# Portfolio Optimisation using GAs

Shan Kuang, ID 202381266

2024-02-24

# Part 1: Optimization of the portfolio using GA

# 1: Getting data

Total 10 stocks were select from different sectors. We've choose from sectors in technology, entertainment, game and retail.By carefully selecting stocks from these sectors, we aim to construct portfolios that's brings their own return and risks, allowing us to find a financial potential performance.

```
symbols <- c("AAPL", "MSFT", "NFLX", "GOOG", "AMZN", "TSLA", "CCOEF", "NTDOY","DIS","META")
getSymbols(symbols, src = 'yahoo', from = '2014-01-01', to = '2018-12-31')
```

```
##  [1] "AAPL"  "MSFT"  "NFLX"  "GOOG"  "AMZN"  "TSLA"  "CCOEF" "NTDOY" "DIS"
## [10] "META"
```

```
prices <- do.call(merge, lapply(symbols, function(sym) Cl(get(sym))))
returns <- na.omit(Return.calculate(prices))
```

# 2: Splitting the Data into Train and Test set

As we need to have separate training data to optimize the model and testing data to evaluate its performance, we split the date into Train and Test data here. Since the whole data including a period from 01/01/2014 to 31/12/2018, we use the year of 2018 as our test data which count for approximately 25%.

```
# Split the data into train and test
# Find the last date no later than December 31, 2017
last_date_in_2017 <- max(index(returns)[index(returns) <= as.Date('2017-12-31')])
# Computed split point
index_split <- which(index(returns) == last_date_in_2017)
returns_train <- returns[1:index_split, ]
returns_test <- returns[(index_split+1):nrow(returns), ]
```

To find the best weights we are going to use in our financial investments, we are more focusing on gaining the return while balancing the risk. Thus our fitness function is defined to calculate the "Sharp Ratio". A sharp ratio is the return of the portfolio divided by the portfolio risk, which is a measure of return per unit of risk.

Since the genetic algorithm minimizes the fitness function, we make the Sharpe ratio negative so that the higher Sharpe ratio (better) is treated as the lower value minimized by the genetic algorithm.

```
# Define Fitness Function-sharp ratio
fitness_function <- function(weights, data) {
  returns <- data %*% weights
  annualized_return <- prod(1 + returns)^(252/nrow(returns)) - 1
  annualized_sd <- sd(returns) * sqrt(252)
  sharpe_ratio <- annualized_return / annualized_sd
  return(-sharpe_ratio) # Take a negative value because GA is maximizing fitness
}
```

```
# Setting GA parameters
ga_settings <- list(
   type = "real-valued",
   popSize = 50,
   maxiter = 150,
   run = 20
)
```

# 3: Run the GA Function and get the Optimal Weights

Here we plot the result and get a graph to visualize the performance of our genetic algorithm over generations.

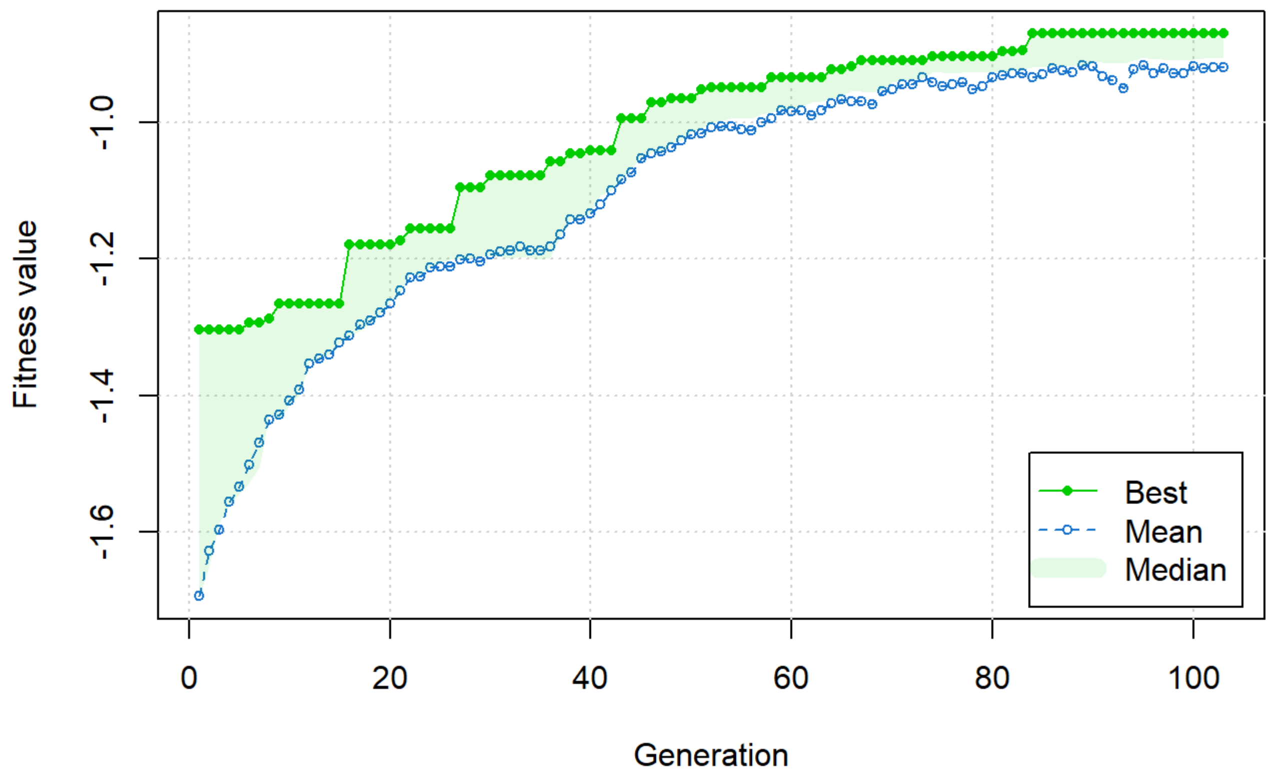
From the graph we can clearly see that our solution is improving over generations especially in the first 20. Around 140th generation, our curve become flattened which indicate there is a good solution found to the problem. Therefore, running more generation may not be helpful for a improvement.

```
# Run the GA function
ga_result <- ga(
  type = ga_settings$type,
  fitness = function(weights) fitness_function(weights, returns_train),
  lower = rep(0, length(symbols)),
  upper = rep(1, length(symbols)),
  popSize = ga_settings$popSize,
  maxiter = ga_settings$maxiter,
  run = ga_settings$run
)
```

```
# Check the result on training set
summary(ga_result)
```

```
## ── Genetic Algorithm ─────────────────────────────────────
##
## GA settings:
## Type                  =  real-valued
## Population size       =  50
## Number of generations =  150
## Elitism               =  2
## Crossover probability =  0.8
## Mutation probability  =  0.1
## Search domain =
##       x1 x2 x3 x4 x5 x6 x7 x8 x9 x10
## lower  0  0  0  0  0  0  0  0  0   0
## upper  1  1  1  1  1  1  1  1  1   1
##
## GA results:
## Iterations             = 103
## Fitness function value = -0.8705629
## Solution =
##              x1        x2        x3        x4         x5        x6         x7
## [1,] 0.03078954 0.1546878 0.0545761 0.1639123 0.06496403 0.9515192 0.03107603
##              x8        x9        x10
## [1,] 0.08487819 0.6919967 0.02789175
```

```
plot(ga_result)
```




```
# find the optimal weights
optimal_weights_train <- ga_result@solution
total_weight <- sum(optimal_weights_train)
normalized_optimal_weights_train <- optimal_weights_train / total_weight
print(normalized_optimal_weights_train)
```

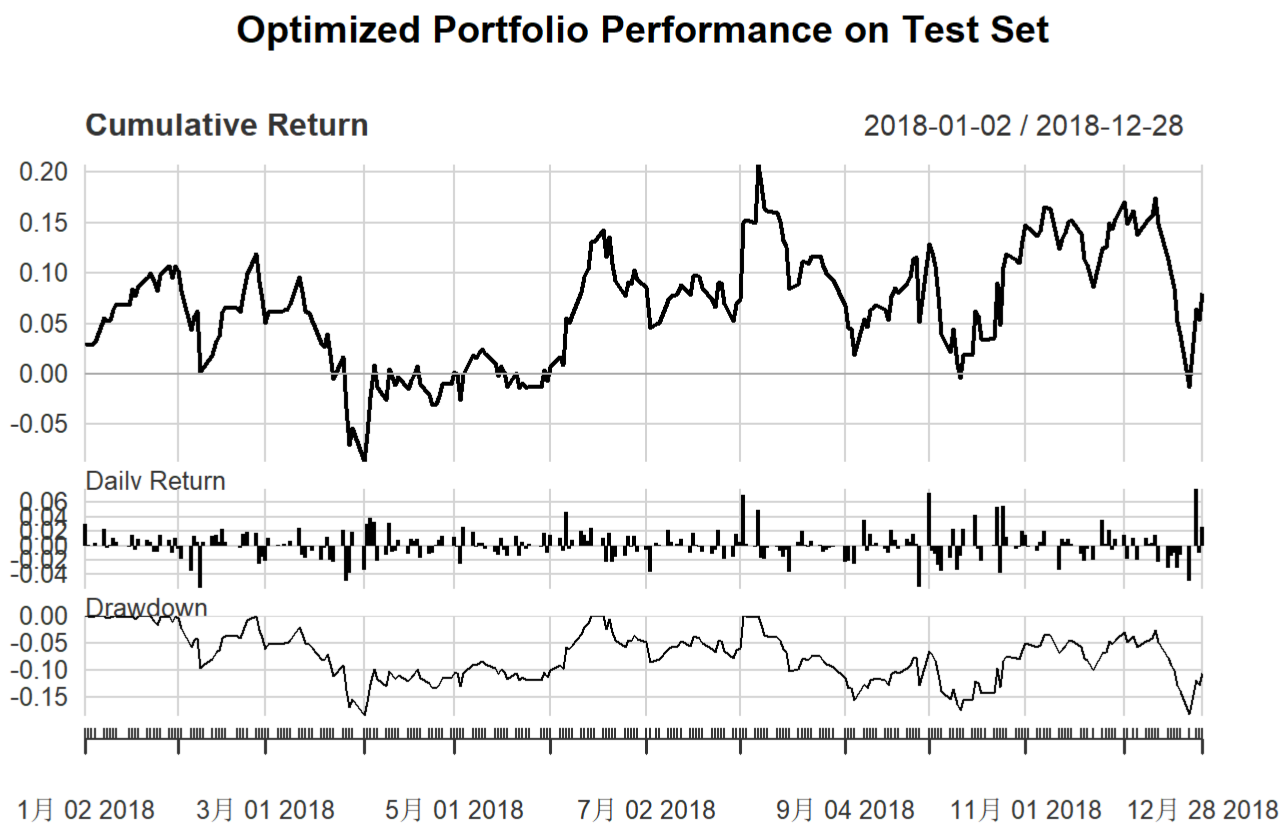
```
##              x1         x2        x3         x4         x5        x6         x7
## [1,] 0.01364608 0.06855844 0.0241884 0.07264677 0.02879239 0.4217182 0.01377306
##              x8        x9       x10
## [1,] 0.03761845 0.3066965 0.01236177
```

# 4.Create a Summary of the Portfolio’s Performance

To get the portfolio returns over the test period, we use matrix multiplication based on return in test and the optimal weights we got from GAs.It’s a way to simulate how the investment strategy would have worked in real-life conditions not seen during the optimization process which we called as “future”.

```
# make sure "returns_test" is matrix
returns_test_matrix <- as.matrix(returns_test)
# make sure "normalized_optimal_weights_train" is a vector
optimal_weights_vector <- as.numeric(normalized_optimal_weights_train)

# Matrix multiplication, The optimal weights on the training set are used to calculate the performance on
the test set
test_portfolio_returns <- returns_test_matrix %*% optimal_weights_vector
performance_test <- PerformanceAnalytics::charts.PerformanceSummary(test_portfolio_returns, main="Optimize
d Portfolio Performance on Test Set")
```




# 5. Calculate the Cumulative Returns on Possible Portfolios

To evaluate the performance of optimized portfolio, we need to compare it to other possible portfolios.

- Balanced Portfolio is a basic benchmark which can help compare and evaluate the GA-optimized portfolio. It can help us easily understand if we get a better result form GA algorithm.

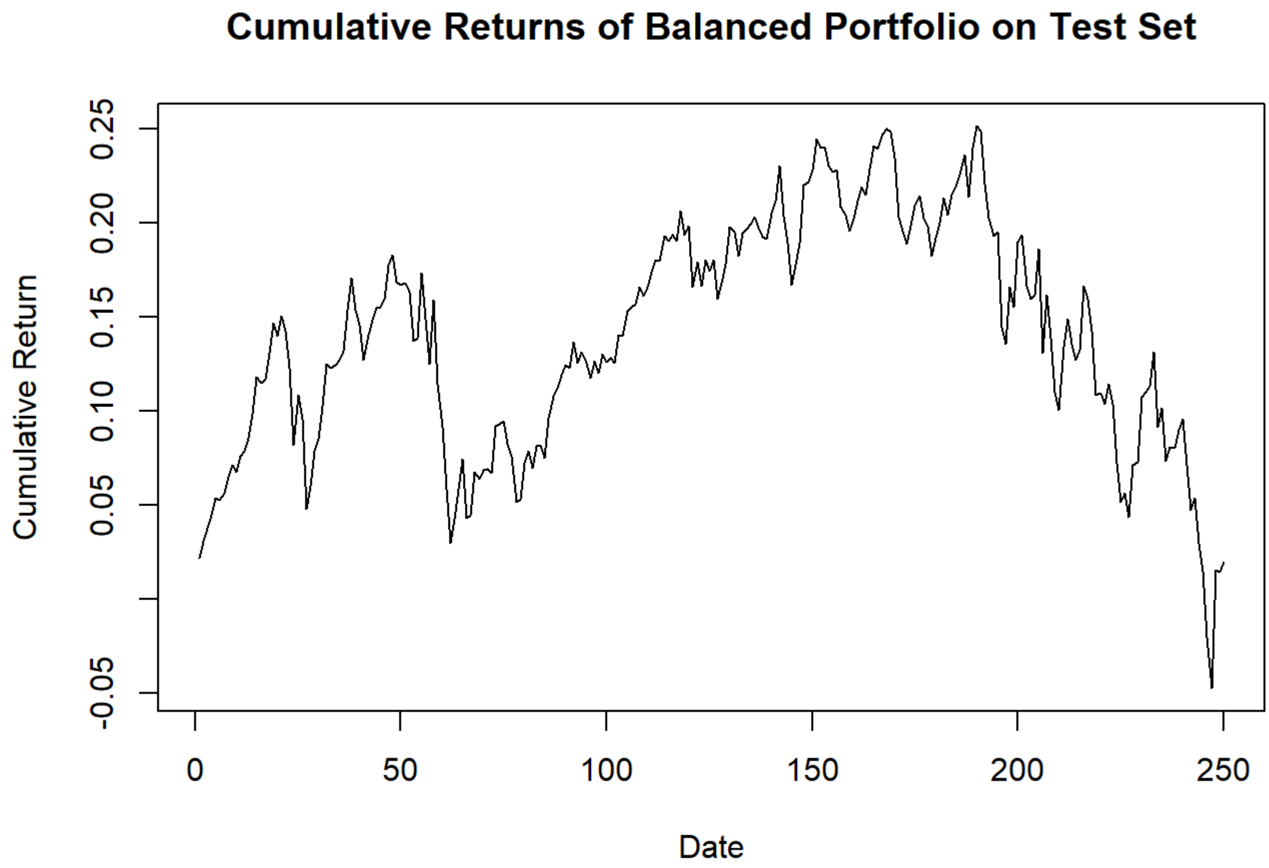
```
# Generate a balanced portfolio and calculate its performance
num_assets <- length(symbols)
balanced_weights <- rep(1/num_assets, num_assets)
print(balanced_weights)
```

```
##  [1] 0.1 0.1 0.1 0.1 0.1 0.1 0.1 0.1 0.1 0.1
```

```
# Calculate the daily return on a balanced portfolio
balanced_returns_test <- returns_test %*% balanced_weights
```

```
# Calculating cumulative rate of return
cumulative_balanced_returns_test <- cumprod(1 + balanced_returns_test) - 1
```

```
# Plot cumulative returns
plot(cumulative_balanced_returns_test, type = 'l', main = "Cumulative Returns of Balanced Portfolio on Tes
t Set", ylab = "Cumulative Return", xlab = "Date")
```




- Random Portfolio

```
# Generate and evaluate the performance of a random portfolio
set.seed(123) # get a random seed and ensure that random generation is repeatable
random_weights <- replicate(1000, runif(length(symbols)))
random_weights <- random_weights / rowSums(random_weights)
```

```
# Calculate the cumulative return for each randomly weighted portfolio
random_returns_test <- sapply(1:ncol(random_weights), function(i) {
  daily_returns <- returns_test %*% random_weights[,i]  # get daily return
  cumprod(1 + daily_returns) - 1  # Calculating cumulative rate of return
})
```

# 6.Compare the Performance of a Random, a Balanced, and Optimized Portfolios

Since the plot for random portfolio is not well structured, we only calculate the Sharpe ratio for compares.

Based on the definition of the Sharpe ratio, the random portfolio obviously has the highest Sharpe ratio surprisingly, reaching 47.06003, which means that a higher excess return has been obtained for the unit risk. So, from a Sharpe ratio perspective, a random portfolio looks like the best option.

However, after searching online for this issue, we found that is important to note that the Sharpe ratio for a random portfolio may not be realistic, as it is simply a portfolio generated based on random weights. The sharpe ratio of the optimized portfolio and the balanced portfolio reflects the performance of the portfolio generated according to a certain optimization strategy or balance principle.

**So, based on current selected symbols and time period (2014-2018), a optimized portfolio may be better in the future (after 2018).**

```
# check the cumulate return on random portfolio
head(random_returns_test[,1])
```

```
## [1] 0.0002269698 0.0003660890 0.0004791326 0.0005451125 0.0005900361
## [6] 0.0005987589
```

```
random_performance_test <- apply(random_returns_test, 2, function(x) {
  x <- na.omit(x)
  annualized_return <- prod(1 + x)^(252/length(x)) - 1
  annualized_sd <- sd(x) * sqrt(252)
  sharpe_ratio <- annualized_return / annualized_sd
  return(sharpe_ratio)
})
mean_random_sharpe <- mean(na.omit(random_performance_test))
cat("Mean Sharpe Ratio for Random Portfolios:", mean_random_sharpe, "\n")
```

```
## Mean Sharpe Ratio for Random Portfolios: 47.06003
```

```
# The sharp ratio for optimized portfolio
sharpe_optimized <- -fitness_function(optimal_weights_vector, returns_test_matrix)  # 使用之前确保维度匹配的变量
cat("Sharpe Ratio for Optimized Portfolio:", sharpe_optimized, "\n")
```

```
## Sharpe Ratio for Optimized Portfolio: 0.2580638
```

```
# The sharp ratio for balanced portfolio
sharpe_balanced <- -fitness_function(balanced_weights, returns_test_matrix)
cat("Sharpe Ratio for Balanced Portfolio:", sharpe_balanced, "\n")
```

```
## Sharpe Ratio for Balanced Portfolio: 0.08417876
```

# Part 2: Extending the Solution to Asset Selection

```
large_asset_pool <- c("AAPL", "MSFT", "GOOG", "AMZN", "TSLA",
                      "META", "BABA", "NVDA", "NFLX", "ADBE",
                      "BAC", "KO", "DIS", "PEP", "CMCSA",
                      "INTC", "CSCO", "WMT", "VZ", "T",
                      "JNJ", "PFE", "MRK", "UNH", "ABBV",
                      "CVS", "MDT", "JPM", "BMY", "WFC",
                      "MA", "V", "PYPL", "AXP", "GS",
                      "HD", "LOW", "HD", "MCD", "NKE",
                      "SBUX", "AAP", "AMGN", "GILD", "DIS",
                      "COST", "TGT", "WBA", "MET")

getSymbols(large_asset_pool, src = 'yahoo', from = '2014-01-01', to = '2017-12-31')
```

```
##  [1] "AAPL"  "MSFT"  "GOOG"  "AMZN"  "TSLA"  "META"  "BABA"  "NVDA"  "NFLX"
## [10] "ADBE"  "BAC"   "KO"    "DIS"   "PEP"   "CMCSA" "INTC"  "CSCO"  "WMT"
## [19] "VZ"    "T"     "JNJ"   "PFE"   "MRK"   "UNH"   "ABBV"  "CVS"   "MDT"
## [28] "JPM"   "BMY"   "WFC"   "MA"    "V"     "PYPL"  "AXP"   "GS"    "HD"
## [37] "LOW"   "MCD"   "NKE"   "SBUX"  "AAP"   "AMGN"  "GILD"  "COST"  "TGT"
## [46] "WBA"   "MET"
```

```
prices <- do.call(merge, lapply(large_asset_pool, function(sym) Cl(get(sym))))
returns <- na.omit(Return.calculate(prices))
```

# 1: Calculate the Fitness of a Portfolio Based on the Selected Assets

Here we defined a new function which can help us calculated the fitness rate of our portfolio.

```
# Fitness Function for choosing the assets
asset_selection_fitness <- function(selected, data) {
  if(sum(selected) == 0) return(-Inf)  # If no asset is selected, a very low fitness value is returned

  # Consider only the selected assets
  filtered_data <- data[, selected == 1, drop = FALSE]
  portfolio_returns <- rowMeans(filtered_data)  # Assume equal weight
  annualized_return <- prod(1 + portfolio_returns)^(252/nrow(filtered_data)) - 1
  annualized_sd <- sd(portfolio_returns) * sqrt(252)
  sharpe_ratio <- annualized_return / annualized_sd

  return(sharpe_ratio)  # maximize sharp_ratio meets the general norms of investment performance indicato
rs
}
```

# 2: Get the Selected Assets and Calculate Their Weights using GA

A new GA function “ga_asset_selection” call that selects an asset from the Large pool of available options using a defined fitness function. It uses a binary representation for the selection process, where each bit represents the selection state of the asset (selected or not selected).

As we adjust the parameter of GA, the number of selected assets will change significantly. A number set of parameter has been tested, here we use the best parameter for this pool. Which give us 8 assets in selection.

```
# Using GA to select Assets
ga_asset_selection <- ga(
  type = "binary",  # use binary to represent for the selection process
  fitness = function(selected) asset_selection_fitness(selected, returns),
  nBits = length(large_asset_pool),  # number of available assets for selection is equals to column in "la
rge_asset_pool"
  popSize = 50,
  maxiter = 100,
  run = 20
)
```

```
# Gets the index of the selected asset
selected_assets <- which(ga_asset_selection@solution == 1)

print(selected_assets)
```

```
## [1]  4  7  8 10 18 38 39 46
```

```
selected_symbols <- large_asset_pool[selected_assets]

print(selected_symbols)
```

```
## [1] "AMZN" "BABA" "NVDA" "ADBE" "WMT"  "HD"   "MCD"  "COST"
```

```
fitness_function_selected_assets <- function(weights, data) {
  # Consider only selected assets
  selected_data <- data[, selected_assets, drop = FALSE]
  portfolio_returns <- selected_data %*% weights
  annualized_return <- prod(1 + portfolio_returns)^(252/nrow(selected_data)) - 1
  annualized_sd <- sd(portfolio_returns) * sqrt(252)
  sharpe_ratio <- annualized_return / annualized_sd

  return(-sharpe_ratio)  # Take a negative value because GA is maximizing fitness
}

# Use the optimized weight of the selected asset
ga_optimize_weights <- ga(
  type = "real-valued",
  fitness = function(weights) fitness_function_selected_assets(weights, returns),
  lower = rep(0, length(selected_assets)),
  upper = rep(1, length(selected_assets)),
  popSize = 50,
  maxiter = 150,
  run = 20
)
```

# 3. The Optimal Weights for Selected Assets

```
optimal_weights_large <- ga_optimize_weights@solution
total_weight_large <- sum(optimal_weights_large)
normalized_weights_large <- optimal_weights_large / total_weight_large
print(normalized_weights_large)
```

```
##              x1         x2         x3          x4        x5         x6
## [1,] 0.02865855 0.02697792 0.01978449 0.006980449 0.3809499 0.06912844
##              x7       x8
## [1,] 0.04634325 0.421177
```