STAT 243 Final Project Report

December 13, 2017

Eric Kim, Shuyu Zhao, Rui Chen, Hangyu Huang **Github name: erikim**

```
library(devtools)

## Warning:  package 'devtools' was built under R version 3.4.2

install_github("erickim/GA")

## Installation failed:  Timeout was reached:  Resolving timed out after 10000 milliseconds

library(GA)

##
## Attaching package:  'GA'
## The following object is masked from 'package:methods':
##
##    initialize
```

1 Part I: Functionality and Procedure

The Genetic algorithms (GAs) are stochastic search algorithms that mimic the process of Darwinian natural selection. GAs simulate the biological evolution, where breeding among highly fit organisms that ensures desirable attributes to be passed to future generations, thereby provides a set of increasingly good candidate solutions to the optimization. The select function enables the application of genetic algorithms to problems where the decision varibles are encoded as "binary".

Selection mechanism mimic the process by which parents are chosen to produce offspring. Crossover and mutation operator is used to produce offspring chromosomes from chosen parent chromosomes.

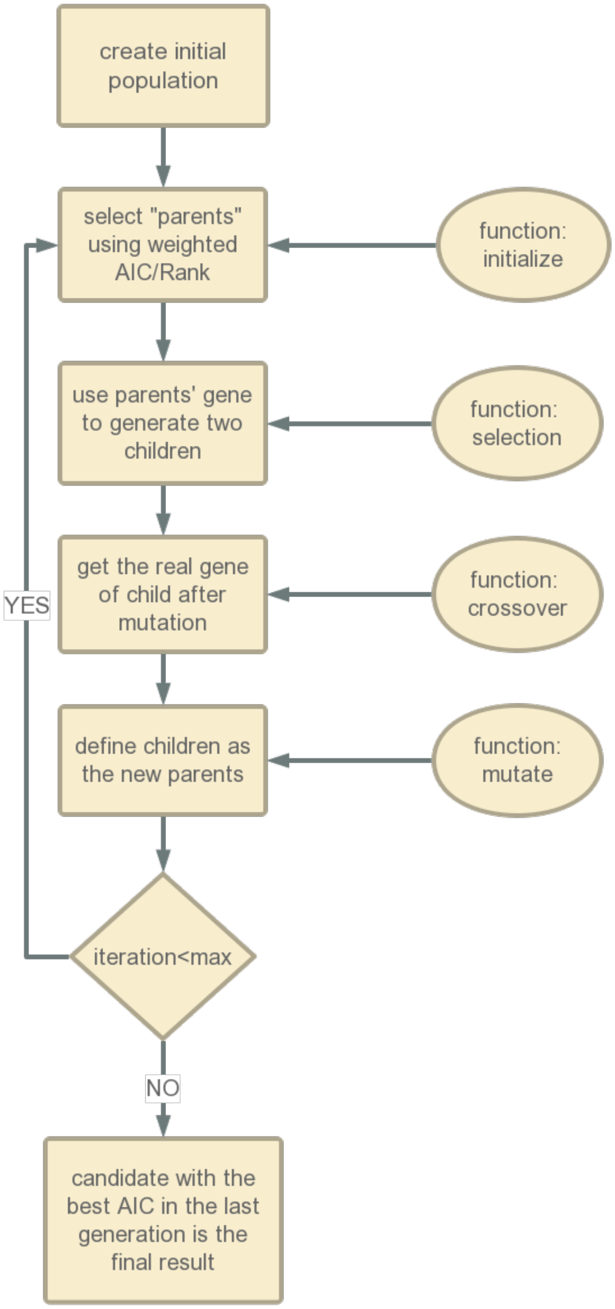
Rank-based method is applied here to prevent GAs convergence to a poor local optimum. Parents are chosen based on the rank of values of negetive AIC function. Any R function, which takes as input an individual string representing a potential solution, and returns a numerical value describing its "fitness", is allowable to perform as a fitness function.

The population size is in the range of the chromosome length to two times of chromosome length. In this function. the population size is twice of chromosome length, which is the number of columns of the feature matrix.

1.1 main function

The main function select() will initialize the first generation by calling initialize(), then select the individuals that serve as parents for the second generation by calling selection(). Then the parents produce the two individuals of the second generation by calling corssover() and finally generate mutation on the offsprings by calling mutate(). By then, the second generation is completely generated. Then we do iteration over the whole procedure above except for the initialization, because we treat the offsprings as the parents of the next generation. The final stopping condition we choose is based on the iteration number.Here we want the 100th generation to be our final solution.

Following is a graph demonstrating how this select function works.


create initial population
select "parents" using weighted AIC/Rank
function: initialize
use parents' gene to generate two children
function: selection
get the real gene of child after mutation
function: crossover
define children as the new parents
function: mutate
YES
iteration<max
NO
candidate with the best AIC in the last generation is the final result

parameter: Y ,dependent variable vector. Input should be a numeric vector
X ,covariate matrix. Input should be a numeric data frame or matrix
regType ,the type of regression, default is ”lm”. Input should be ”lm” and ”glm”
family ,the family of the glm, default is ”gaussion”. Input can be ”binomial”, ”poisson”, ”Gamma”,”quasi”, etc.
fitness ,the method to evaluate the fitness of the model, default is AIC.
rank ,indicates if we use rank as a selection criteria, default is TRUE. Input should be logical
selectionType, the selection criteria, default is ”twoprop”.

elitism, indicates if there is a generation gap, default is TRUE. Iuput should be logical.
crossoverType, type of cutting points defualt is ”sinle”. Input should be
either ”single” or ”multiple”
numCrossover, mumber of cutting points. Input should be an integer range from 1 to N-1
when type argument is ”multiple”, where N is the length of the parent vector.
mutationRate,the mutation rate for each variable. Input should be a numeric number between
0 and 1.
maxIter, the number of iterations needed, i.e. How many generations are being produced.
P, the number of individuals in each generation(i.e Population size), default is twice as
the number of covariates. Input should be integer.

output: The individual with best fitness among the final generation. We constructed this object with four fields: variable, a 0-1 vector that indicates which features to choose; fit, the model associated with this individual; fitness, the fitness value of this model; type, what kind of fitness criteria is being used.

1.2 individual functions

1.2.1 initialize

Funcion initialize() is used to create P initial parents for Genetic Algorithms. In this function, we created 0-1 vectors to indicate which variables are selected for each parent.

Input: Y (a vector), X (a matrix), P (2* ncol(X)), regType, family and seed.

Output: A list with two fields : variables (parent gene, a 0-1 vector indicating which variables are selected) and fit (fitting information of a parent gene).

1.2.2 selection

Function selection() is used to select parents from current generation so that they can produce offsprings. In this function, we provide two different methods of selection based on the probability of being chosen as a parent. The probability are calculated as the finess of the individuals devided by the sum of fitness value of the whole population. ”Oneprob” method randomly selects one parent based on the fitness probability, and selects second parent randomly at an equal chance from the remaining population. ”twoprob” method generates a pair of parents randomly based on the fitness probability.

Input: type; type of selection. arguments can be ”oneprob” or ”twoprob” only.
pop_fitness : the fitness vector for this generation. Input should be numeric vector with the
same length of the population vector.
output: return one pair of parents that will produce two offsprings later.

1.2.3 crossover

Function ”corssover” describes the simplified procedure of producing offsprings. It swaps the segments of parents’ chromosomes to generate two new children. We can have one cutting points, which means one chromosome is divided into two segments. We can also have multiple cutting points, which means one chromosomes is divided into more than two segments.

Input: parent1 and parent2: numeric vectors of 0s and 1s, which represent two parents.
type: type of cutting points. Input should be either ”single” or ”multiple”
num_splits: mumber of cutting points. Input should be an integer range from 1 to N-1 when
type argument is ”multiple”, where N is the length of the parent vector.
Output: two offspring vectors, which are 0-1 vectors.

1.2.4 mutate

Function ”mutate” is used to get the real child gene after mutation. In this function, we set the mutation rate and create a p-length 0-1 vector to indicate which variable will mutate, changing the gene in the corresponding location.

Input: rate ,the mutation rate for each variable

offspring ,child gene generated by crossover, is a 0-1 vector
Output: offspring, updated child gene, is a 0-1 vector

2 Part II: Testing

2.1 main function

To test whether our implementation gives correct results, we generated simulated dataset and compared the result given by GA function with the result given by regsubsets, which is a function for model selection.

```
library(devtools)
install_github("erickim/GA")

## Installation failed:  Timeout was reached:  Connection timed out after 10002 milliseconds

### TEST ###
set.seed(1)
Y <- rnorm(100)
X <- as.data.frame(matrix(rnorm(100*26), nrow = 100))
names(X) <- letters
P <- ncol(X)*2

### BEST SUBSETS VS GA WITH TEST ###
library(leaps)
testBest <- regsubsets(Y ~ ., data = X, nvmax = 26)
testBestWhich <- summary(testBest)$which
testBestAIC <- c()
for (i in 1:26) {
  testBestAIC <- c(testBestAIC,
                   AIC(glm(Y ~ .,
                           data = X[testBestWhich[i,-1]])))
}

testGASelect <- GA::select(Y, X, regType = "glm")

# optimized AIC
-1*testGASelect$fitness # GA

## [1] 262.3583

min(testBestAIC) # best subsets

## [1] 262.3583

# fit (same)
which(testGASelect$variables == 1)

## [1] 16 19 22

which(testBestWhich[which.min(testBestAIC),-1])

##  p  s  v
## 16 19 22
```

The optimized AIC of GA select function is the same as the best AIC of regsubsets function, and the two functions got the same selected variables. This test shows that our implementation gives correct results, the overall function passed the test.

2.2 modules

To test individual functions that carry out the individual computations that make up the algorithm, we mainly tested whether the inputs are vaild. Besides, we use simulated data to show the output of each function.

There are six functions needed to be tested: Initialize, crossover, mutate, selection, fitnessRank and regFun. We use est_that() to test these functions; use expect_error to test whether the function will get error if the input is invalid; use expect_equal, expect_false and expect_false to test whether the output of the function is reasonable. All functions passed tests.

3 Part III: Examples

In this part, we applied the GA select function to two examples, got the results and compared it to function ”regsubsets”, which is a function for model selection.

The results we got by two functions are the same.

The first example is about Swiss Fertility and Socioeconomic Indicators. The data frame is ”swiss” in r package ”dataset” and has 47 observations on 6 variables. We regressed Fertility on other aspects to find the influencing factors of Swiss Fertility.

```
#Swiss Fertility and Socioeconomic Indicators (1888) Data
Y<-swiss$Fertility
X<-swiss[2:6]

#use select function to select variables
GASelect<-GA::select(Y,X)

#use regsubsets to select variables
testBest <- regsubsets(Y ˜ ., data = X, nvmax = 5)
testBestWhich <- summary(testBest)$which
testBestAIC <- c()
for (i in 1:5) {
  testBestAIC <- c(testBestAIC,
                   AIC(glm(Y ˜ .,
                          data = X[testBestWhich[i,-1]])))
}

# compare the selected variables
which(GASelect$variables == 1)

## [1] 1 3 4 5

which(testBestWhich[which.min(testBestAIC),-1])

##      Agriculture        Education         Catholic Infant.Mortality
##                1                3                4                5
```

Both GA select function and regsubsets function select 4 varibles from 5 for the regression model, the selected variables are Agriculture, Education, Catholic and Infant.Mortality.

The second example is about crime data of Boston. The data frame is from ”Boston” in r package ”MASS” and has 506 rows and 14 columns. We regressed crim (per capita crime rate by town) on other aspects to find the influencing factors of crime rate.

```
library(MASS)

##
## Attaching package:  ’MASS’
## The following object is masked from ’package:GA’:
##
##     select

#Housing Values in Suburbs of Boston
Y<-Boston$crim
X<-Boston[2:14]

#use select function to select variables
GASelect<-GA::select(Y,X)

#use regsubsets to select variables
testBest <- regsubsets(Y ~ ., data = X, nvmax = 13)
testBestWhich <- summary(testBest)$which
testBestAIC <- c()
for (i in 1:13) {
  testBestAIC <- c(testBestAIC,
                   AIC(glm(Y ~ .,
                          data = X[testBestWhich[i,-1]])))
}

# compare the selected variables
which(GASelect$variables == 1)

## [1]  1  4  7  8 10 11 12 13

which(testBestWhich[which.min(testBestAIC),-1])

##      zn     nox     dis     rad ptratio   black   lstat    medv
##       1       4       7       8      10      11      12      13
```

Both GA select function and regsubsets function select 8 varibles from 13 for the regression model, the selected variables are zn, nox, dis, rad, ptratio, black, lstat and medv.

4 Part IV :contribution

Eric set up the package, wrote code, design the logic of the code, wrote the main select function, debugged and wrote test; Rui Chen wrote code, tested functions, debugged and wrote the Latex file; Shuyu Zhao wrote code, applied the function to examples, tested functions, debugged and wrote the Latex file; Hangyu Huang wrote code, tested functions, debugged and wrote the help page.