# STAT 243 Final Project: Genetic Algorithm for Model Selection

.....Chris Gagne

Dec 15 2017

## Description of Algorithm and Approach to Modularization

Our package has one main function select(). This does the entire genetic algorithm on a dataset containing rows as examples and columns as variables.

Our approach to modularization was break the genetic algorithm into its natural processes. (1) initialize population of chromosomes initialize(), (2) calculation of fitness fitness(), (3) selection of parents for breeding selection(), (4) breeding parents and choosing the next generation nextGeneration(). Within this function, we use crossover() to do the actual breeding and mutation().

We allow changes to the general algorithm in a number of ways:

We implemented 3 ways to select the parent generation. The first method was to use fitness-rank-based probabilities to select both parents in the pairs chosen to breed. The second method used fitness-rank-based probabilities to choose just one parent in each pair, and the second parent was chosen randomly. This might help preserved diversity in population, which is important for not converging to local minimum. The third method is tournament selection, which FILL IN HERE. These methods can be toggled between by the user by using the K parameter.

We also implemented a flexible generation gap (the percent of children replacing the parents in each generation) as well as allowed for a user specified number of generations and starting population size.

In terms of represenation, our chromosomes are boolean vectors which are used to index an X-matrix, with columns for each predictor variables. The population set of chromosomes is a matrix of the boolean chromosome vectors and is passed into various functions. nextGeneration() returns a modified version of the population set of chromosomes. We did this, rather than keep old generation chromosomes, to save memory. We do, however, store the fitness scores for

each model for each generation that we use to plot the progress of the algorithm (see examples).

## Installation

Our package can be installed by either downloading and setting the working directory and installing or installing directly from github.

```
install('GA')
# or git_install('GA')
```

The package is kept in XYZ's github repo here: The username for the repo is:

## Tests

Testing

```
library('testthat')
test('GA')
```

## Example 1:

Here is a basic example of how to use our package.

```
library('GA')
dat<-mtcars # specifify a data set
# as the documentation points out,
# the first column needs to by y and
# the rest of the columns are possible predictors

results<-select(dat) # fit

## [1] "Generation:  10  Fitness:  156.652338825641"
## [1] "Generation:  20  Fitness:  155.476628510258"
## [1] "Generation:  30  Fitness:  155.476628510258"
## [1] "Generation:  40  Fitness:  155.476628510258"
## [1] "Generation:  50  Fitness:  155.476628510258"
## [1] "Generation:  60  Fitness:  155.476628510258"
## [1] "Generation:  70  Fitness:  155.476628510258"
## [1] "Generation:  80  Fitness:  155.476628510258"
```

```
## [1] "Generation:  90  Fitness:  155.476628510258"
## [1] "Generation:  100  Fitness:  155.476628510258"
## [1] "ALgorithm Ends"
```

We can inspect the best fitted model.

```
# inspect fittest model
summary(results$fittestModel)

##
## Call:
## model(formula = form, data = dat)
##
## Residuals:
##     Min      1Q  Median      3Q     Max
## -3.9290 -1.5598 -0.5311  1.1850  5.8986
##
## Coefficients:
##             Estimate Std. Error t value Pr(>|t|)
## (Intercept) 38.75179    1.78686  21.687  < 2e-16 ***
## cyl         -0.94162    0.55092  -1.709 0.098480 .
## hp          -0.01804    0.01188  -1.519 0.140015
## wt          -3.16697    0.74058  -4.276 0.000199 ***
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## Residual standard error: 2.512 on 28 degrees of freedom
## Multiple R-squared:  0.8431,Adjusted R-squared:  0.8263
## F-statistic: 50.17 on 3 and 28 DF,  p-value: 2.184e-11
```

As well as see the fitness of the chromosomes as the algorithm proceeds.

```
plotGA(results)

## Error in plotGA(results):  could not find function "plotGA"
```

## Example Comparison against forward/backward selection

Backwards selection chooses the best model containing regressors for (wt, qseq, am). This seems to be the best model - as it's the lowest we've seen in any

iteration of our algorithm. Note that forward selection does not achieve this model.

```
# Comparing against backwards selection
full=lm(dat[,1]~., data=dat[,-1])
step(full, direction="backward",trace=FALSE)


##
## Call:
## lm(formula = dat[, 1] ~ wt + qsec + am, data = dat[, -1])
##
## Coefficients:
## (Intercept)           wt         qsec           am
##       9.618       -3.917        1.226        2.936
```

## Example User Inputs

Here, we show different parameters that we can specifiy for our algorithm. It obtains the fittest model (wt,qsec,am).

```
library('stats')
library('GA')
#function(fit,...){return(extractAIC(fit,...)[2])}
results<-select(dat=mtcars,
                P=50,
                model=glm,
                numGens=100,
                G=0.25,
                method=2,
                K=2,
                fitnessFunction=AIC
                )


## [1] "Generation:  10  Fitness:  154.119370868901"
## [1] "Generation:  20  Fitness:  154.119370868901"
## [1] "Generation:  30  Fitness:  154.119370868901"
## [1] "Generation:  40  Fitness:  154.119370868901"
## [1] "Generation:  50  Fitness:  154.119370868901"
## [1] "Generation:  60  Fitness:  154.119370868901"
## [1] "Generation:  70  Fitness:  154.119370868901"
## [1] "Generation:  80  Fitness:  154.119370868901"
## [1] "Generation:  90  Fitness:  154.119370868901"
## [1] "Generation:  100  Fitness:  154.119370868901"
## [1] "ALgorithm Ends"
```

```
results$fittestModel

##
## Call:  model(formula = form, data = dat)
##
## Coefficients:
## (Intercept)           wt         qsec           am
##       9.618       -3.917        1.226        2.936
##
## Degrees of Freedom: 31 Total (i.e. Null);  28 Residual
## Null Deviance:     1126
## Residual Deviance: 169.3  AIC: 154.1
```

We can plot this as well

```
plotGA(results)

## Error in plotGA(results):  could not find function "plotGA"
```

## Example Early Convergence

A big difference in our algorithm is the generation Gap. With too high of a generation gap, the algorithm often converges to a local minimum. We would consider adding mutation rate as a flexible parameter in the future.

```
library('stats')

results<-select(dat=mtcars,
                P=50,
                model=glm,
                numGens=100,
                G=0.76,
                method=2,
                K=2,
                fitnessFunction=function(fit,...){return(extractAIC(fit,...)[2])}
                )

## function (formula, family = gaussian, data, weights, subset,
##     na.action, start = NULL, etastart, mustart, offset, control = list(...),
##     model = TRUE, method = "glm.fit", x = FALSE, y = TRUE, contrasts = NULL,
##     ...)
## {
```

```
##     call <- match.call()
##     if (is.character(family))
##         family <- get(family, mode = "function", envir = parent.frame())
##     if (is.function(family))
##         family <- family()
##     if (is.null(family$family)) {
##         print(family)
##         stop("'family' not recognized")
##     }
##     if (missing(data))
##         data <- environment(formula)
##     mf <- match.call(expand.dots = FALSE)
##     m <- match(c("formula", "data", "subset", "weights", "na.action",
##         "etastart", "mustart", "offset"), names(mf), 0L)
##     mf <- mf[c(1L, m)]
##     mf$drop.unused.levels <- TRUE
##     mf[[1L]] <- quote(stats::model.frame)
##     mf <- eval(mf, parent.frame())
##     if (identical(method, "model.frame"))
##         return(mf)
##     if (!is.character(method) && !is.function(method))
##         stop("invalid 'method' argument")
##     if (identical(method, "glm.fit"))
##         control <- do.call("glm.control", control)
##     mt <- attr(mf, "terms")
##     Y <- model.response(mf, "any")
##     if (length(dim(Y)) == 1L) {
##         nm <- rownames(Y)
##         dim(Y) <- NULL
##         if (!is.null(nm))
##             names(Y) <- nm
##     }
##     X <- if (!is.empty.model(mt))
##         model.matrix(mt, mf, contrasts)
##     else matrix(, NROW(Y), 0L)
##     weights <- as.vector(model.weights(mf))
##     if (!is.null(weights) && !is.numeric(weights))
##         stop("'weights' must be a numeric vector")
##     if (!is.null(weights) && any(weights < 0))
##         stop("negative weights not allowed")
##     offset <- as.vector(model.offset(mf))
##     if (!is.null(offset)) {
##         if (length(offset) != NROW(Y))
##             stop(gettextf("number of offsets is %d should equal %d (number of observatio
##                 length(offset), NROW(Y)), domain = NA)
```

```
##     }
##     mustart <- model.extract(mf, "mustart")
##     etastart <- model.extract(mf, "etastart")
##     fit <- eval(call(if (is.function(method)) "method" else method,
##         x = X, y = Y, weights = weights, start = start, etastart = etastart,
##         mustart = mustart, offset = offset, family = family,
##         control = control, intercept = attr(mt, "intercept") >
##             0L))
##     if (length(offset) && attr(mt, "intercept") > 0L) {
##         fit2 <- eval(call(if (is.function(method)) "method" else method,
##             x = X[, "(Intercept)", drop = FALSE], y = Y, weights = weights,
##             offset = offset, family = family, control = control,
##             intercept = TRUE))
##         if (!fit2$converged)
##             warning("fitting to calculate the null deviance did not converge -- increase
##         fit$null.deviance <- fit2$deviance
##     }
##     if (model)
##         fit$model <- mf
##     fit$na.action <- attr(mf, "na.action")
##     if (x)
##         fit$x <- X
##     if (!y)
##         fit$y <- NULL
##     fit <- c(fit, list(call = call, formula = formula, terms = mt,
##         data = data, offset = offset, control = control, method = method,
##         contrasts = attr(X, "contrasts"), xlevels = .getXlevels(mt,
##             mf)))
##     class(fit) <- c(fit$class, c("glm", "lm"))
##     fit
## }
## <bytecode: 0x7ffad1042b88>
## <environment: namespace:stats>

## Error in initialization(C = C, P = P): could not find function "initialization"

results$fittestModel

## Error in eval(expr, envir, enclos):  object 'results' not found
```

We can plot this as well

```
plotGA(results)

## Error in plotGA(results):  could not find function "plotGA"
```

## Contributions of Team Members

- design of algorithm (all members) - wrapper function (Z,C) - selection ... - package organization (chris) - tests ... - pdf document...