STAT 243 Final Project

Ming Qiu, Junyuan Gao, Jeffrey Kwarsick, Titouan Jehl

December 14, 2017

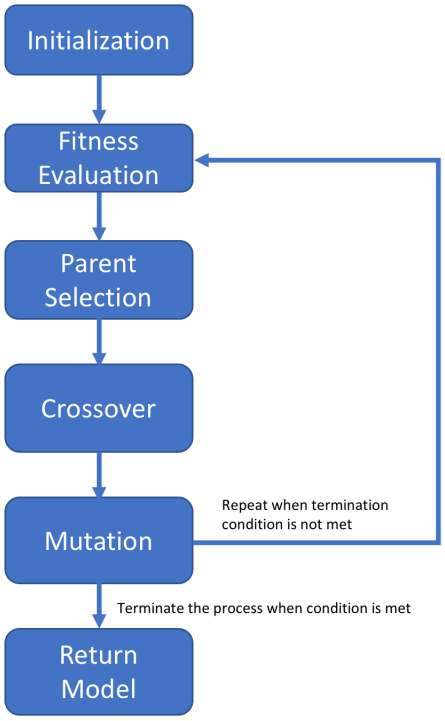
1 Final Project Location

Our Genetic Algorithm (GA) Package is located on Ming Qiu's github. The account name is **carslawbroccoli**. To install the package, run

```
install_github("carslawbroccoli/GA")
```

2 Skeleton of the Genetic Algorithm

The genetic algorithm is designed according to the following flow chart.



To implement the genetic algorithm, we chose to break the algorithm into independent blocks so each of us could work separately. To realize those standalone blocks, we first consider two parts that are separable: the initialization and the loop. Initialization should create the first generation. On the other hand the loop can still be broken into subblocks. We decided to implement 5 methods:

- A **training** method:
  - Input:
    - Candidate: a matrix with each row the sequence of chromosome of candidates, which is a binary vector;
    - Data: the data set;
    - Method: **lm()** or **glm()**;
    - Fitness function: **AIC, BIC** or user defined functions.

  The **training** method returns the fitness values of the candidate chromosomes. This method can deal with all the candidates for one generation at a time.

- A **select_parents** method:
  - Input:
    - fitness_values: avector of fitness values of the candidate of the current generation
    - mechanism: a parent selection mechanism(select both parents by rank/select one parent by rank and the other by random/tournament selection)
    - P: size of generation
    - c: length of chromosome

  This function takes input values and returns $\frac{P}{2}$ parent pairs for the next step.

- A **breed** method:
  - Input:
    - Selected_parent: a matrix with each row the indices of chromosome of one pair of parents.
    - Mutation_rate: the rate of mutation.
    - Crossover_points: the number of crossover points.
    - Gap: the proportion of parents replaced by offspring in the next generation.

  The **breed** method returns the chromosome sequences of the next generation.

- A **get_model()** method:
  - Input:
    - candidate:the sequence of chromosome of a candidate, which is a binary vector;
    - fitness values: a vector of fitness values of the candidate of the current generation;
    - Method: textbf`lm()` or **glm()**;
    - data: the data set;
    - Fitness function: **AIC, BIC** or user defined functions.

  This function takes inputs and returns the model with smallest fitness value.

2.1 Algorithm

Algorithm 1 Genetic Algorithm

1: **procedure** **GA**(**P**, **df**, **Y**, **mechanism**, **crossover_points**, **mutation_rate**, **Gap**)
2: **init(P, df): Generate initial generation P_0.**
3: `iter = 0.` ▷ `iter`: number of generations.
4: `N = 0` ▷ `N`: number of consecutive generations sharing the same minimum fitness values.
5: **while** `iter < max_iter` or `N < 200` **do**
6: ▷ Terminate after `max_iter` or minimum fitness values unchanged for 200 generations.
7: **training(P_{iter}, df):** Evaluate the fitness values of P_{iter}.
8: **select_parents(mechanism):** Select the parents according to mechanism.
9: **breed(crossover_points, mutation_rate, Gap):** Generate the next generation with crossover, mutation. Replace parents by the offspring according to Gap.
10: `iter = iter + 1.`
11: **get_model(P_{iter}, df)**: Evaluate the fitness value and model for the selected one.
12: **return** A list with **iter, model, fitness_value**.

Each of these methods are completely stand alone and can be tested without the rest of the method. However, provided the right comments to standardize the input and the output so assembling the blocks doesnt raise errors, this skeleton enables the team to separately implement the algorithm efficiently.

3 Collaborator Contributions

3.1 Ming Qiu's Contribution

Ming Qiu worked on the select.R function that handles the iterations for the genetic algorithm as well as the init, training, get_model functions. She also worked to implement plotting features for our algorithm and improving the code from other parts. Lastly, she worked to debug the package as a whole to run more efficiently and help prepare the help manual.

3.2 Junyuan Gao's Contribution

Junyuan Gao worked on the formal testing for the completed algorithm, checking to ensure thatproper inputs for each function in our algorithm were able to handle improper inputs. He also worked on writing the breeding function (crossover and mutation included within the breeding function).

3.3 Jeffrey Kwarsick's Contribution

Jeffrey Kwarsick worked on the *select_parents()* function. He was also responsible for building the package and making sure it carried the proper structure. He ensured that it was able to be installed through github and tested. Lastly, he prepared all of the documentation within the package itself.

3.4 Titouan Jehl's Contribution

Titouan designed the skeleton. He coded the skeleton of the utils.R file so the whole group could be on the same page about the data type of the inputs and outputs. To do so I adopted the Google style of comment for each method.

4 Testing

We conducted testing on all of the inputs for all functions in our package and confirmed that if an invalid input was entered, that the code would properly handle the error and not execute. In addition to testing

all of the outputs, all the functions were also tested independently. Below is an itemized list of how each function within our algorithm was tested.

4.1 Unit Tests

- **`init()`**: For this function, since the input will be evaluated in the further integration test, we only test the type of output and the random initialization(i.e. Repeated initialization does not return the same chromosomes).
- **`training()`**: For this function, we create an input test to check whether inputs are in correct type, an output test to check whether outputs are in correct type and two functional tests to check whether each option of regression methods(i.e. `lm()` or `glm()`) and fitness functions(e.g. AIC, BIC) can work.
- **`select_parents()`**: For this function, we make an input test to check whether inputs are in correct type and several tests that checked correctness and consistency(i.e. our algorithm will always produce correct number of parent pairs at any time) of the number of parent pairs. Moreover, a test that checks the whether selecting process is random is also included.
- **`breed()`**: For this function, we firstly make an input test to check whether inputs are in correct type and an output test to check whether outputs are in correct type. In addition, several functional tests are made to check whether crossover and mutation will produce exactly different new generations and make sure that the algorithm only works when the generation gap rate $\in (0, 1]$.
- **`get_model()`**: Since this function is very similar to `training()`, we just make a output test to check whether the output is a `lm` or `glm` object.

4.2 Integration Test

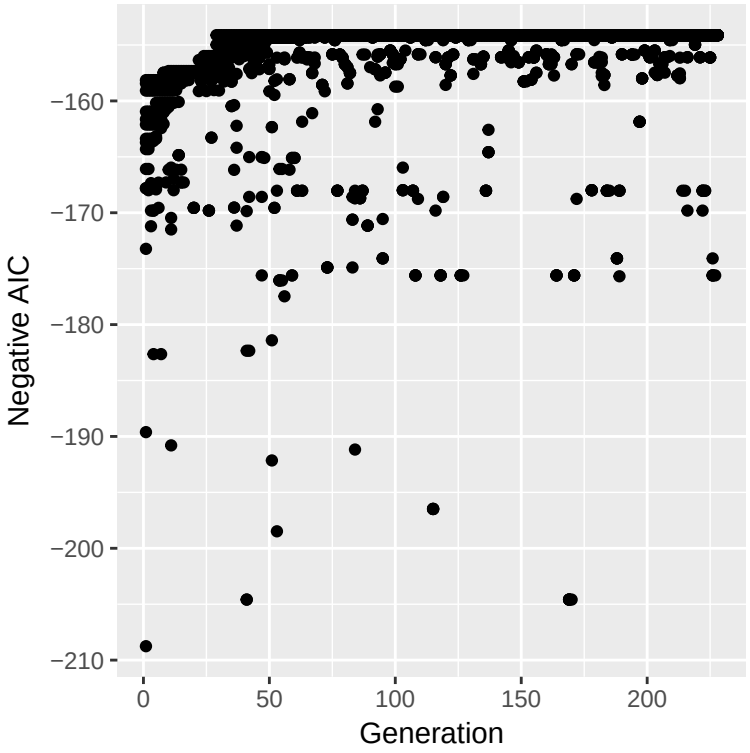
In this section, we evaluate the main function `select()` in several ways:

1. Create an input test to check whether inputs are missing and in correct type.
2. Create an output test to check whether outputs are in correct type.
3. A stopping condition check whether the `max_iter` and convergence condition(i.e. we think the algorithm converge until number of consecutive generations sharing the same minimum evaluating values ≥ 200) are useful.
4. We test the consistency of convergence(i.e. algorithm converge to same result regardless of the selection of mechanism)
5. Lastly, we verify the efficiency difference when using different parent selection mechanism.

5 Example

Below is an example for fitting using an R dataset "mtcars", with the dependent variable being "mpg."

```
library('GA')
mtcars_model <- select(mtcars, "mpg", plot.return = TRUE)
```


Negative AIC
−160
−170
−180
−190
−200
−210
0
50
100
150
200
Generation