Package 'GA'

January 12, 2022

Type Package

Title On Multivariate Feature Selection with Genetic Algorithms

Version 0.1.1

Author Andrew Ma, James Hall, and Sky Qui

Maintainer `<andrew.c.ma@berkeley.edu>`

Description This package provides a genetic algorithm to identify optimal features
for linear regression under R's LM and
GLM models. Users are able to set
multiple parameters for the genetic
algorithm and request optimization via
R2, AIC, BIC, AICc or a user defined
function. The algorithm can process
these score functions in parallel
on a single machine with the
parallel = TRUE argument.

License MIT + file LICENSE

Encoding UTF-8

LazyData true

Imports assertthat,
stats,
testthat

RoxygenNote 7.1.2

Config/testthat/edition 3

R topics documented:

apply_elitism — *Processes the Highest Fitness Creatures to ensure that they Make it into the Next Generation*

Description

Since the primary selection mechanisms of the genetic algorithm are probabilistic, there is a chance that a high performing gene goes extinct causing the gene pool to become less fit. This function counteracts this by ensuring that the best performing creatures survive into future generations. This function selects a number of most fit creatures equal to the ceiling(pop * elite_prop) and guarantees that they make it to the next generation. Additionally, this function makes a copy of each most-fit creature and conducts one gene mutation randomly on each creature. If the copy is more fit than the original, it is returned instead of the original so that the final returned matrix of elite creatures are at least as fit as the incoming elite creatures.

Usage

```
apply_elitism(
  data,
  gene,
  score_vec,
  elite_prop = 0.1,
  mutate = 0.1,
  metric = "AIC",
  family = "gaussian",
  custom_function = NULL,
  fittest
)
```

Arguments

`data`	a matrix or dataframe with ncol = gene_length plus 1
`gene`	matrix with all 0s and 1s with ncol = gene_length and nrow = pop
`score_vec`	a positive, numeric vector with length equal to the total population
`elite_prop`	a numeric, the percentage of the population that is selected for elitism
`mutate`	percentage mutation rate
`metric`	a character, a user specified statistic such as R2, AIC, BIC, or AICc
`family`	model family corresponding to GLM. Defaults to 'gaussian'
`custom_function`	defaults to NULL, if defined takes a single row from a generation_matrix and returns a numeric value
`fittest`	whether a custom function has the highest value corresponding to the fittest or the lowest value

Value

a matrix containing all 1s and 0s representing the genes of the elite population

Examples

```
data <- matrix(rnorm(11*3),nc=11)
gene <- matrix(rbinom(10*10,1,.5),nc=10)
score_vec <-rnorm(10)
elite_prop <- .05

elite_gen <- apply_elitism(score_vec = score_vec,gene = gene,data = data,elite_prop = elite_prop)
```

`breed_next_gen` *Breeds Next Generation*

Description

The purpose of this function is to take a vector of new parents, their corresponding fitness score vector and their corresponding genes to output the next generation in the genetic algorithm. It does this via ' several crossover methods including uniform, k-point, fitness-uniform along with their corresponding parameters. After Crossover is complete mutation is applied in two different methods either adaptive which can increase mutation when diversity becomes too low and fixed which maintains a steady mutation rate. Lastly, there is an option to minimize inbreeding prior to crossover and mutation.

Usage

```
breed_next_gen(
  generation_matrix,
  new_parents,
  score_vec,
  number_of_parents,
  mutation = "fixed",
  minimize_inbreeding = FALSE,
  crossover = "uniform",
  mutation_rate = 0.01,
  ad_max_mutate = 0.1,
  ad_min_mutate = 0.01,
  ad_inflection = 0.3,
  ad_curve = 15,
  required_pop,
  number_of_crossovers = 1
)
```

Arguments

`generation_matrix`	matrix with ncol = gene_length and nrow = total number of creatures, consists only of 1s and 0s, no all-zero rows.
`new_parents`	a numeric vector with elements indexing the next generation of parents, length is total population x number_of_parents.
`score_vec`	a numeric vector with length equal to total population.
`number_of_parents`	is an integer defining the number of parents per offspring

`mutation`	a character either 'fixed' or 'adaptive' indicating the type of mutation
`minimize_inbreeding`	a logical indicating whether or not we minimize inbreeding.
`crossover`	a character that can be 'uniform', 'fitness', or 'k_point' indicating how to carry out crossover.
`mutation_rate`	a numeric between 0 and 1 indicating how often a creature is selected for mutation.
`ad_max_mutate`	max mutation rate for adaptive mutation, numeric between 0 and 1 and more than ad_min
`ad_min_mutate`	min mutation rate for adaptive mutation, numeric between 0 and 1 and less than ad_max
`ad_inflection`	percentage of diversity in population where adaptive mutation begins to increase rapidly
`ad_curve`	rate that influence how intensely adaptive mutation changes
`required_pop`	the number of offspring required
`number_of_crossovers`	number of k_point crossovers, needs to be less than gene_length

Details

crossover Methods Uniform - each gene is randomly selected from 2 or more parents from a PMF proportional to the number of parents.

Fitness - each gene is randomly selected from 2 or more parents from a PMF proportional to the parent's fitness.

K-Point - parents genes are broken into k+1 segments, then the offspring inherits portions randomly from the parents. Takes the parameter number_of_crossovers which must be less than 1/2 gene_length.

For all methods candidate offspring are accepted / rejected so that they don't have completely 0-vector genes.

mutation Fixed - Each offspring has a mutation_rate chance of being selected for mutation. Once selected one gene is switched form one to zero or zero to one.

Adaptive - the overall population is measured for diversity. As diversity becomes lower, the mutation rate increases. Once selected for mutation a single gene is switched from one to zero or zero to one. The adaptive function is controlled by a simple logistic function with parameters ad_min and ad_max describing the minimum and maximum mutation rates. ad_inflection controls where the logistics point pivots, and ad_curve controls how rapidly the logistics curve increase.

For all mutation methods, candidate offspring are accepted / rejected so that they don't have completely 0-vector genes.

Minimize inbreeding. his option reduces (though does not remove) the chance of similar creatures creating offspring together. Each parent is assigned new partner(s) randomly from a PMF proportional to how different their genes are. Parents are drawn without replacement so that if they were selected to become parents, they will still remain parents.

Value

returns matrix of 1s and 0s with no all-zero rows representing the genes of the next generation after crossover and mutation

Examples

```
generation_matrix <- matrix(rbinom(10*10,1,.5),nc=10)
new_parents <- sample(1:10,20,replace = TRUE)
mutation <- 'fixed'
crossover <- 'uniform'
number_of_parents <- 2
required_pop <- 10
score_vec <-abs(rnorm(10))

next_generation_matrix <- breed_next_gen(generation_matrix = generation_matrix,
new_parents = new_parents, mutation = mutation, crossover = crossover,
 number_of_parents = number_of_parents,required_pop = required_pop,
 score_vec = score_vec)
```

`create_initial_generation`
Creates the Initial Gene Pool for Genetic Algorithm

Description

This function creates the initial population to be used in the genetic algorithm. It checks to make sure there are now 0-vectors because 0-vectors are meaningless in the context of LM and will cause LM to fail. Function allows users to specify number of population gene length, probability of genes being represented and specify initial population.

Usage

```
create_initial_generation(pop, gene_length, prob = 0.5, user_genes = NULL)
```

Arguments

`pop`	an integer representing the total population maintained for each generation of the genetic algorithm
`gene_length`	an integer defining how many genes are encoded for each creature, needs to be the same length as the number of columns in the independent variable matrix
`prob`	a number 0-1 representing the chance that an individual has any given gene
`user_genes`	a matrix consisting of 1s and 0s without any rows with all zeros ncol equal to gene_length and nrow less than or equal to pop

Value

returns a matrix of 1s and 0s without any rows with all zeros representing the initial genes for the first generation of the genetic algorithm.

Examples

```
pop <- 500
gene_length <- 50
prob <-.25

generation_matrix <- create_initial_generation(pop = pop, gene_length = gene_length,prob = prob)
```

`generate_data` *Generates normal data with 50 Covariates.*

Description

This function creates a matrix that is 51 x 1000 where the first column is the response variables and the remain 50 are independent variables. The covariates are generated uniform(0,100) The parameter for each covariate is defined such that the first one is 20x0.9e1 followed by 20x0.9e2 with each subsequent parameter decaying until the last one is 20x0.9e50. This allows us to validate a genetic algorithm while generating a large dataset

Usage

```
generate_data()
```

Value

a 1000x51 matrix

Examples

```
generate_data()
```

`score_fitness2` *Scores fitness for an entire generation of creatures*

Description

This takes a generation_matrix representing all the genes of a single generation, the data, and user input describing the metric such as R2, AIC, BIC, and AICC and can even use a custom function defined by the user. The underlying fitness function for R2 is lm() and the underlying fitness function for AIC, BIC, and AICc is glm(). AIC, BIC, and AICC has the option to specify a specific family of functions however, the data needs to be defined across the support for the given family for instance exponential must be greater than 0.

Usage

```
score_fitness2(
  gene,
  data,
  metric = "AIC",
  family = "gaussian",
  custom_function = NULL,
  fittest = "high"
)
```

Arguments

`gene`	matrix with all 0s and 1s with ncol = gene_length and nrow = pop
`data`	a matrix or dataframe with ncol = gene_length plus 1
`metric`	a character, a user specified statistic such as R2, AIC, BIC, or AICc
`family`	, model family corresponding to GLM. Defaults to 'gaussian'
`custom_function`	defaults to NULL, if defined takes a single row from a generation_matrix and returns a numeric value
`fittest`	whether a custom function has the highest value corresponding to the fittest or the lowest value

Value

returns a list with the first element being a vector of processed fitness scores to for continued use by the algorithm and element 2 being true (user specified) values user reference, for instance true AIC values

Examples

```
gene <- matrix(rbinom(2*10,1,.5),nc=10)
data <- matrix(rnorm(11*3),nc=11)
pop_required <- 10
metric <- 'AIC'

score_vec <- score_fitness2(data = data,gene = gene,metric = metric)
```

`see_if_terminate` *Tests if early termination conditions are met.*

Description

This function defines additional termination criteria which will run in addition to the max_iteration. Default is null for all values. User can specify a percentage_convergence such that when diversity falls below a specific threshold the program terminates. User can identify a estimate (from the standard summary() function that if the estimate reaches a specific threshold the program terminates or if the estimate pauses for a certain number of iterations terminates.

Usage

```
see_if_terminate(
  generation_matrix,
  summary_data_frame,
  estimator = NULL,
  pause_length = NULL,
  score_threshold = NULL,
  percent_converge = NULL,
  iteration,
  metric = "AIC",
  fittest = "high"
)
```

Arguments

`generation_matrix`	generation matrix with all 0s and 1s with ncol = gene_length and nrow = pop
`summary_data_frame`	a matrix that captures the summary() data for each generation's scores
`estimator`	character string referencing the named entry from summary() for instance Min.
`pause_length`	a numeric indicating how many iterations without improvement in the estimator before you terminate
`score_threshold`	a numeric indicating a threshold to cutoff if the estimator reaches that value
`percent_converge`	percentage of diversity when algorithm terminates for instance .1 will terminate if there are only .90 of the population is represented by the same genes
`iteration`	is the current iteration the main algorithm is on
`metric`	is the user specified statistic such as R2, AIC, BIC, or AICc
`fittest`	whether a custom function has the highest value corresponding to the fittest or the lowest value

Details

summary_data_frame - is imported from the main function and updated each iteration

estimator - cam be nominated from the summary() function aka Min., 1st Qu., Median, Mean, 3rd Qu., or Max.

percent_converge - terminates once diversity falls below a certain percentage. diversity is sum(unique(genes))/total_genes

Value

TRUE if terminate conditions are met with a reason, FALSE otherwise

Examples

```
generation_matrix <- matrix(rbinom(2*10,1,.5),ncol=10)
summary_data_frame <- data.frame(matrix(1:12,ncol=6))
names(summary_data_frame) <-c("Min.","1st Qu.","Median","Mean","3rd Qu.","Max.")
estimator <- 'Mean'
score_threshold <- 2.4
iteration <- 2

confirm_terminate <- see_if_terminate(generation_matrix,
summary_data_frame,estimator = 'Mean',
score_threshold = score_threshold, iteration = iteration)
```

`select` *Select - Runs a Genetic Algorithm*

Description

This function optimizes a linear regression using a genetic algorithm. User can enter a custom function or select a metric from (AIC, BIC, AICC, R2) to optimize features to include in the linear regression. This function has a broad range of genetic algorithm features including multiple parent section methods, crossover options, mutation options and other features such as elitism, minimizing-inbreeding, using more than two parents and selecting from a range of early termination options. See below for details)

Usage

```
select(
  total_number_generations,
  number_of_parents = 2,
  pop,
  gene_length,
  response_vec,
  independent_vars,
  prob = 0.05,
  user_genes = NULL,
  metric = "AIC",
  family = "gaussian",
  method = "roulette",
  susN = 1,
  tourn_size = 2,
  mutation = "fixed",
  mutation_rate = 0.02,
  minimize_inbreeding = FALSE,
  crossover = "uniform",
  number_of_crossovers = 1,
  elitism = TRUE,
  elite_prop = 0.05,
  ad_max_mutate = 0.1,
  ad_min_mutate = 0.01,
  ad_inflection = 0.3,
  ad_curve = 15,
  custom_function = NULL,
  estimator = NULL,
  pause_length = NULL,
  percent_converge = NULL,
  score_threshold = NULL,
  fittest = "high"
)
```

Arguments

`total_number_generations`	an integer representing the maximum number of iterations

`number_of_parents`	an integer defining the number of parents per offspring
`pop`	an integer defining the number of creatures per a generation
`gene_length`	an integer representing the number of genes per a creature
`response_vec`	a numeric vector without infinite or NA values
`independent_vars`	a matrix or dataframe with ncol equal to number of genes and nrow with the same length as the response_vec
`prob`	a number 0-1 representing the chance that an individual has any given gene
`user_genes`	a matrix consisting of 1s and 0s without any rows with all zeros ncol equal to gene_length and nrow less than or equal to pop
`metric`	a character from the following 'R2', 'AIC', 'BIC', or 'AICc', the statistic LM or GLM is returning
`family`	the GLM family of distributions (must have the same support as the data for instance exponential cannot have negative numbers)
`method`	the method of choosing parents, 'roulette', 'rank', 'tournament', and 'sus'
`susN`	if sus is selected how many parents are chosen scholastically
`tourn_size`	if tournament is chosen how many candidates are in the tournament
`mutation`	a character either 'fixed' or 'adaptive' indicating the type of mutation
`mutation_rate`	a numeric between 0 and 1 indicating how often a creature is selected for mutation.
`minimize_inbreeding`	a logical indicating whether or not we minimize inbreeding.
`crossover`	a character that can be 'uniform', 'fitness', or ‘k_point’ indicating how to carry out crossover.
`number_of_crossovers`	number of k_point crossovers, needs to be less than gene_length
`elitism`	a logical representing whether elitism is requested
`elite_prop`	the proportion of each generation that is selected for elitism
`ad_max_mutate`	max mutation rate for adaptive mutation, numeric between 0 and 1 and more than ad_min
`ad_min_mutate`	min mutation rate for adaptive mutation, numeric between 0 and 1 and less than ad_max
`ad_inflection`	percentage of diversity in population where adaptive mutation begins to increase rapidly
`ad_curve`	rate that influence how intensely adaptive mutation changes
`custom_function`	user defined custom mast take vector representing a single creatures genes, data, and return a single numeric
`estimator`	character string from 'Min.','1st Qu.','Median','Mean','3rd Qu.', or 'Max.'. This term controls termination conditions for instance, terminating after the Mean AIC is above a certain threshold
`pause_length`	a numeric indicating how many iterations without improvement in the estimator before you terminate

`percent_converge`	percentage of diversity when algorithm terminates for instance .1 will terminate if there are only .90 of the population is represented by the same genes
`score_threshold`	a numeric indicating a threshold to cutoff if the estimator reaches that value
`fittest`	a character either 'high' or 'low' defining whether a custom functions

Details

user_genes allows the user specific genes to the initial generation provided that the matrix consist only of 1s and 0s has no all-zero rows, ncol = gene_length, and has nrow less than or equal the total population pop

custom_function allows the user to specify a custom function instead of lm() or glm(). User-provided function must have its first two arguments be (1) generation_matrix and (2) data. Function needs to take a single row of a generation_matrix and return a single numeric without NAs or infinities.

Otherwise, user should specify a metric from R2, AIC, BIC, or AICC. The underlying fitness function for R2 is lm() and the underlying fitness function for AIC, BIC, and AICc is glm(). AIC, BIC, and AICC has the option to specify a specific family of functions however, the data needs to be defined across the support for the given family for instance exponential must be greater than 0.

Parents are selected via a number of methods: #'Roulette Method selects parents randomly with a probability proportional to their fitness

Rank Method selects parents randomly with a probability proportional to the rank of their fitness

Tournament uses tourn_size to randomly group that many creatures together. The most fit in that group becomes a parent.

Stochastic Universal Sampling works like roulette but also selects a number (susN) more candidates at a fixed width from the first draw to increase diversity

crossover Methods Uniform - each gene is randomly selected from 2 or more parents from a PMF proportional to the number of parents.

Fitness - each gene is randomly selected from 2 or more parents from a PMF proportional to the parent's fitness.

K-Point - parents genes are broken into k+1 segments, then the offspring inherits portions randomly from the parents. Takes the parameter number_of_crossovers which must be less than 1/2 gene_length.

For all methods candidate offspring are accepted / rejected so that they don't have completely 0-vector genes.

mutation Fixed - Each offspring has a mutation_rate chance of being selected for mutation. Once selected one gene is switched form one to zero or zero to one.

Adaptive - the overall population is measured for diversity. As diversity becomes lower, the mutation rate increases. Once selected for mutation a single gene is switched from one to zero or zero to one. The adaptive function is controlled by a simple logistic function with parameters ad_min and ad_max describing the minimum and maximum mutation rates. ad_inflection controls where the logistics point pivots, and ad_curve controls how rapidly the logistics curve increase.

For all mutation methods, candidate offspring are accepted / rejected so that they don't have completely 0-vector genes.

Minimize inbreeding. his option reduces (though does not remove) the chance of similar creatures creating offspring together. Each parent is assigned new partner(s) randomly from a PMF proportional to how different their genes are. Parents are drawn without replacement so that if they were selected to become parents, they will still remain parents.

Elitism preserves the most-fit creatures from each generation. by selecting a number of most fit creatures equal to the ceiling(pop * elite_prop) and guarantees that they make it to the next generation. Additionally, this features makes a copy of each most-fit creature and conducts one gene mutation randomly on each creature. If the copy is more fit than the original, it is returned instead of the original so that the final returned matrix of elite creatures are at least as fit as the incoming elite creatures.

Early Termination Conditions: User can define early termination criteria to include a percentage_convergence such that when diversity falls below a specific threshold the program terminates. User can identify a estimate (from the standard summary() function that if the estimate reaches a specific threshold the program terminates or if the estimate pauses for a certain number of iterations terminates.

estimator - cam be nominated from the summary() function aka Min., 1st Qu., Median, Mean, 3rd Qu., or Max. so for instance estimator=Max. metric = 'AIC' and score_threshold = 500 would terminate when the Max. AIC falls below 500.

diversity is defined as sum(unique(genes))/total_genes

Value

returns a list consisting of [[1]] the fittest gene, [[2]] the fittest gene's score, [[3]] the most fit score by generation, [[4]] the summary data by generation and [[5]] the final generation's genes.

Examples

```
pop <- 50
gene_length <- 10
response_vec <-rnorm(100)
independent_vars <- matrix(rnorm(100*10),ncol=10)
total_number_generations <-15
select(pop = pop,gene_length = gene_length,
response_vec = response_vec,independent_vars = independent_vars,
total_number_generations = total_number_generations)

response_vec <-rnorm(100)
independent_vars <- matrix(rnorm(100*50),ncol=50)
pop <- 50
gene_length <-50
elitism <- TRUE
elite_prop <- .05
mutation_rate <- .1
crossover <- 'k_point'
method <-"rank"
number_generations <-50

select(pop = pop,gene_length = gene_length,
response_vec = response_vec,independent_vars = independent_vars,
total_number_generations = total_number_generations,
elitism = elitism,mutation_rate = mutation_rate,crossover = crossover,
method = method)
```

`select_parent` *Selects parents for the next generation of creatures*

Description

This function's purpose is to take a score vector and to return a vector of the next generation of parents. To do this to do this it the number of parents required per an offspring, and the general population required. The resulting new_parents vector will be of length pop_required * number_of_parents.

Usage

```
select_parent(
  score_vec,
  number_of_parents,
  method = "roulette",
  susN = 1,
  tourn_size = 4,
  pop_required
)
```

Arguments

`score_vec`	a positive, numeric vector with length equal to the total population
`number_of_parents`	an integer indicating the number of parents per offspring
`method`	the method of choosing parents, 'roulette', 'rank', 'tournament', and 'sus'
`susN`	if sus is selected how many parents are chosen scholastically
`tourn_size`	if tournament is chosen how many candidates are in the tournament
`pop_required`	number of offspring needed at the start of the next generation

Details

Roulette Method selects parents randomly with a probability proportional to their fitness

Rank Method selects parents randomly with a probability proportional to the rank of their fitness

Tournament uses tourn_size to randomly group that many creatures together. The most fit in that group becomes a parent.

Stochastic Universal Sampling works like roulette but also selects a number (susN) more candidates at a fixed width from the first draw to increase diversity

Value

a numeric vector indexing which parents are reproducing for the next generation

Examples

```
score_vec <- abs(rnorm(10))
number_of_parents <-2
pop_required <- 10

new_parents <- select_parent(score_vec = score_vec,
 number_of_parents = number_of_parents,pop_required = pop_required)
```

`test_user_function` *Tests User_Provided Custom Function*

Description

This function confirms that a user-provided function takes genes from a generation_matrix and a data matrix and returns a score for each gene It also confirms there are not NAs or infinity values. This function does not confirm that the scores are in descending order aka higher fitness creatures receive a higher fitness scores. It also does not confirm that all possible scores are defined.

Usage

```
test_user_function(func, generation_matrix, data)
```

Arguments

`func`	user provided function (needs generation_matrix followed by data)
`generation_matrix`	generation matrix with all 0s and 1s with ncol = gene_length and nrow = pop
`data`	a matrix or dataframe with ncol = gene_length plus 1

Details

User-provided function must have its first two arguments be (1) generation_matrix and (2) data. Function needs to take a single row of a generation_matrix and return a single numeric without NAs or infinities.

Value

TRUE if it passes and an error message otherwise

Examples

```
generation_matrix <- matrix(rbinom(2*10,1,.5),ncol=10)
data <- matrix(rnorm(11*3),nc=11)
func <- function(generation_matrix,data) {sum(generation_matrix)}

test <- test_user_function(func,generation_matrix,data)
```