STAT 243 Final Project: Genetic Algorithm Documentation

Kunal Desai, Fan Dong, James Duncan, Xin Shi

December 6, 2017

How *select* Works

Modularity and Approach

When designing our package, we first planned out the necessary components in a way that would not only be easy to divide up work, but would also provide modularity in our program. We first wrote a function that would randomly determine how to setup the first generation and which features should be included in each member of the generation. Once that was completed, we wrote a function that would calculate the fitness of a member inside a generation. After all of the member's had their fitness calculated, it would be ranked to determine the most fit members of the population. Next, we would pick the pairs of members that would become parents for the next generation. These members were picked proportional to their fitness if selection was chosen to be proportional. Otherwise selection was chosen to be tournament selection in which the generation is randomly partitioned k-ways and the best in each group becomes a parent. Once the parents had been chosen, the children needed to be created via cross over. We also included a mutation step as the reading mentioned in which we would randomly alter the expression of a feature. Once the children our selected, they are ranked and we replace the n worst individuals with n new individuals from the old generation. This is our implementation of the generation gap which 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. Then, we continue iterating till we reach out convergence criteria. In our case, we determined the convergence criteria to be determined by a number of iterations that the user can specify or is defaulted at 100 iterations. The code is broken modularized into these functions:

1. initialize parents: sets up the intial generation
2. calculate fitness: calculates the fitness of a given feature set
3. ranked models: rank all the models based on their fitness value
4. breed: takes a generation and outputs its children
   (a) crossover: takes in a list of places to split and creates a set of children
   (b) mutate: mutates some features with a low probability
5. tournament, proportional: two ways of selecting parents
6. generation gap: determines which parents will belong in the new generation and which members of the new generation will be kicked out
7. select: puts all the functions together while iterating till reaching convergence criteria

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 model, generation gap and wrote their respective tests. Modified and finalized calculated fitness and wrote its respective tests. Wrote the *An Example* section in this documentation.

James *INSERT CONTRIBUTION HERE*

Xin *INSERT CONTRIBUTION HERE*