=3)
title("Portfolio Return")
```

```
legend("topleft", c("GA","Random","Equal Weight"), cex = 2.5, fill = .colorwheelPalette(4),border
= "white",bg= "white")


dev_mode(on=F)
```