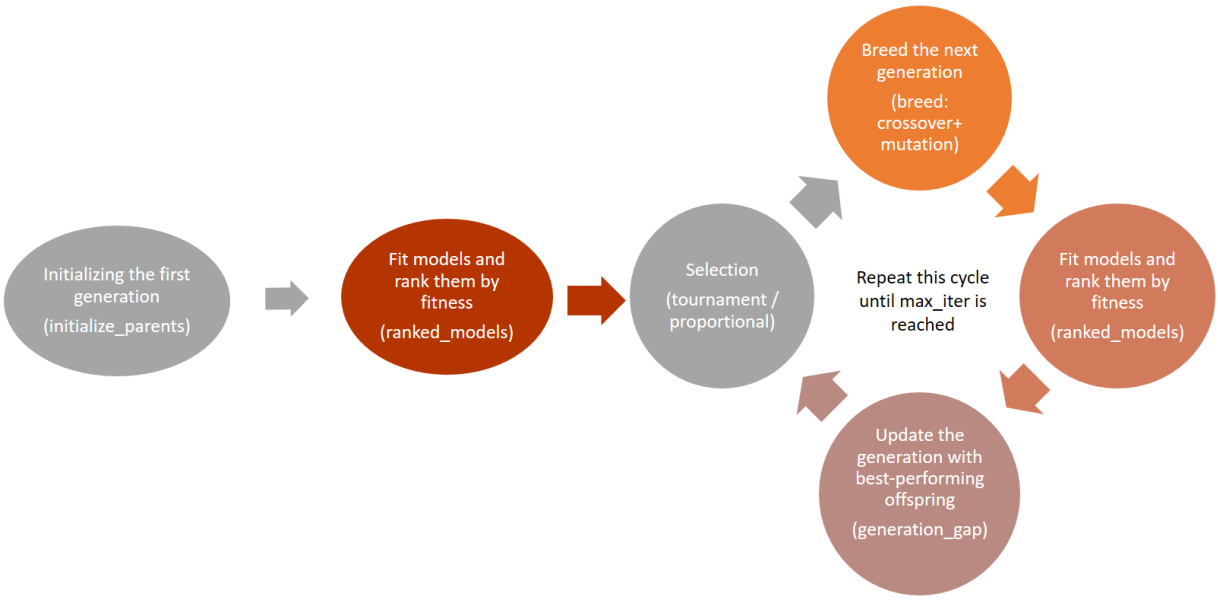
STAT 243 Final Project: Genetic Algorithm Documentation

Kunal Desai, Fan Dong, James Duncan, Xin Shi

December 14, 2017

How *select* Works

Modularity and Approach



The Genetic Algorithm package is designed modularly, each step with auxiliary functions accomplishing discrete tasks.

As the flowchart above shows, the algorithm consists of six steps. First, through *initialize_parents*, we setup the first generation of P models by randomly selecting features for each member of the generation. Once that was completed, we calculate the fitness of each model inside the generation and rank all the models by their fitness (using *calculate_fitness* and *ranked_models*).

Next, we enter the loop and start the first selection process of picking the pairs of parents for the next generation. There are three selection mechanism: proportional, rank or tournament. The first two methods use *proportional*. When calling proportional selection (set random=TRUE), one parent is picked proportional to its fitness and the other picked completely randomly; when calling rank selection (set random=FALSE), both parents are picked proportional to their fitness. Otherwise selection was chosen to be tournament selection (use *tournament*) in which everytime we randomly select k members from the generation and the best in each round becomes a parent. Once the parents had been chosen, the children needed to be created (use *breed*) via cross over. Then, to increase diversity, there is a 1% possibility that the expression of a feature will be randomly altered. Once the children are selected, like their parents, they are ranked by fitness (*ranked_models*). Next, using *generation_gap*, we replace the n worst individuals with n new individuals

from the old generation. This is the final step in determining the new generation. Once this is complete, we pick the member in the generation with the best fitness and make that the overall best member.

We repeat this process until the convergence criteria is met. In our case, the criteria is the maximum number of iterations, which can either be specified by the user or set at a default of 100.

To summarize, when calling the primary function *select*, the following is happening under the hood:

select: put all the functions together while iterating till reaching convergence criteria

- *initialize_parents*: set up the intial generation
- *ranked_models*: rank all the models based on their fitness value
  - *calculate_fitness*: calculate the fitness (AIC by default) of a given feature set
- *breed*: take a generation and output its children
  - *crossover*: take in a list of places to split (number of splits can be specified by the user or 2 by default) and create a set of children
  - *mutate*: mutate some features with a low probability (1% by default)
- *tournament*, *proportional* (random = TRUE / FALSE): select parents out of the current generation
- *generation_gap*: determine which parents will belong in the new generation and which members of the new generation will be kicked out

Testing

In terms of testing, we first ensured that all functions were tested for proper inputs. This includes auxillary functions that aren't intended for public use. We did input sanitization for all functions to ensure that it would gracefully handle mal-formed error including non-integer/float inputs and NA inputs. We also had to write tests for inputs that didn't make sense in relation to the function. For example, if the number of crossover points chosen was greater than the length of the chromosome. Finally, we also ensured that all of our tests worked for inputs we expected it to work on. We also checked for the accuracy of our results in our test cases. When writing our code, we used a pair programming approach to limit defects in the code. We also would write a function and have someone else write tests for it to ensure we could catch as many cases as possible.

An Example

To test the algorithm, we randomly generate a dataset of 40 predictors and 500 observations, in which 6 are real predictors (named $real_1 \sim real_6$) and the rest are pure noise variables($noi_1 \sim noi_{34}$). The response variable, y, is thus given by the equation (coefficients are picked randomly):

$y = \beta_0 + \beta_1 \times real_1 + \beta_2 \times real_2 + \beta_3 \times real_3 + \beta_4 \times real_4 + \beta_5 \times real_5 + \beta_6 \times real_6$

Code for data simulation:

```
n <- 500
c <- 40
X <- matrix(rnorm(n * c), nrow = n)
beta <- c(88, 0.1, 123, 4563, 1.23, 20)
y <- X[ ,1:6] %*% beta
colnames(X) <- c(paste("real", 1:6, sep = ""),
                      paste("noi", 1:34, sep = ""))
```

Ideally, our genetic algorithm will successfully select $real_1 \sim real_6$ out of the 40 predictors.

How the Team Works

The project resides in Kunal's repository (Git Username: kunaljaydesai, Repo name: GA).

The specific tasks completed by each group member is listed below:

- Kunal

Wrote the initialize parents function and calculate fitness function. Wrote tests for respective functions and created testing pipeline (directory and autotester). Created framework for package including roxygen2 documentation setup. Wrote Modularity and Approach and Testing section in this documentation.

- Fan

Wrote ranked_models, tournament and their respective tests. Modified and finalized calculated fitness. Finalized roxygen2 documentation for all functions. Modified and finalized *Modularity and Approach* section in this documentation. Wrote the *An Example* section in this documentation with Xin.

- James

INSERT CONTRIBUTION HERE

- Xin

INSERT CONTRIBUTION HERE