# 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 Qius github. The account name is **carslawbroccoli**.

## 2 Skeleton of the Genetic Algorithm

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. Initialisation should create the first generation from the data provided by the user and the number of individual per generation. On the other hand the loop can still be broken into subblocks. We decided to implement 5 methods:

- a training method that takes as input a candidate with his information, data, a model to fit and fitness function. This method fits the model on the data with respect to the candidate genetic information. It then outputs the fitness value of this candidate. This method dealing with only one candidate as a time, it enables us to later parallelize the update of a generation by running this method on different cores.
- a select parent method that takes as input the vector of fitness values of the candidate of the current generation as well as all the way to select the parents - with/without replacement, tournament, etc. This method selects pairs of parents with respect to the fitness values of the candidate. It returns a list of pairs of parents that will breed to create the next generation
- a breeding method that takes as input the pairs of parents, and the input needed to chose the breeding technique (number of crossover points etc.). This methods create the candidates of the next generation. This method does not implement the mutation. Note that this method could have been split in a smaller block on which we could have parallelize a loop. The output of this method is the binary dataframe representing the genetic information of the next generation
- a mutation method that takes as input the new generation and the rate of mutation to modify the genetic information randomly and output the new generation post mutations
- a get model method that returns for a given candidate the model that has been fitted on it

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 training function. She also worked to implement plotting features for our algorithm with the *get_model()*

function. Lastly, she worked to debug the package as a whole to run more efficiently. She also worked on the $get_model()function$.

### 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.*

- *training*
  - *The algorithm works with both lm() and glm().*
  - *That the algorithm works with both AIC and BIC for fitness functions.*
  - *That the output vector of the training function is non-zero and correctly outputs all fitness values for candidate chromosomes.*
- $select_parents$
  - *The function produces correct length of parent pairs for breeding*
  - *That each call of select parents() produces a different set of parent pairs for breeding.*
  - *If either selection mechanism is called, the same number of parent pairs (based on the number of input parents) is the same.*

*breed*

- *That the output is a data frame of the correct (P x c) dimension*
- *Everytime repeated crossover and mutation will produce different new generations*
- *That the generation gap can only be between (0,1]*

*get_model – output is the same class as the training method*

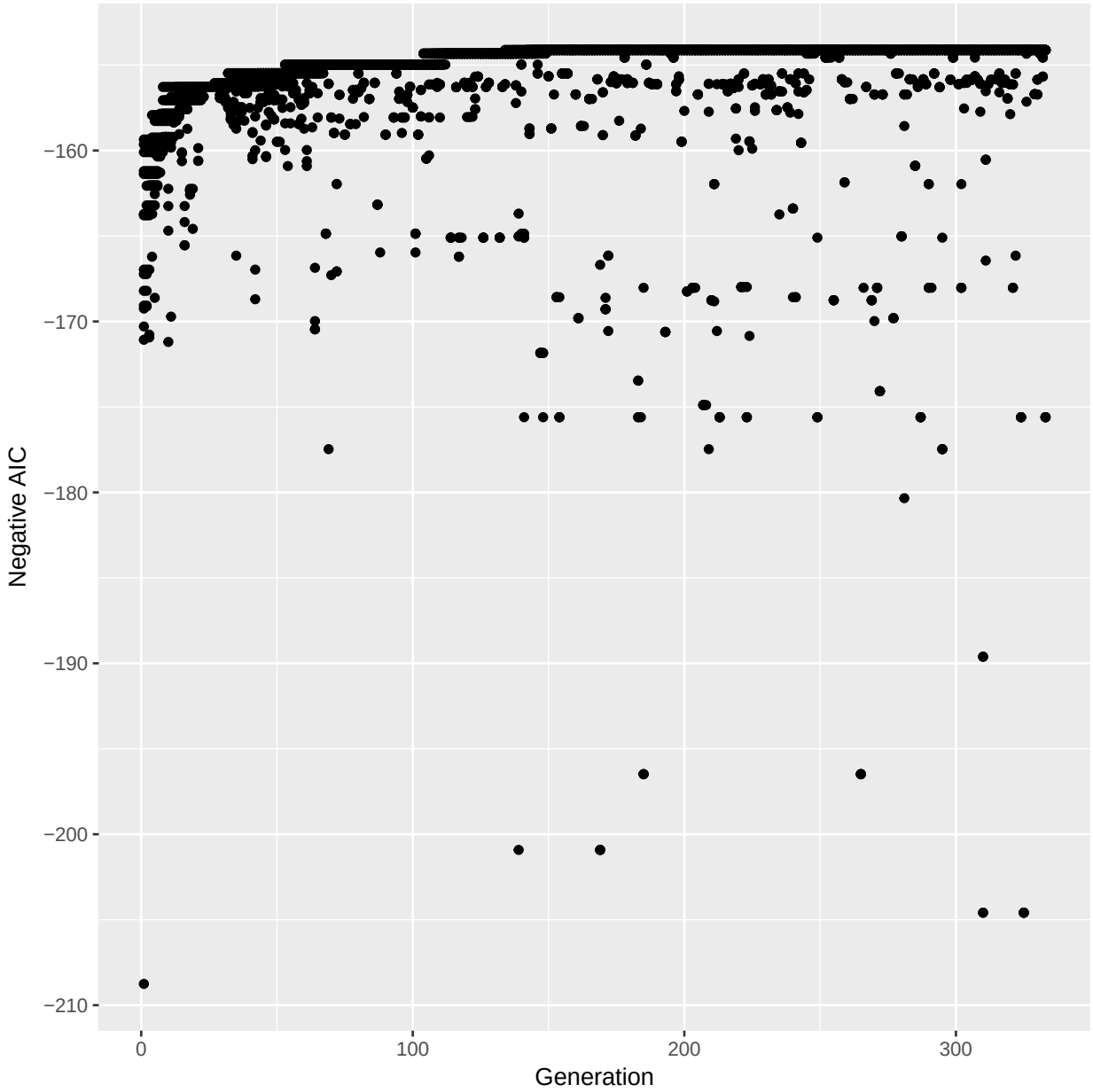
*select – integration test of the function, checking all inputs*

# 5 Example

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

```
library('GA')
library('ggplot2')

select(mtcars, "mpg", plot.return = TRUE)
```




```
## $count
## [1] 333
##
## $model
##
```

```
## Call:
## method(formula = fmla, data = X)
##
## Coefficients:
## (Intercept)           wt         qsec           am
##       9.618       -3.917        1.226        2.936
##
##
## $fitness_value
## [1] 154.1194
##
## $plot
```