Genetic Algorithm Report

Elizabeth Tang (A0107689N)

Saturday, November 01, 2014

Problem Statement:

To find the best allocation of stocks for a set Target Investment Amount, from results after testing the prediction models that we have trained.

Argue for Genetic Algorithm:

With the many stocks chosen by the models, we need to chose - how many of each stock to buy within an allocated amount of money. - for diversification, we need to spread the money equally within different sectors. - to buy more of the stock if the combined test probability given by the models is higher.

With Genetic Algorithm, we can set these as contraints to get the best stock allocation.

Data we are using:

```
## Loading required package: foreach
## Loading required package: iterators
## Package 'GA' version 2.2
## Type 'citation("GA")' for citing this R package in publications.
```

Using a sample of testing results for testDate of 2009-12-01, and sectors: Consumer Discretionary,Financials

Columns from testing results:

```
str(data)
```

```
## 'data.frame':    288 obs. of  25 variables:
##  $ X.1              : int  1 2 3 4 5 6 7 8 9 10 ...
##  $ SecId            : int  7 21 23 24 25 27 34 35 38 45 ...
##  $ X                : int  1 2 3 4 5 6 7 8 9 10 ...
##  $ Date             : Factor w/ 1 level "2009-12-01": 1 1 1 1 1 1 1 1 1 1 ...
##  $ diff20           : num  1.27 2.78 6.74 8.97 2.03 ...
##  $ bin.diff20       : int  0 0 0 1 0 1 0 1 0 1 ...
##  $ ksvm             : int  0 0 0 0 0 0 0 0 0 0 ...
##  $ nnet             : int  1 0 0 0 0 1 0 0 0 0 ...
##  $ rf               : int  0 0 1 0 0 0 0 0 0 0 ...
##  $ ada              : int  0 0 0 0 0 0 0 0 0 0 ...
##  $ ksvm.prob.0      : num  0.909 0.904 0.485 0.865 0.842 ...
##  $ ksvm.prob.1      : num  0.0915 0.0957 0.5153 0.135 0.1583 ...
##  $ nnet.prob        : num  9.99e-01 0.00 1.18e-06 0.00 0.00 ...
##  $ rf.prob.0        : num  0.786 0.786 0.444 0.678 0.658 0.664 0.856 0.858 0.564 0.846 ...
##  $ rf.prob.1        : num  0.214 0.214 0.556 0.322 0.342 0.336 0.144 0.142 0.436 0.154 ...
##  $ ada.prob.1       : num  0.983 0.999 0.863 0.992 0.978 ...
##  $ ada.prob.2       : num  0.01679 0.00116 0.13724 0.00844 0.02236 ...
```

```
##  $ votes             : int  1 0 1 0 0 1 0 0 0 0 ...
##  $ voted             : int  0 0 0 0 0 0 0 0 0 0 ...
##  $ all.prob          : num  0.3304 0.0777 0.3021 0.1164 0.1307 ...
##  $ voted.prob        : int  0 0 0 0 0 0 0 0 0 0 ...
##  $ type              : Factor w/ 1 level "test": 1 1 1 1 1 1 1 1 1 1 ...
##  $ sector            : Factor w/ 2 levels "Consumer Discretionary",..: 2 2 2 2 2 2 2 2 2 2 ...
##  $ PriceToday        : num  49.8 46.4 30.8 13.9 30.7 ...
##  $ PriceForward20Days: num  50.3 47 31.7 16 29.7 ...
```

Therefore Data includes

- votes and probability from prediction models:
  - support vector machine
  - neural network with 100 hidden nodes
  - random forest
  - ada boost
- combined probability is mean of the 4 model probabilities.
- prices on the testDate. Note that the prediction was a binary classification, its purpose was to test whether the tested stocked was in the top 20% 20Day Forward Price Momentum.

Preparation of Data:

1. Filtering rows of results to those with more than 1 vote, where at least 1 model predicted a true.
2. Using only variables

- SecId
- votes
- all.prob
- sector
- PriceToday, remove others.

3. Order the stocks by their sector and combined probability, while experimenting, found that with this ordering, the solutions were general better.

```
data = data[data$voted.prob >0,]
data = data[, c("SecId", "votes", "all.prob", "sector", "PriceToday")]
data = data[order(data$sector, data$all.prob),]
```

There are 32 rows which will be passed for genetic algorithm processing.

Genetic Algorithm

Chromosome Representation

As the R package "GA" is used. There are 3 types of genetic algorithm that can be run:

- binary
- real-valued <- floating points
- permutation <- "for problems that involves reordering of a list"

Since this problem is about stock allocation, we are trying to find the number to buy for each stock. This number must be an integer and not a floating point. Also, this is not a permutation problem.

Therefore we are only left with the "binary" type.

```
numStocks = nrow(data)
bitsPerStock = 5
nBits = bitsPerStock*numStocks
```

Assuming we put the number of bits for each stock as 5, the maximum number that can be bought with each stock is 2^number of bits is 32. This can be changed but for simplication of the problem, shall set as 5 bits.

So the chromosome will be 5 * (number of stocks), 160 bits. This binary string has to be decoded into integers for the fitness function and for deciphering the solution.

```
decode <- function(string) {
  string <- gray2binary(string)
  startingPositions = rep(1,numStocks) + c(0:(numStocks-1) *bitsPerStock)
  noToBuy = as.integer()
  for (i in 1:numStocks ) {
    noToBuy = c(noToBuy,
                binary2decimal(string[startingPositions[i]:
                                        (startingPositions[i]+bitsPerStock-1)]))
  }
  return(noToBuy)
}
```

Objective

1. Total Proposed Spent on Stocks to be bought to reach the target Investment Amount.
2. The total Proposed amount spent on each sector should be the same.
3. As the combined test probability increases, the Proposed value spent on the stock should also increase.

Fitness Function

Using Soft constraints, where there will be increasing penalties will be imposed when constraints are not met.

1. For Objective 1:
   - abs(Target Investment Amount - Total Proposed Spent on Stocks)
2. For Objective 2:
   - abs(Total Proposed Spent on Sector 1 - Total Proposed Spent on Sector 2)
3. For Objective 3:
   - bin combined probability (3 bins, bin1 for lower probability)
   - limit bin1's proposed amount spent by an amount: targetTotalInvestAmount/numStocks * combined probability of the stock, only penalize if more than 0.
   - limit bin2's proposed amount spent by an amount: targetTotalInvestAmount/numStocks * combined probability of the stock * 2, only penalize if more than 0. bin2's limit has a *2 to make it higher.
   - bin1 and bin2 has an upper limit, bin3 does not.

Hard constraints are not used because it means stopping the GA run entirely and not only for a single iteration.

```
fitness <- function(string) {
  noToBuy <- decode(string)
  spent = noToBuy * data$PriceToday
  sumSpent = sum(spent)

  #penalize difference from targetTotalInvestAmount
  diffFromTarget = abs(targetTotalInvestAmount - sumSpent)

  #penalize difference spent on sectors
  sector1 = which(data$sector==sectors[1])
  sector2 = which(data$sector==sectors[2])
  diffInSectors = abs(sum(spent[sector1]) - sum(spent[sector2]))

  prob.bin = cut(data$all.prob,
                 c(0.5, 0.6, 0.8, 1),
                 labels=c(1,2,3), include.lowest=TRUE)

  # limit the amount spent on bin1 stocks
  diffSpentBin1 = spent[prob.bin == 1]-
                  (targetTotalInvestAmount/numStocks*data$all.prob[prob.bin == 1])
  diffSpentBin1[diffSpentBin1<0] = 0
  diffSpentBin1 = sum(diffSpentBin1)

  # limit the amount spent on bin2 stocks
```

```
  diffSpentBin2 = spent[prob.bin == 2]-
    (targetTotalInvestAmount/numStocks*data$all.prob[prob.bin == 2]*2)
  diffSpentBin2[diffSpentBin2<0] = 0
  diffSpentBin2 = sum(diffSpentBin2)

  return(-(diffFromTarget*10 + diffSpentBin1 + diffSpentBin2 + diffInSectors))
}
```

Operators

For the Default Binary Type,

- Population
  - initially randomly populated with nBits binary strings.
- Crossover
  - using single point cross over, where "one crossover point is selected, binary string from beginning of chromosome to the crossover point is copied from one parent, the rest is copied from the second parent". *Referenced from http://www.obitko.com/tutorials/genetic-algorithms/crossover-mutation.php.*
  - crossover is 0.8, 80%.
- Mutation
  - using Uniform random mutation, where "The mutation of bit strings ensue through bit flips at random positions." *Referenced from http://en.wikipedia.org/wiki/Mutation_(genetic_algorithm).*
  - mutation is 0.1, 10%.
- Selection
  - using Linear-rank selection, where "In linear ranking selection (Baker, 1985), first the individuals are ranked according to the fitness values. Those with high fitness values will be ranked high and those with low fitness values will eventually have lower ranks. Then the individuals are selected with a probability that is linearly proportional to the rank of the individuals in the population." *Referenced from A REVIEW OF SELECTION METHODS IN GENETIC ALGORITHM, by R.SIVARAJ.*

Run

```
targetTotalInvestAmount = 5000

GA <- ga(type="binary", fitness=fitness, nBits=nBits, popSize = 500,
         pmutation = 0.3, maxiter=200)
```

Evalution of Best Solution

```
noToBuy = decode(GA@solution)
prob = data$all.prob
spent = noToBuy * data$PriceToday
cbind(probability=round(prob,2), noToBuy, price = round(data$PriceToday,2), spent=round(spent,2))
```

```
##       probability noToBuy price  spent
##  [1,]        0.51       0 18.23   0.00
##  [2,]        0.52       4 28.28 113.12
##  [3,]        0.53       0  6.47   0.00
##  [4,]        0.54       1 13.69  13.69
##  [5,]        0.58       1 29.62  29.62
##  [6,]        0.59      10 14.11 141.10
##  [7,]        0.60       2 26.27  52.54
##  [8,]        0.62      11  9.85 108.35
##  [9,]        0.62       5 23.84 119.20
## [10,]        0.65       1 21.25  21.25
## [11,]        0.71       2 30.23  60.46
## [12,]        0.83      31 10.42 323.02
## [13,]        0.83      16 10.73 171.68
## [14,]        0.83      13 19.45 252.85
## [15,]        0.83      27  9.31 251.37
## [16,]        0.85       6 14.27  85.62
## [17,]        0.85      13 16.01 208.13
## [18,]        0.87      16 13.69 219.04
## [19,]        0.88      25 10.01 250.25
## [20,]        0.91       6 12.88  77.28
## [21,]        0.50       8 11.20  89.60
## [22,]        0.52      26  5.95 154.70
## [23,]        0.53      24  7.76 186.24
## [24,]        0.55       2 42.72  85.44
## [25,]        0.56      18 12.63 227.42
## [26,]        0.57       4 25.60 102.40
## [27,]        0.57       9 13.16 118.44
## [28,]        0.65      11 49.38 543.18
## [29,]        0.67      13 19.17 249.21
## [30,]        0.68      22 12.91 284.02
## [31,]        0.79      20  5.78 115.60
## [32,]        0.80      31 11.13 345.03
```

```
sum(spent)
```

```
## [1] 5000
```

The total Proposed amount to Invest is 4999.8515. <- 1st part of fitness function.

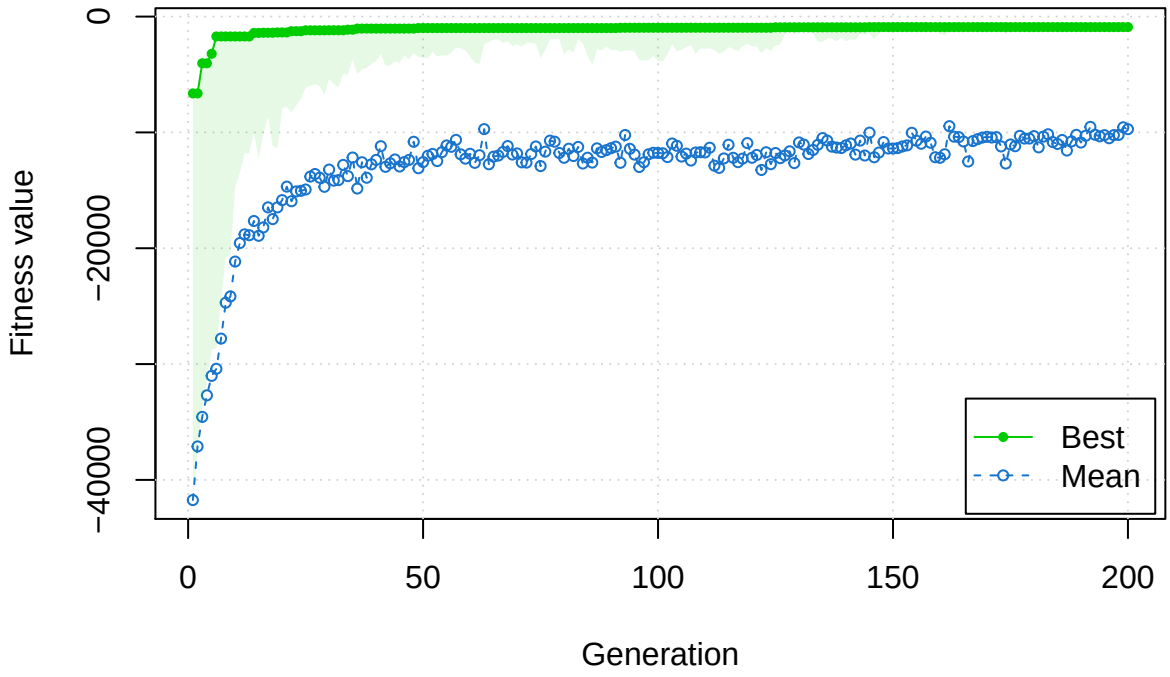
```
sector1 = which(data$sector==sectors[1])
sector2 = which(data$sector==sectors[2])
abs(sum(spent[sector1]) - sum(spent[sector2]))
```

```
## [1] 2.712
```

The difference in amount spent on the 2 sectors is 2.7115. <- 2nd part of fitness function.

Plot of the GA iterations:

```
plot(GA)
```



Plot of Combined Probability of stock vs Proposed Spent on Stock.

```
qplot(prob, spent, color=data$sector)
```

Shows the 3rd part of the fitness function. As probability increases, proposed amount spent generally increases. Though only some points with higher probability has higher proposed spent.

Conclusion

This Optimization Problem seems to have fulfilled the original objectives though the 3rd Object of letting higer probabilities have higher proposed spent is not very satisfactory. Will need to refine the fitness function for that for better accuracy.