GA Solution

Zhuoran LIN, Zishan CHENG, Yu Hsuan LIN

12/19/2019

The `GA` package resides in 'zhuoranlin/GA' on github.com.

1. Approach used to build the algorithm

Primary function select() mainly contains 3 parts of functions/modularity. The function allows users to provide a formula for variable selection, corresponding dataset, user-defined fitness function, mutation rate, generation gap and family function for error in linear regression or GLMs.

The modularity are as follows:

1) Basic Settings and Preprossessing

Input Classes and Values: Assertions will make sure inputs are of the required class and values, as specified in details on the help page.

First Generation/Initialization: The function initializes genetic algorithm with a first generation of P = 2C random chromosomes, where C is the chromosome length, i.e. the number of independent variables provided in the formula. If there're factor variables, they will be converted to numeric variables first. The genes of chromosomes are from binary alphabets {0, 1}, where 1 stands for selected variables and 0 stands for truncated variables.

Generations and the Generation Gap: Every generation will be saved in a C by P matrix, where each column stands for one chromosome. The first generation is generated one by one at random, checking the duplicates every time a new chromosome is bred. The population will be partially updated with a generation gap G, which is either user-defined or 1 by default.

Fitness: The fitness of chromosomes are saved in a 100 by P matrix, where the i th row contains the fitness of P chromosomes in the i th generation, i = 1, ... 100.

Number of Iterations: Maximum number of iterations is set to be 100 as suggested by Givens and Hoteing.

2) Breeding: Parent choosing & Genetic operators (crossover and mutation built in)

Parent Choosing: The offsprings are produced by two parent chromosomes, one parent with probability proportional to fitness, and the other at random. The fitness function is by default AIC, but could also be user-defined. Here, the fitness rank is used to compute the probabilities of selecting the individuals for reproduction. Specifically, $probability = \frac{rank}{sum(rank)}$. This selection based on relative fitness prevents premature convergence or other problems introduced by the exact form of the fitness function.

Genetic Operators:

(a) **crossover()**: Here comes the auxiliary function of crossover(), which is defined within GA package. The crossover function takes two parents as input, and split both parent at the same random position between two adjacent loci. Then it produce one offspring by combining the left part of the first parent and the right part of the second parent.

(b) **mutation()**: After breeding, mutation() is applied. It's also an auxiliary function within the package. It takes in two parents, one offspring from the parents, and a mutation rate (m) between 0 and 1. If mutation rate is not defined by users, by default, theoretical work and empirical studies have supported a rate of 0.01. mutation() first find out the loci where both parents share some genes(so as the offspring), and then modify the genes of offspring at these loci to another value in {0, 1}, each with a probability of m.

Dealing with Duplicate Individuals: For every generation, select() will make sure there's no duplicates, so as not to waste computing resources or distort the parent selection criterion.

3) Termination and Interpretation from Chromosomes to Variables

The last part is to terminate breeding when maximum number of iteraions is reached, which is 100 by default if not define by users. After the process terminates, the chromosome that have the best fitness in the last generation will be interpreted into the final variables selected. The function will return a list of a) a vector of the final variables selected, b) its corresponding fitness, c) a C by P matrix that contains the chromosomes in the last generation, and d) the 100 by P fitness matrix that records the fitness of all chromosomes in the whole process.

2. Testing

The tests are set for the main function select(), and the two genetic operator crossover() and mutation().

1) Test for select()

The functionfirst checks to make sure the inputs are valid as required on the help page.

The test check if will return error and give error message when the inputs are wrong: a) formula is not of class *formula*, b) mutation rate is not a number between 0 and 1, c) generation gap is not a number between 1/P and 1, d) family is not valid, e) user-defined fitness is not a function that returns a numeric value.

Then, it check if the function works well with a valid user-defined fitness(r square in specific), general situations including factor variables, numeric independent variable with gaussian family, 0/1 independent variable with binomial family, integer independent variable with poisson family, and numeric independent variable with Gamma family.

2) Test for crossover()

The function first checks to make sure there's no missing inputs, the input parents are numeric vectors and they are of same length.

The test also checks if the function will work under a general situation, a situation where parents are identical, and a situation where the parents have only one gene.

Then, it checks if the function will return error when the inputs are missing or not numeric (character or logical in specific).

3) Test for mutation()

The function first checks to make sure there's no missing inputs, the input parents are numeric vectors and they are of same length. It also check if m is valid between 0 and 1.

The test also checks if the function will return expected value and format under a generic situation, as well as a situation where the parents are different at every loci of gene.

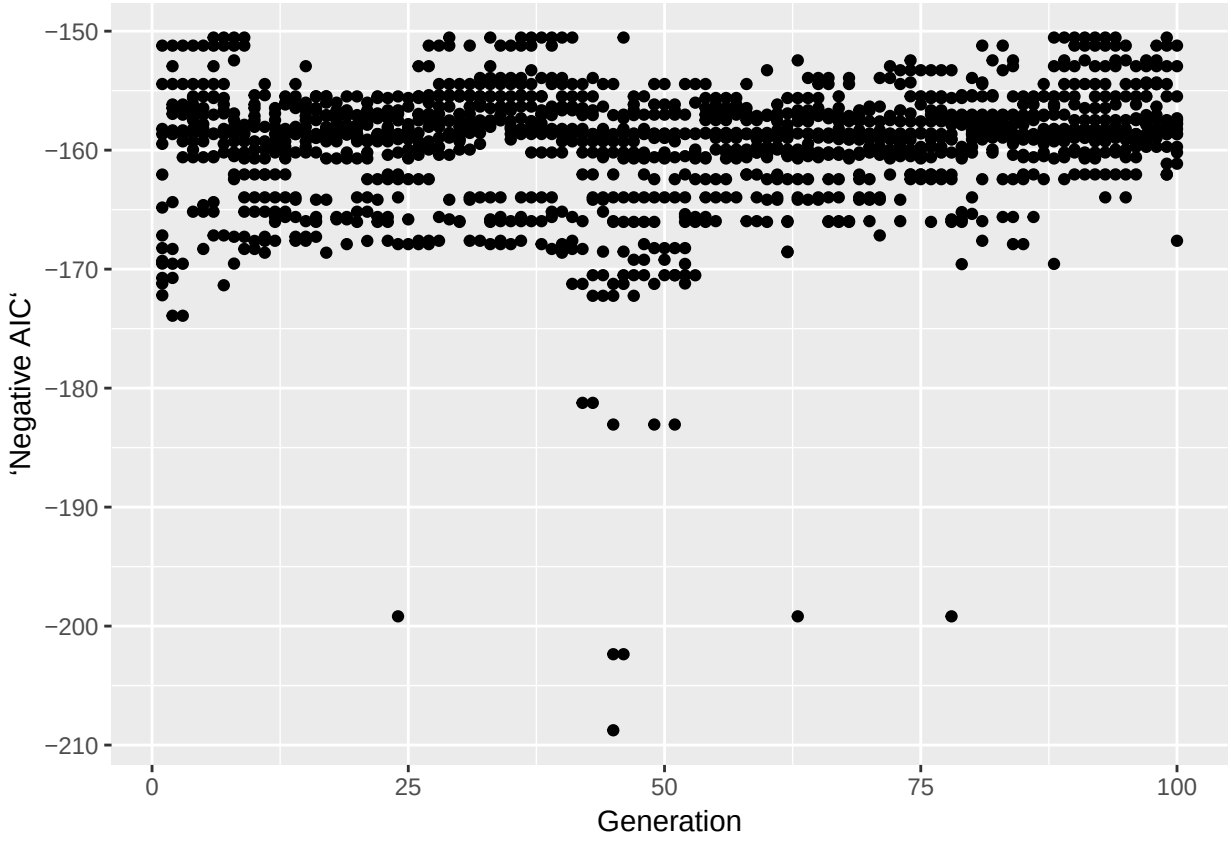
Then, it checks if the function will return error and give error message when the inputs are missing(and missing m should be allowed), not numeric (character in specific) , not of same length or m has an invalid value.

3. Application examples

1) *Motor Trend* Car Road Tests model

We want to regress `mpg` on `cyl`, `disp`, `hp`, `cyl*disp`, `wt`, `gear`, `carb`, and `gear`, using dataset of `mtcars`.

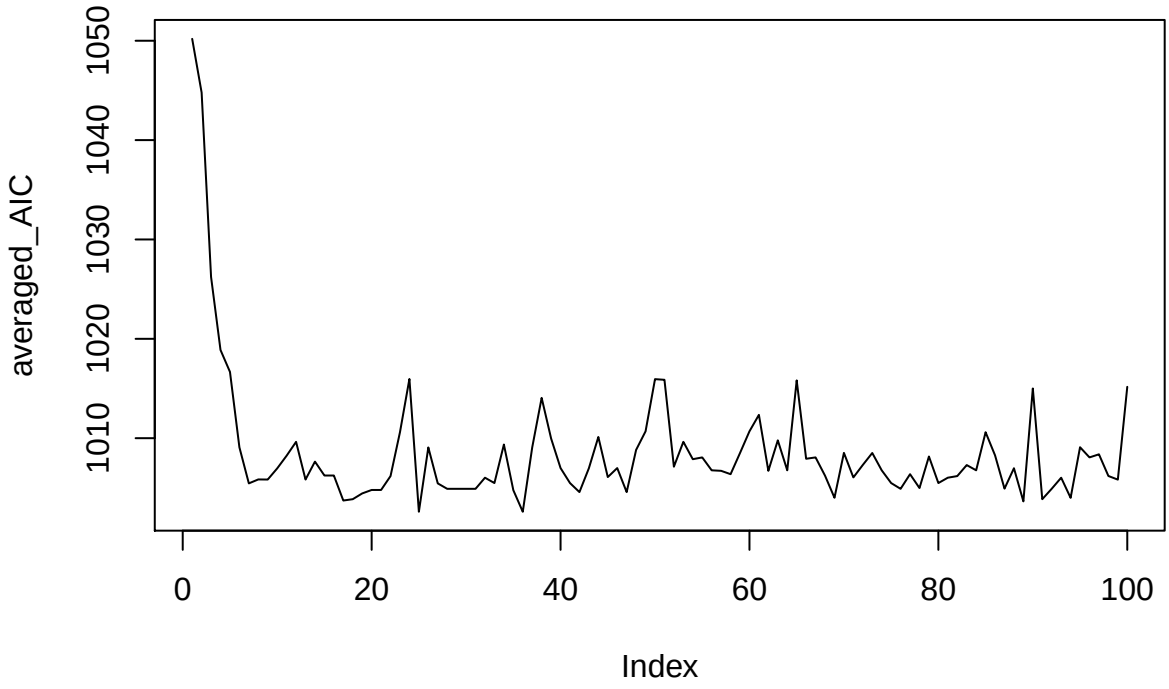
```
formula <- mpg~cyl+disp+hp+cyl*disp+wt+gear+carb+gear
result <- select(formula, data = mtcars, gap = 1)
selected_vars <- result$selected
max_negAIC <- result$fitness
fitness_mat <- result$Neg

gaPlot(fitness_mat)
```



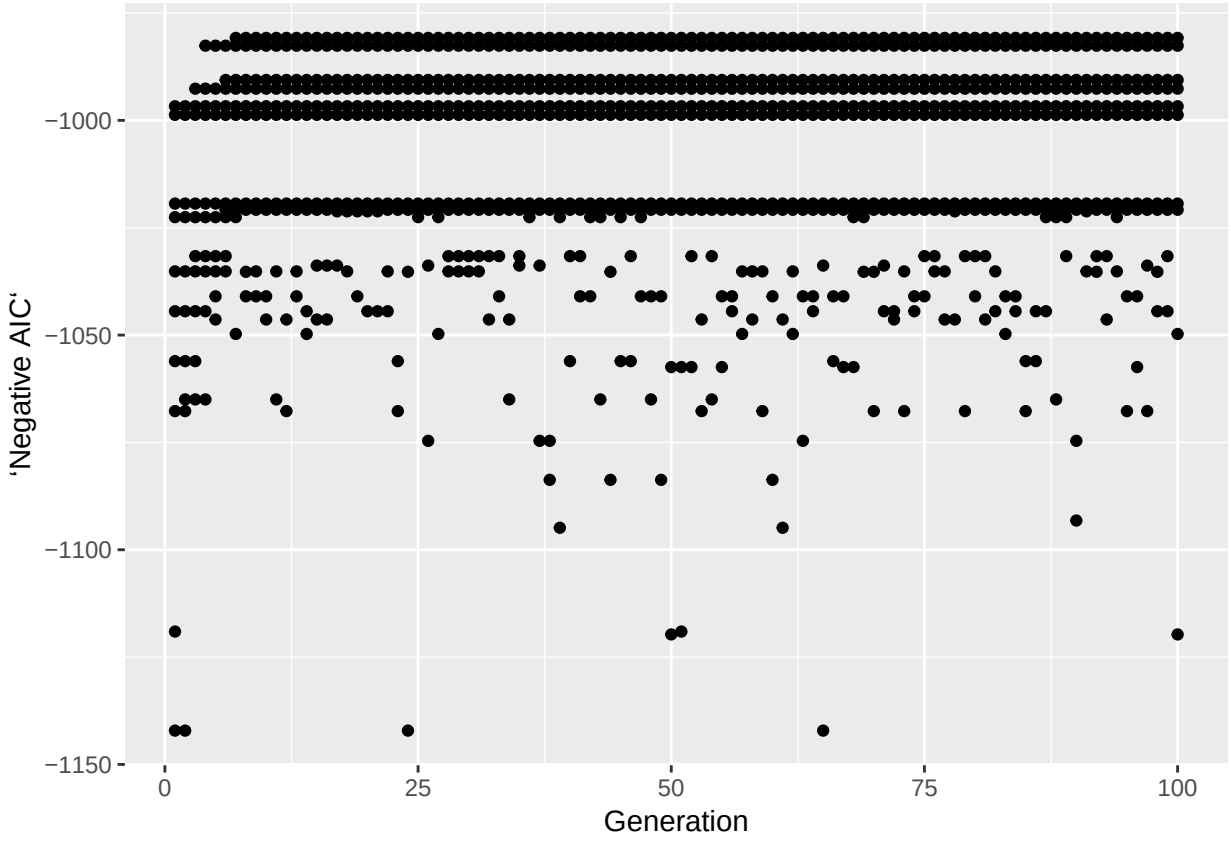
2) Air quality model

We want to build model to predict ozone on solar, wind, temperature, month, and the interaction term of wind and temperature, using dataset of airquality.

```r
formula <- Ozone~Solar.R+Wind+Temp+Month+Wind*Temp
result <- select(formula, airquality, m = 0.05, gap = 0.25)
AIC <- -result$Neg
averaged_AIC <- apply(AIC, 1, mean)
plot(averaged_AIC, type = 'l')
```




```r
selected_vars <- result$selected
max_negAIC <- result$fitness
fitness_mat <- result$Neg

gaPlot(fitness_mat)
```



3) Large scale education dataset

The data was collected from 28,063 students in 6 regions in 2015, which was selected from a larger dataset containing data from 56 regions. The original dataset was collected by TIMSS &PIRLS international study center to compare the different mathematical performance in different regions. Note that `FlagAIB` indicates whether a student's performance is above averge.

```
x <- read.csv("x_train.csv", header=TRUE)
y <- read.csv("y_train.csv", header=TRUE)
data = merge(x, y, by = "StudentID")
colnames(data)
data$StudentID = NULL

#factor
data$FlagAIB = factor(data$FlagAIB)
data$Gender = factor(data$Gender)
data$NumBook = factor(data$NumBook)
data$NumDevice = factor(data$NumDevice)
data$EdMother = factor(data$EdMother)
data$EdFather = factor(data$EdFather)
data$Region = factor(data$Region)

summary(data)

data_TWN = data[which(data$Region == "TWN"),]
summary(data_TWN)
```

```
data_TWN$Region = NULL

formula = FlagAIB ~ .

result = select(formula, data = data_TWN, family = binomial, m = 0.01, gap = 0.25)

selected_vars <- result$selected
max_negAIC <- result$fitness
fitness_mat <- result$Neg

gaPlot(fitness_mat)
```

4. Contributions of each part

main function select(): Zhuoran wrote the function.

crossover(): Zhuoran wrote the components, Yu Hsuan checked and improved the function.

mutation(): Zhuoran wrote the components, Zishan checked and improved the function.

Formal tests: Zhuoran wrote the tests for select(), Yu Hsuan for crossover() and Zishan for mutation().

Help page: Zishan and Yu Hsuan wrote the help page.

This description document: Zishan wrote sectios of approaches and tests, Zhuoran and Yu Hsuan wrote the examples.

5. References

Givens, Geof H., and Jennifer A. Hoeting. Computational Statistics. Wiley, 2013.

Mullis, I. V. S., Martin, M. O., Foy, P., & Hooper, M. (2016). TIMSS 2015 International Results in Mathematics. Retrieved from Boston College, TIMSS & PIRLS International Study Center website: http://timssandpirls.bc.edu/timss2015/international-results/

6. Acknowledgements

Many thanks to Professor Chris Paciorek and Jared Bennett. Thank you Dr.Aijun Zhang for providing the modified large scale education dataset.