Stat243: Final group project
Due Friday Dec. 17, 5 pm

December 8, 2021

The project will be done in groups of three, with students assigned randomly.

A few comments. First, the project, when split amongst the group members, is not intended to be a huge undertaking. The goals of the project are to give you experience in working collaboratively and developing a well-designed, well-tested piece of software.

The project will be graded as a letter grade and will count for about as much as two problem sets in your final grade.

Please use standard citation practices to cite any papers/online reasources/people whose ideas you make use of. Please do not consult with class members who are not in your group.

Problem

Your task is to make an R package to implement a genetic algorithm for variable selection in regression problems, including both linear regression and GLMs. Some details on genetic algorithms are available in Section 3.4 of the Givens and Hoeting book on Computational Statistics - see the electronic book available through UC Library search, or the PDF of Chapter 3 in the *project* directory of the class GitHub repo. You should also be able to find plenty of other information about genetic algorithms online.

1. Your solution should allow the user to provide reasonable inputs, in terms of specifying a dataset and the type of regression. Much of this is information you should just be able to pass along to *lm()* or *glm()*. By default you should just use AIC as your objective criterion but allow users to provide their own objective function. Similarly you can use the genetic operators described in Givens and Hoeting for variable selection, but ideally your code should be general enough that a user could provide additional operators.

2. Your solution should involve modular code, with functions or OOP methods that implement discrete tasks. You should have an overall design and style that is consistent across the components, in terms of functions vs. OOP methods, naming of objects, etc.

3. In terms of efficiency, the generations are inherently sequential. However, you should try to vectorize as much as possible. If you like, you can allow for parallel processing on a single machine when working with the population in a given generation, in particular the evaluation of the fitness function, but it is not required.

4. Show the results of using your implementation on two or more examples. Have the use of the package on the examples be coded in the example section of the R help for your main function.

5. Formal testing is required, with a set of tests where results are compared to some known truth. You should have tests for the overall function and any modules that do anything complicated. For testing the overall function, since the algorithm is stochastic, you'll need to think carefully about how to set this up. You should also have unit tests for individual functions that carry out the individual computations that make up the algorithm. See more details below on how to set up the tests. There is more information on testing here: https://r-pkgs.had.co.nz/tests.html.

6. You should be writing your own code for essentially everything except the model fitting and other standard functionality available in base R and the R packages such as *stats* that are provided with the standard R installation. If you'd like to use any other code or packages, please consult with me first.

Formatting requirements and additional information

1. Your solution to the problem should have two parts:

   (a) An R package named *GA*, including the .tar.gz file created by `R CMD build GA`. I should be able to install your package by simply running `R CMD INSTALL GA_version.tar.gz` or `install_github(paste0(username, '/GA'))` where *username* contains the GitHub user name of the group member in whose GitHub account the project resides. A good starting place for information about R packages is Hadley Wickham's book: https://r-pkgs.had.co.nz/. You can use *usethis::create_package* to create the initial set of directories for the package.

   The package should include:

   i. A primary function called *select* that carries out the variable selection, located in a file *select.R* in the *R* directory of the package, and including appropriate code comments.

   ii. Other supporting code in additional files in the *R* directory of the package, including appropriate code comments

   iii. Formal tests, located in the package. You can set this up with *usethis::use_testthat*; however, I don't care about the exact structure of the tests folder in your R package, so long as I can run `testthat::test_package('GA')` or some standard invocation that you give me and have it run all your tests. Please check that you can run the tests successfully when you install your package outside of the context in which you are doing your development (e.g., on the SCF).

   iv. Help information for the main function, in the form of standard R documentation in a file called *select.Rd* in the *man* directory. You can either write *select.Rd* by hand or you can have it generated based on using the *roxygen2* package (see https://adv-r.had.co.nz/Documenting-functions.html for an example), with the documentation included in the R code files. You do not need help pages for your auxiliary functions.

   (b) A PDF document describing your solution, prepared in R Markdown or LaTeX+knitr. The description does not need to be more than 2-4 pages, but should describe the approach you took in terms of functions/modularity/object-oriented programming, the testing that you carried out, and the results of applying the implementation to the example(s). It must include a paragraph describing the specific contributions of each team member and which person/people were responsible for each component of the work. **Please submit a paper copy of the document to me - either directly to me, under my door, or in my mailbox. On your paper solution, please indicate the GitHub user name of the group member in whose Github repository the final version of the project resides.**

2. You should start the process by mapping out the modular components you need to write and how they will fit together, as well as what the primary function will do. After one person writes a component, another person on the team should test it and, with the original coder, improve it. You could also consider using pair programming for some of your development.

3. You should use Git and GitHub to manage your collaboration, with branching as needed and regular commits. Please have the project be a repository named (exactly) *GA* within the github.com account of one of the project members. Please make the repository private and share it with me (user 'paciorek' on GitHub.com).