Final Report

Mirella Rodriguez, Joan Martinez, Sylvia Yang

12/18/2019

Introduction

This document presents the solution for the final project that consists of an R package that implements a genethic algorithm for variable selection in regression problems. The functionality of this package extends to linear regressions and to GLMs. Here, we describe step-by-step the rationale used in developing the package.

Givens & Hoeting (2012: 77) identify the following components of genetic algortihms: selection, crossover and mutation (see Figure 3.5., for instance); and we take them as the building blocks of our main function as well. The goal of genetic algorithms is to find a solution θ^* that is a member of a space of candidate solutions, i.e. $\theta \in \Theta$ such that $\theta^* = arg\max f(\theta)$; where $\Theta \subset \mathbb{R}^L$ for L. This type of algorithms mimic the process of natural selection, hence they favor *fitness* of the candidate solutions, θ.

In the following paragraphs we will describe how we implemented the selection, crossover and mutation processes adapted to a problem of **variable selection**. In this setting, the population is represented by a matrix and the length (of chromosomes in traditional genetic algorithms) is equivalent to the nunber of columns of this population matrix. Preceeding the three building blocks we have mentioned, the algorithm requires two stages: initialization and fitness assignment.

Code

Our main function *select()* takes in the values dataset, criterion, model, generation, and size. For a more detailed description please view the man page using ?select in our GA package. The *select()* returns a list the best fitted predictors, fitness score, and coefficient values for the predictors. Please read sections below regarding the auxilary functions we created and their role in the genetic algorithm.

Initialization

First, we aim to create our initial population. We do so by setting up a function that generates random sample of boolean values. Let us note that genetic algorithms generate a new population of individuals at each generation based on the selection of individuals with highest fitness.

Fitness

Following the initialization, we assign fitness values to each chromosone in the initialized matrix. In order the evaluate the fitness of each chromosome we run the user specified model to obtain the objective criterion. After obtaining their objective criterion, we use the "Rank Based" method in which the chromosomes are sorted, then their fitness is assigned based on their rank. $\phi(i) = k \times R(i) \quad i = 1, ..., N \quad R(i) : rank$ (*Gomez & Queseda*) in which we chose our k constant, the selective pressure, to be 1.5.

Selection

After we have assigned their fitness, we use that score to select the new parent chromosome matrix that will recombine for the next generation. We select the chromosomes that are most fit for the next genration i.e. the ones with the highest fit score. We use the selection method known as the "Roulette Wheel" in which we place the chromosomes on a "roulette" and their area is determined by their fitness. Therefore if they have a higher fitness, they are more likely to be chosen for the next generation.

Reproduction

Once we have determined the new parent chromosome matrix for the preceeding generation, we begin to implement our crossover and mutation procceses. We begin this by taking our parent chromosome matrix and selecting two consectuive chromosomes to undergo the crossover and mutation processes. This function returns the crossed and mutated matrix.

Crossover

The Reproduction function mentioned above provides the crossover function with two chromosomes at a time, which we define as the "parents". The method we chose to implement was the uniform crossover method in which we look at individual genes (or bits) of the parent chromosomes. This is also known as the "coin flipping" method in which each gene has a 50% chance of coming from parent 1 and a 50% chance of coming from parent 2.

Mutation

Before returning the crossover offspring back to the reproduction function, we have the offspring undergo a mutation process. To determine whether a gene will be mutated we generate a binary value and assign the probability of observing a 1, in which we do mutate the gene, to be $\frac{1}{m}$ where m is the number of features. Conversly, we assign 0, in which we do not mutate, to be $1 - \frac{1}{m}$. Once it has completed it returns back to crossover which then returns back to reproduction. We repeat this process until we have gone through the entire population.

Testing

The tests used the package `testthat`. We organize the tests in three contexts that are all contained in a single *.R* file:

- Initialization
- Fit
- Select

In the *Initilization* context, we propose two tests that aim to detect if the user enters a non admitted input, i.e. NULL or NA dataset. In the *Fit* context, we test for the cases when the user has entered an empty dataset (i.e. NA) as an argument of the function `fit`. Aditionally, we test for the user possibly entering a range of datasets that contain numeric, factor, string, etc. types of variables. For doing this, we test the fit function with the R sample datasets named `mtcars, iris, ToothGrowth` and `USArrests`.

Next, in the *Select* context, we aim to test wether the user enters the appropiate dataset as a *select()* function argument; and if the (stochastic) algortihm computes estimates the optimized objective function in the neighborhood of the known local maxima using one of the sample R datasets when the data generating process for the initial population is changed. In the latter type of tests, we use a seed to fix our performance measure of fitness (i.e. objective function).

First, we test that the function *select* works and gives the correct type of output (i.e. a list) with a set of sample R datasets. Next, we test wether the *select* function works with different initializations of the initial population. In this test, we choose and arbitrary data generating function, a uniform density function, to generate the population. With this input, we make use of the select function and check wether with this arbitrary initial population our function *select* achieves to have a final criterion function that locates within a fixed range of the **known** maximum criterion function (154.2075). There is one test that checks that the output fitness is within the upper and lower bound computed around the know maximum. Finally, the last two tests do a similar test but using the binomial density in order to generate the initial population.

Implementation

Example 1: Using Motor Trend Car Road Tests Data

For this example, we will use this dataset to predict Miles/(US) gallon based on the 10 features provided in the dataset. We will first run the linear model with all the predictors and then do a comparison using our genetic algorithm and compare the AIC scores.

Displaying the data:

```
head(mtcars)
```

```
##                   mpg cyl disp  hp drat    wt  qsec vs am gear carb
## Mazda RX4         21.0   6  160 110 3.90 2.620 16.46  0  1    4    4
## Mazda RX4 Wag     21.0   6  160 110 3.90 2.875 17.02  0  1    4    4
## Datsun 710        22.8   4  108  93 3.85 2.320 18.61  1  1    4    1
## Hornet 4 Drive    21.4   6  258 110 3.08 3.215 19.44  1  0    3    1
## Hornet Sportabout 18.7   8  360 175 3.15 3.440 17.02  0  0    3    2
## Valiant           18.1   6  225 105 2.76 3.460 20.22  1  0    3    1
```

Running a linear model using all the predictors and obtaining the AIC score

```
AIC(lm(mtcars$mpg ~ ., data = mtcars))
```

```
## [1] 163.7098
```

Now, let's use the same dataset with our genetic algorithm

```
select(dataset = mtcars, criterion = AIC, model = lm, generation = 20, size = 50)
```

```
## $finalPredictors
## [1] "wt"   "qsec" "am"
##
## $finalCriterion
## [1] 154.1194
##
## $finalCoeff
## (Intercept)          wt        qsec          am
##    9.617781   -3.916504    1.225886    2.935837
```

From this, we can see that we reduced the AIC score from ~ 163 to ~ 154 using our genetic algorithm. Additionally, we only kept 3 predictors.

Example 2: Using Violent Crime Rates by US State Data

For this example, we will use this dataset to predict Murder arrests (per 100,000 based on Assault arrests (per 100,000), Percent urban population, and Rape arrests (per 100,000). We will first run the generalized linear model with all the predictors and then do a comparison using our genetic algorithm and compare the BIC scores.

Using the entire glm() function with all the predictors

```
BIC(glm(USArrests$Murder ~ ., data = USArrests))
```

```
## [1] 251.8409
```

Using our genetic algorithm

```
select(USArrests, BIC, glm, generation = 50, size = 50)
```

```
## $finalPredictors
```

```
## [1] "Assault"
##
## $finalCriterion
## [1] 248.2629
##
## $finalCoeff
## (Intercept)     Assault
##  0.63168266  0.04190863
```

From this, we can see that we reduced the BIC score from ~ 251 to ~ 248 using our genetic algorithm. Additionally, we only kept the assault predictor.

Contribution

Mirella wrote the reproduction, mutation, crossover functions, and wrote the select function with Sylvia. Joan wrote the tests using the testthat package, wrote comments in the R code, contributed to the final write-up, and modified and improved the algorithm. Sylvia wrote the fitness and selection functions, wrote assertions in the code, and contributed to the overall select function. Mirella did the package set up, wrote the roxygen2 documentation, and contributed to the write-up.

Citations

"Chapter 3: Combinatorial Optimization ." Computational Statistics, by Geof H. Givens and Jennifer A. Hoeting, Wiley, 2013.

Gomez, Fernando, and Alberto Quesada. "Genetic Algorithms for Feature Selection." Genetic Algorithms for Feature Selection | Machine Learning Blog, www.neuraldesigner.com/blog/genetic_algorithms_for_feature_selection.