Implementing a Genetic Algorithm for Variable Selection in Regression

Asem Berkalieva, Jing Xu, Zihan Ye

12/12/2019

Overview

For this project, we created a package to implement a genetic algorithm for variable selection in regression problems using the guidelines outlined in chapter 3 of Computational Statistics by Geof H. Givens and Jennifer A. Hoeting. We wrote modular functions for each step of the algorithm and used those functions in the body of the main function, which we called select(). Select will take 6 inputs:

1. The object corresponding to the dataset being used.
2. A character string corresponding to the name of the response variable the user would like to model.
3. A vector of the character strings of names of the predictor variables (covariates) that the user would like to consider using. This allows the user to directly leave out certain information in the dataset that they don't consider to be useful in prediction.
4. A character string corresponding to the name of the objective criterion/fitness function that will be used to evaluate each model. By default, the function uses "AIC", but the user can use other objective criterion available in base R (ex. BIC) or write their own function.
5. A character string corresponding to the type of regression the user would like to perform. By default, the function uses "Gaussian", which is the default identity link that's equivalent to performing standard linear regression. However, the user can specify other families, such as binomial, which uses a logit link by default and allows the user to perform logistic regression. The user can specify any family that is accepted by the glm() function.
6. A boolean variable called "maximize", indicating whether the objective criterion should be maximized or minimized. For example, if the objective criterion select() is using is written by the user and the user would like to maximize this objective criterion (instead of minimize, as one would if the objective criterion were AIC), they can specify "maximize = TRUE" as an input. By default, maximize is false.

Functions

Here are the functions that we wrote to implement the algorithm:

Generate

The generate() function initializes a list of p sequences (consisting of 0's and 1's), which each sequence representing a regression model. We will refer to the list of p sequences as a "generation". The length of the sequence is equal to the number of covariates that the user would like to consider using in the regression model. A 0 in the 1st number in the sequence means the model won't use the 1st covariate, and a 1 in the 3rd number in the sequence means the model will use the 3rd covariate. The number of sequences generated, p, is determined by the select() function (more details in the "Select" section). We chose to generate a "1" in a given sequence with higher probability than "0" because we want to initialize models that include more covariates (rather than leaving out more covariates). This will also avoid the problem of having sequences

that contain only 0's, which is useless in the context of regression since one cannot fit a model without any covariates.

Selection

The selection() function takes as input a generation (list) of p sequences and outs $\frac{p}{2}$ pairs of "parents", with each pair being a numeric vector of 2 index values, each corresponding to a sequence in the input. We chose to select one parent with probability proportional to fitness and one parent completely at random, which will reduce the chance of one parent dominating the gene pool at early iterations of the algorithm and cause the algorithm to converge prematurely. We used a rank-based method for selection by ranking each model based on the objective criterion (which is AIC by default) and assigning each model a probability based on rank as follows:

$$\phi(\vartheta_i^{(t)}) = \frac{2r_i}{P(P+1)}$$

where r_i is the rank of the sequence based on the objective criterion (higher is better), P is the number of sequences in a generation, and ϕ is the probability of being selected as a parent for a given model. For a given iteration number t, the ith sequence can be chosen as parent 1 with probability $\phi(\vartheta_i^{(t)})$. Parent 2 is chosen completely at random, so each sequence has an equal chance of being selected as parent 2.

Crossover

The crossover() function takes as input a numeric vector of two index values and performs the crossover genetic operator on the 2 "parent" sequences in the generation corresponding to those two index values. Let c be the length of a sequence in this problem. To achieve this, we chose a splitting point to split the sequences. The splitting point will be a value between 1 and $c-1$, randomly drawn with equal probability. If the value of the splitting point is 2, then the function will split each sequence into two parts, with one part consisting of the sequence up to (and including) the 2nd bit and one part consisting of the sequence after (not including) the second bit. As a result, the function splits each sequence into parts A and B. To generate "child" sequences from the "parent" sequences, the function concatenates part A of sequence 1 with part B of sequence 2 to create the first child and does the same with part A of sequence 2 with part B of sequence 1 to create the second child.

Mutate

The mutate() function takes as input a sequence of length c and mutates each bit in the sequence with probability $\frac{1}{c}$. We chose the mutation rate of $\frac{1}{c}$ because "theoretical work and empirical studies have supported a rate of $1/C$" [Givens and Hoeting, 80]. If after mutating a chromosome, the chromosome contains all 0's, then the function repeats the mutation process until the chromosome doesn't contain all 0's, as it's not possible to fit a model with no covariates.

Choose

The choose() function takes as input a list of mutated sequences and returns the "best" sequence in that generation in terms of objective criterion score, along with its objective criterion score and the model object of the regression model corresponding to that sequence. To achieve this, the function fits a model for every sequence provided as input, computes its score according to the objective criterion, and finds the sequence that provided the "best" score. Again, choose() will return the sequence, the score, and the model object corresponding to that sequence.

Select

Lastly, the select() function uses the modular functions written earlier and implements the genetic algorithm as follows:

1. Generate an initial list of p sequences using the function generate(). Since this sequence is a binary encoding representing whether or not to include a covariate in the model, one suggested way to do this is to choose p such that $C \leq P \leq 2C$, where c is the length of each sequence [Givens and Hoeting, 79]. We decided to make $P = ceiling(\frac{1.5 \times C}{2}) \times 2$, which ensures that it's an even number between C and $2C$.

Then, run steps 2-5 until the absolute value of the difference between the objective criterion score of the current iteration and the objective criterion score of the previous iteration is less than 10^{-4} (absolute convergence).

2. Select $\frac{p}{2}$ pairs of parents to do genetic operations on using the select() function.
3. Use crossover() to create a pair of children for each of then $\frac{p}{2}$ pairs of parents.
4. Use mutate() to mutate each of the p children sequences.
5. Fit the models corresponding to each of the p mutated sequences and return the best model along with its objective criterion score using choose().
6. Once the algorithm converges, return the model with the best objective criterion score from the final iteration of the algorithm, and that will be the model that is selected by the genetic algorithm.

Tests

Tests for the package includes a series of sophisticated tests for the overall function "select" and unit tests for individual functions including "selection", "crossover", "generate" and "mutation".

First, we test the type and format of the inputs and outputs for individual functions to make sure that they're what we expect them to be. Then, we test whether the individual functions works collabratively in the overall funcion "select". Finally, we test the overall "select" function with given datasets and expected outputs, and test whether or not it works with standard input, with a subset of the available covariates, with a non-linear form of regression (ex. logistic), and that the model won't include a covariate that has low correlation with the response variable.

Generate

For generate(), we test for the following properties:

1. Whether generate function generates a nested list.
2. Whether the elements with in each list is an integer.
3. Whether the size of the object generated is equal to the given inputs, and perform the same check for each list generated within the nested list.

Crossover

For crossover(), we test for the following properties:

1. Whether crossover() returns an object of length 2.
2. Whether crossover() each of the two offsprings have the same size/length as the parents.

Selection

For selection(), we test for the following properties:

1. Whether or not the returned object is of length p/2 (since the object is a list of p/2 pairs).
2. Whether or not each of the pairs in the list object is of length 2 (i.e. only 2 sequences in a pair).
3. Whether or not the sequences in a pair are of the same length.
4. Whether or not we can use other objective criterion/fitness functions with selection().

Mutate

For mutate(), we test for the following properties:

1. Whether mutate() function works with a single numeric vector as input.
2. mutate() doesn't change the length of the object.
3. mutate() doesn't change the type of the object.

Select

For select(), we test for the following properties:

1. Whether select() works fine with a set of standard input (response variable, covariate variables, and dataframe) and returns the expected class.
2. Whether select() works with fewer covariates than is contained in the data frame input.
3. Whether select() works fine with another regression model rather than the default (linear regression).
4. If we create a new randomly generated covariate (unrelated to the response variable), select() won't include it in the chosen model most of the time. We consider this a "known truth" because a variable generated independently of the response variable should have little correlation with the response and thus should not appear in the model. Note that this last test takes a longer time to run, since we decided to repeating the algorithm 20 times and test the proportion of times the false covariate was correctly excluded from the final model.

Application

```
library(GA)
```

```
##
## Attaching package: 'GA'
```

```
## The following object is masked from 'package:base':
##
##     choose
```

For the examples below, we will use mtcars, which is a built-in dataset containing 11 variables. Here is how the dataset looks like.

```
data <- mtcars
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
```

Example 1

First, we will run the genetic algorithm for linear regression using all the available variables in the dataset to predict "mpg".

```
response <- "mpg"
covariates <- c("cyl", "disp", "hp", "drat", "wt", "qsec", "vs", "am", "gear", "carb")
select(data, response, covariates)
```

```
## Objective Function Score At Final Iteration: 157.584
##  Objective Function Score At Penultimate Iteration: 157.584
##  Number of Iterations: 6
```

```
##
## Call:  glm(formula = formula, family = family, data = data)
##
## Coefficients:
## (Intercept)           hp           wt         qsec           vs
##    19.18191     -0.01376     -2.94501      0.66248      0.47817
##          am         carb
##     3.31485     -0.34780
##
## Degrees of Freedom: 31 Total (i.e. Null);  25 Residual
## Null Deviance:       1126
## Residual Deviance: 156.4     AIC: 157.6
```

Example 2

Here, we will use select() to perform logistic regression. Since the variables "vs" takes values 0 or 1, we will perform logistic regression on that variable using the remaining 10 as covariates.

```
response <- "am"
covariates <- c("mpg", "cyl", "disp", "hp", "drat", "wt", "qsec", "vs", "gear", "carb")
select(data, response, covariates, family = "binomial")
```

```
## Objective Function Score At Final Iteration: 10
##  Objective Function Score At Penultimate Iteration: 10
##  Number of Iterations: 4
```

```
##
## Call:  glm(formula = formula, family = family, data = data)
##
## Coefficients:
## (Intercept)           hp         drat           wt           vs
##   -393.4086       0.5503     181.1751    -114.4893     -68.9090
##
## Degrees of Freedom: 31 Total (i.e. Null);  27 Residual
```

```
## Null Deviance:       43.23
## Residual Deviance: 4.297e-09     AIC: 10
```

Example 3

Here, let's say we don't think the variables "gear" and "carb" are useful for predicting "mpg". In this case, we can provide as input a subset of all the available variables in the dataset as covariates and select() will only run the genetic algorithm on that subset of variables. I will also use BIC as the objective criterion to show that the function can take other objective criterion functions.

```
response <- "mpg"
covariates <- c("cyl", "disp", "hp", "drat", "wt", "qsec", "vs", "am")
select(data, response, covariates, criterion="BIC")
```

```
## Objective Function Score At Final Iteration: 164.887
##  Objective Function Score At Penultimate Iteration: 164.887
##  Number of Iterations: 10
```

```
##
## Call:  glm(formula = formula, family = family, data = data)
##
## Coefficients:
## (Intercept)          cyl         disp           wt
##   41.107678    -1.784944     0.007473    -3.635677
##
## Degrees of Freedom: 31 Total (i.e. Null);  28 Residual
## Null Deviance:       1126
## Residual Deviance: 188.5     AIC: 157.6
```

Contributions

For the function, Asem wrote crossover() and choose(), Jing wrote generate() and mutate(), and Zihan wrote selection(), the overall function select(), and the docstrings for the functions. Asem and Jing wrote the tests for each function and Zihan wrote the manual. All three of us contributed to this writeup by writing the sections relevant to the parts of the project that we worked on. The package resides in Asem's GitHub.com repository (username: asem-berkalieva).