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.

Check out the documentation by running ?select.

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

Tests

Testing

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

Example 1:

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

```
dat<-mtcars # specifify a data set
# notes: the first column needs to by y
# the rest of the columns are possible predictors

# FIX
setwd("~/Desktop/GA/R")
source('select.R')

results<-select(dat) # fit

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

We can inspect the best fitted model.

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

## Error in summary(results$fittest_model):  object 'results' not found
```

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

```
i<-nrow(results$fitnessScores)

## Error in nrow(results$fitnessScores):  object 'results' not found

g<-ncol(results$fitnessScores)

## Error in ncol(results$fitnessScores):  object 'results' not found

X<-matrix(rep(seq(g),i,g),nrow=i,ncol=g)

## Error in seq(g):  object 'g' not found

Y<-t(results$fitnessScores)

## Error in t(results$fitnessScores):  object 'results' not found

plot(X,Y,xlab='generation',ylab='AIC',pch=19,cex=0.5)

## Error in plot(X, Y, xlab = "generation", ylab = "AIC", pch = 19,
cex = 0.5):  object 'X' not found

lines(seq(g),apply(results$fitnessScores,FUN=min,MARGIN=2),lty=1,col='green')

## Error in seq(g):  object 'g' not found
```

Example User Inputs

Here, we show different parameters that we can specifiy for our algorithm.

```
library('stats')

dat<-mtcars
P<-100 # population size
mod<-glm # model type
```

```
numGens<-100
G<-0.75 # generation gap (replace 75% of parents with children)
K<-2 # which method to use for parent selection  ### Fix documentation for this ##
f<-function(fit,...){return(extractAIC(fit,...)[2])} # doesn't seem to be working **
results<-select(dat,P=P,mod=mod,numGens=numGens,G=G,K=K,fitnessFunction=f)

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

We can plot this as well

```
i<-nrow(results$fitnessScores)

## Error in nrow(results$fitnessScores):  object 'results' not found

g<-ncol(results$fitnessScores)

## Error in ncol(results$fitnessScores):  object 'results' not found

X<-matrix(rep(seq(g),i,g),nrow=i,ncol=g)

## Error in seq(g):  object 'g' not found

Y<-t(results$fitnessScores)

## Error in t(results$fitnessScores):  object 'results' not found

plot(X,Y,xlab='generation',ylab='AIC',pch=19,cex=0.5)

## Error in plot(X, Y, xlab = "generation", ylab = "AIC", pch = 19,
cex = 0.5):  object 'X' not found

lines(seq(g),apply(results$fitnessScores,FUN=min,MARGIN=2),lty=1,col='green')

## Error in seq(g):  object 'g' not found
```

Example Comparison against forward/backward selection

Forward selection chooses:..

```r
# Comparing against forward selection
null=lm(dat[,1]~1, data=dat[,-1])
full=lm(dat[,1]~., data=dat[,-1])
step(null, scope=list(lower=null, upper=full), direction="forward",trace=FALSE)


##
## Call:
## lm(formula = dat[, 1] ~ wt + cyl + hp, data = dat[, -1])
##
## Coefficients:
## (Intercept)           wt          cyl           hp
##    38.75179     -3.16697     -0.94162     -0.01804
```

Backwards selection chooses the best model containing regressors for (wt, qseq, am)

```r
# Comparing against backwards selection
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
```