STAT243 Final Project: Regression Model Selection Using Genetic Algorithms

Florian Papion
Qiuhan Wu
Brianna Noland

12/12/2014

Introduction

A genetic algorithms (GA) is a search heuristic that was originally developed by John Hollands in 1975. It mimics the process of natural evolution and uses concepts of Natural Selection and Genetic Inheritance developed by Charles Darwin (1859). It is widely used in business, science, and engineering, especially in optimization and search problems, as well as scheduling and timetabling.

Overview of GAs for Model Selection

In this report, GA is implemented for variable selection in regression problems, including both linear regression (LM) and general linear models (GLM). Below is a brief description of how GA works:

1. Produce population of individuals (regression models)
    - Abinary matrix is randomly generated, with proposed independent variables as the column names.
    - Each row represents one individual (chromosome/model). If 1 is in the column of a particular column, the corresponding independent variable is used in the regression model. Otherwise, it is excluded from the model.
2. Evaluate the fitness of all individuals
    - Evaluate the fitness of each individual using the fitness function, such as calculating the AIC or using other criteria specified by the user.
3. Select the most fit individuals for reproduction (Natural Selection)
    - Rank the individuals according to the criteria used in the fitness function, and assign corresponding breeding probability to each individual (model).
    - Higher ranked individuals are assigned higher probability.
    - Sum of probability of all individual equals to 1.
4. Crossover parent chromosomes (Genetic Inheritance)
    - Crossover of each chromosomes is performed according to the probability of crossover.
    - If crossover occurs, the location of crossover is randomly selected along one of the chromosomes.
    - Resulting population becomes more genetically diverse than the original one.
5. Mutate resulting chromosomes (Genetic Inheritance)

- For each locus on the chromosome, we allow a small probability of error as determined by the mutation probability.
- The 1 in a particular column might be mutated to a 0 or vice versa. This causes movement in the search place, escaping local maxima and restores lost information to the population, promoting search diversification.

6. Evaluate the fitness of resulting offspring population
   - Using the same fitness function as described in Step 2, the resulting offspring population (crossed over and mutated chromosomes) are also evaluated and ranked.
7. Generate a new population
   - The original parent population is replaced either fully or partially by the offspring population, determined by the replacement probability.
   - The new population now becomes the new parent population for the next generation.
8. Step 3 to 7 are repeated until the termination condition is met.
   - Termination condition is determined by the user, and is usually referred to the condition when the AIC or the fitness criteria used converges to its limit after certain number of generations.
   - When the AIC of fitness criteria value is plotted on a graph, it is typical to see the starting individuals coalesce into a few effective subspecies, with the best overwhelming the rest.

In this case, as each chromosome/individual represents one regression model, the starting population of models will eventually converge to the best model containing the ideal independent variables, achieving the goal of variable selection for regression models.

Code Structure

We decided to build our code in a sequential way following the logic of the Given and Hoeting article [2]. In fact, the article does a great job at identifying every step required by the genetic algorithm. Our main function is the select() function that only calls other functions that we will call auxiliary functions. By naming these auxiliary steps, we defined those functions that, for a given generation, perform the genetic algorithm to move on to the next generation. Hence, the select function only consist of a for loop calling those auxiliary functions. Lets go through these auxiliary functions we have defined. Please see the appendix for full code.

- *ReadDataFile*
  Reads the data and builds a list of two data frames. The first element is the original data and the second element is the modified data where the user can remove columns that are not of interest for the variable selection. For instance, in our car data set, the first column has been removed as it contained the names and models of the cars.
- *Initiate*
  Creates a random initial population of the size specified by the user.
- *Fitness*
  In order to be able to later rank our various individuals, we need to calculate a metric; the fitness function. According to the theory developed in the article, we use the AIC or BIC as a fitness function. Liberty is given to the user to provide these as input parameter.
- *Run Model*
  Recursively call the fitness function on all the possible models that our population represents.
- *Crossover\Mutation*
  These two functions simply perform the named task on the population of interest.

- *Breeding*
  Breeding calls another auxiliary function matching that matches parents together. Our choice was to use a rank-based matching probability in order to make sure that an excessively low fitness function individual would not spread too rapidly within the population. With this auxiliary function, we generate a new population of individuals by matching individuals and by operating crossover.
- *Replacement*
  A new offspring population is then obtained by choosing the replacement percentage between the original parent population and the newly generated offspring population. It is left to the user to decide whether he prefers his genetic algorithm to be solely based on the offspring population or if some trade-off is to be kept with the previous parent population.
- *Testing*
  Testing functions use a set of parameters to test the various functionalities of the main select function.
- *Plotting*
  Plot tests in order to visualize convergence and fitness of models.

Testing and Outcomes

Throughout the process of writing and refining the code for our algorithm, we were performing multiple tests, both to ensure the proper performance of individual functions, as well as to confirm the correctness of the results. Once our code was complete, we tested the *select* function using the *cars* dataset, part of R's *datasets* package.

We were pleased with the results of our testing, as well as the efficiency in terms of system processing time. For an initial population of 50 chromosomes, and 50 generations, our tested LM processed and provided a final best model based on AIC in 13.42 seconds. For the same number of chromosomes and generations, our tested GLM processed and provided a final best model based on BIC in 14.1 seconds. While there is indeed room for improved efficiency, we see this as a favorable outcome.

First Test

We first tested our algorithm on a linear model using AIC as our fitness criteria. In R, a lower AIC indicates a more favorable model. We began with four predictors, VOL, SP, WT, and HP, and the dependent variable, MPG. Our algorithm predicted the best model to be MPG regressed on SP, WT, and HP, thus dropping the variable VOL and returning an AIC of 450.5048. Running a stepwise AIC on the data using R's *stepAIC* function, we received the same results. The AIC was lower using R's built-in function (215.8), but the selected model is the outcome of primary interest.

Using stepAIC, the model converged after just two generations, as did our model. Note that these results would likely be increased significantly if using a model with a large initial number of predictors.

Second Test

For comparative purposes, we used the same data to test our algorithm on a general linear model using BIC as our fitness criteria. Again, our algorithm predicted the best model to be MPG regressed on SP, WT, and HP, returning an BIC of 462.54. Again running a stepwise AIC on the data but using a k-value of $log(n)$, with n as the number of rows in the dataset (in order to produce BIC), we received nearly identical results, with a BIC of 450.4.

Using R's built-in function, the model again converged after just two generations, while our model converged after four.

Plots of the convergence results from running our algorithm sre displayed below:

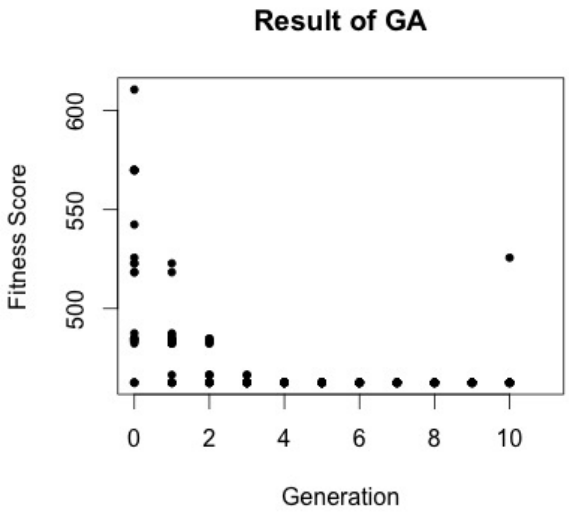

Figure 1: LM using AIC

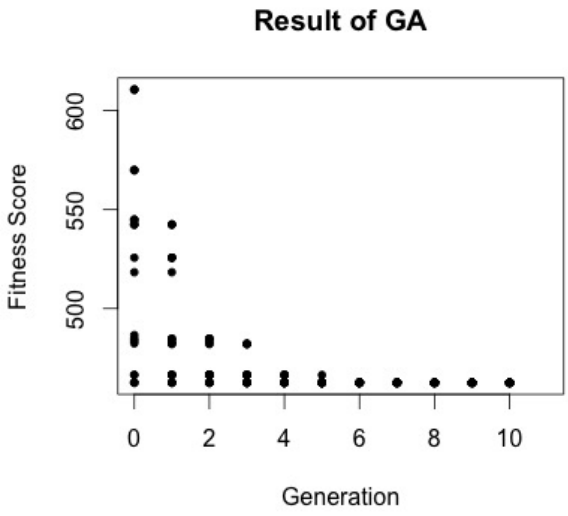

Figure 2: GLM using BIC

Contributions

Contributions to the planning, exuction, and completion of this project were balanced and equitably distributed among the team members. We all individually researched, wrote and tested the code, and wrote the report and documentation. Much of our code was written in a group programming setting, however. A specific breakdown of contributions is as follows:

Florian Papion

R Functions :

- Crossover function
- Breeding function (and helper functions)

Documentation : Written for above functions

Report Sections :

- Code Structure

Qiuhan Wu

R Functions :

- Initiation function
- Mutation function
- Replacement function

Documentation : Written for above functions

Report Sections :

- Introduction and Overview of Model Selection

Brianna Noland

R functions :

- Reading function
- Fitness function (and helper functions)
- Testing function

Documentation : Written for above functions

Report Sections :

- Testing and Outcomes

References

[1] Emanuel Falkenauer *Genetic Algorithms and Groupin Problems* 1998: John Wiley and Sons, England

[2] Geof H. Givens and Jennifer A. Hoeting *Computational Statistics* 2013: John Wiley and Sons, New Jersey. Second Edition. Random House, N.Y.

[3] "Wikipedia" *Genetic algorithm — Wikipedia, The Free Encyclopedia"* 2014: `http://en.wikipedia.org/w/index.php?title=Genetic_algorithm&oldid=633723560`