Package: GenAlgo (via r-universe)

August 25, 2024

Version 2.2.0

Date 2020-10-13

Title Classes and Methods to Use Genetic Algorithms for Feature Selection

Author Kevin R. Coombes

Maintainer Kevin R. Coombes <krc@silicovore.com>

Depends R (>= 3.0)

Imports methods, stats, MASS, oompaBase (>= 3.0.1), ClassDiscovery

Suggests Biobase, xtable

Description Defines classes and methods that can be used to implement genetic algorithms for feature selection. The idea is that we want to select a fixed number of features to combine into a linear classifier that can predict a binary outcome, and can use a genetic algorithm heuristically to select an optimal set of features.

License Apache License (== 2.0)

LazyLoad yes

biocViews Microarray, Clustering

URL http://oompa.r-forge.r-project.org/

NeedsCompilation no

Repository CRAN

Date/Publication 2020-10-15 17:40:03 UTC

Contents

`gaTourResults` *Results of a Genetic Algorithm*

Description

We ran a genetic algorithm to find the optimal 'fantasy' team for the competition run by the Versus broadcasting network for the 2009 Tour de France. In order to make the vignette run in a timely fashion, we saved the results in this data object.

Usage

```
data(gaTourResults)
```

Format

There are four objects in the data file. The first is `recurse`, which is an object of the `GenAlg-class` representing the final generation. The other three objects are all numeric vector of length 1100: `diversity` contains the average population diversity at each generation, `fitter` contains the maximum fitness, and `meanfit` contains the mean fitness.

Source

Kevin R. Coombes

`GenAlg` *A generic Genetic Algorithm for feature selection*

Description

These functions allow you to initialize (`GenAlg`) and iterate (`newGeneration`) a genetic algorithm to perform feature selection for binary class prediction in the context of gene expression microarrays or other high-throughput technologies.

Usage

```
GenAlg(data, fitfun, mutfun, context, pm=0.001, pc=0.5, gen=1)
newGeneration(ga)
popDiversity(ga)
```

Arguments

data	The initial population of potential solutions, in the form of a data matrix with one individual per row.
fitfun	A function to compute the fitness of an individual solution. Must take two input arguments: a vector of indices into rows of the population matrix, and a context list within which any other items required by the function can be resolved. Must return a real number; higher values indicate better fitness, with the maximum fitness occurring at the optimal solution to the underlying numerical problem.
mutfun	A function to mutate individual alleles in the population. Must take two arguments: the starting allele and a context list as in the fitness function.
context	A list of additional data required to perform mutation or to compute fitness. This list is passed along as the second argument when fitfun and mutfun are called.
pm	A real value between 0 and 1, representing the probability that an individual allele will be mutated.
pc	A real value between 0 and 1, representing the probability that crossover will occur during reproduction.
gen	An integer identifying the current generation.
ga	An object of class GenAlg

Value

Both the GenAlg generator and the newGeneration functions return a GenAlg-class object. The popDiversity function returns a real number representing the average diversity of the population. Here diversity is defined by the number of alleles (selected features) that differ in two individuals.

Author(s)

Kevin R. Coombes <krc@silicovore.com>, P. Roebuck <proebuck@mdanderson.org>

See Also

GenAlg-class, GenAlg-tools, maha.

Examples

```
# generate some fake data
nFeatures <- 1000
nSamples <- 50
fakeData <- matrix(rnorm(nFeatures*nSamples), nrow=nFeatures, ncol=nSamples)
fakeGroups <- sample(c(0,1), nSamples, replace=TRUE)
myContext <- list(dataset=fakeData, gps=fakeGroups)

# initialize population
n.individuals <- 200
n.features <- 9
y <- matrix(0, n.individuals, n.features)
for (i in 1:n.individuals) {
  y[i,] <- sample(1:nrow(fakeData), n.features)
```

```
}

# set up the genetic algorithm
my.ga <- GenAlg(y, selectionFitness, selectionMutate, myContext, 0.001, 0.75)

# advance one generation
my.ga <- newGeneration(my.ga)
```

GenAlg-class *Class "GenAlg"*

Description

Objects of the GenAlg class represent one step (population) in the evolution of a genetic algorithm. This algorithm has been customized to perform feature selection for the class prediction problem.

Usage

```
## S4 method for signature 'GenAlg'
as.data.frame(x, row.names=NULL, optional=FALSE, ...)
## S4 method for signature 'GenAlg'
as.matrix(x, ...)
## S4 method for signature 'GenAlg'
summary(object, ...)
```

Arguments

object	object of class GenAlg
x	object of class GenAlg
row.names	character vector giving the row names for the data frame, or NULL
optional	logical scalar. If TRUE, setting row names and converting column names to syntactic names is optional.
...	extra arguments for generic routines

Objects from the Class

Objects should be created by calls to the GenAlg generator; they will also be created automatically as a result of applying the function newGeneration to an existing GenAlg object.

Slots

data: The initial population of potential solutions, in the form of a data matrix with one individual per row.

`fitfun`: A function to compute the fitness of an individual solution. Must take two input arguments: a vector of indices into the rows of the population matrix, and a `context` list within which any other items required by the function can be resolved. Must return a real number; higher values indicate better fitness, with the maximum fitness occurring at the optimal solution to the underlying numerical problem.

`mutfun`: A function to mutate individual alleles in the population. Must take two arguments: the starting allele and a `context` list as in the fitness function.

`p.mutation`: numeric scalar between `0` and `1`, representing the probability that an individual allele will be mutated.

`p.crossover`: numeric scalar between `0` and `1`, representing the probability that crossover will occur during reproduction.

`generation`: integer scalar identifying the current generation.

`fitness`: numeric vector containing the fitness of all individuals in the population.

`best.fit`: A numeric value; the maximum fitness.

`best.individual`: A matrix (often with one row) containing the individual(s) achieving the maximum fitness.

`context`: A list of additional data required to perform mutation or to compute fitness. This list is passed along as the second argument when `fitfun` and `mutfun` are called.

Methods

as.data.frame `signature(x = "GenAlg")`: Converts the `GenAlg` object into a data frame. The first column contains the fitness ; remaining columns contain three selected features, given as integer indices into the rows of the original data matrix.

as.matrix `signature(x = "GenAlg")`: Converts the GenAlg object into a matrix, following the conventions of `as.data.frame`.

summary `signature(object = "GenAlg")`: Print a summary of the GenAlg object.

Author(s)

Kevin R. Coombes <krc@silicovore.com>, P. Roebuck <proebuck@mdanderson.org>

References

David Goldberg.
"Genetic Algorithms in Search, Optimization and Machine Learning."
Addison-Wesley, 1989.

See Also

`GenAlg`, `GenAlg-tools`, `maha`.

Examples

```
showClass("GenAlg")
```

GenAlg-tools *Utility functions for selection and mutation in genetic algorithms*

Description

These functions implement specific forms of mutation and fitness that can be used in genetic algorithms for feature selection.

Usage

```
simpleMutate(allele, context)
selectionMutate(allele, context)
selectionFitness(arow, context)
```

Arguments

allele	In the simpleMutate function, allele is a binary vector filled with 0's and 1's. In the selectionMutate function, allele is an integer (which is silently ignored; see Details).
arow	A vector of integer indices identifying the rows (features) to be selected from the context$dataset matrix.
context	A list or data frame containing auxiliary information that is needed to resolve references from the mutation or fitness code. In both selectionMutate and selectionFitness, context must contain a dataset component that is either a matrix or a data frame. In selectionFitness, the context must also include a grouping factor (with two levels) called gps.

Details

These functions represent 'callbacks'. They can be used in the function GenAlg, which creates objects. They will then be called repeatedly (for each individual in the population) each time the genetic algorithm is updated to the next generation.

The simpleMutate function assumes that chromosomes are binary vectors, so alleles simply take on the value 0 or 1. A mutation of an allele, therefore, flips its state between those two possibilities.

The selectionMutate and selectionFitness functions, by contrast, are specialized to perform feature selection assuming a fixed number K of features, with a goal of learning how to distinguish between two different groups of samples. We assume that the underlying data consists of a data frame (or matrix), with the rows representing features (such as genes) and the columns representing samples. In addition, there must be a grouping vector (or factor) that assigns all of the sample columns to one of two possible groups. These data are collected into a list, context, containing a dataset matrix and a gps factor. An individual member of the population of potential solutions is encoded as a length K vector of indices into the rows of the dataset. An individual allele, therefore, is a single index identifying a row of the dataset. When mutating it, we assume that it can be changed into any other possible allele; i.e., any other row number. To compute the fitness, we use the Mahalanobis distance between the centers of the two groups defined by the gps factor.

Value

Both `selectionMutate` and `simpleMutate` return an integer value; in the simpler case, the value is guaranteed to be a 0 or 1. The `selectionFitness` function returns a real number.

Author(s)

Kevin R. Coombes <krc@silicovore.com>, P. Roebuck <proebuck@mdanderson.org>

See Also

GenAlg, GenAlg-class, maha.

Examples

```
# generate some fake data
nFeatures <- 1000
nSamples <- 50
fakeData <- matrix(rnorm(nFeatures*nSamples), nrow=nFeatures, ncol=nSamples)
fakeGroups <- sample(c(0,1), nSamples, replace=TRUE)
myContext <- list(dataset=fakeData, gps=fakeGroups)

# initialize population
n.individuals <- 200
n.features <- 9
y <- matrix(0, n.individuals, n.features)
for (i in 1:n.individuals) {
  y[i,] <- sample(1:nrow(fakeData), n.features)
}

# set up the genetic algorithm
my.ga <- GenAlg(y, selectionFitness, selectionMutate, myContext, 0.001, 0.75)

# advance one generation
my.ga <- newGeneration(my.ga)
```

---

maha *Compute the (squared) Mahalanobis distance between two groups of vectors*

Description

The Mahalanobis distance between two groups of vectors

Usage

```
maha(data, groups, method = "mve")
```

Arguments

data	A matrix with columns representing features (or variables) and rows representing independent samples
groups	A factor or logical vector with length equal to the number of rows (samples) in the data matrix
method	A character string determining the method that should be used to estimate the covariance matrix. The default value of "mve" uses the cov.mve function from the MASS package. The other valid option is "var", which uses the var function from the standard stats package.

Details

The Mahalanobis distance between two groups of vectors is the distance between their centers, computed in the equivalent of a principal component space that accounts for different variances.

Value

Returns a numeric vector of length 1.

Author(s)

Kevin R. Coombes <krc@silicovore.com>, P. Roebuck <proebuck@mdanderson.org>

References

Mardia, K. V. and Kent, J. T. and Bibby, J. M.
Multivariate Analysis.
Academic Press, Reading, MA 1979, pp. 213–254.

See Also

cov.mve, var

Examples

```
nFeatures <- 40
nSamples <- 2*10
dataset <- matrix(rnorm(nSamples*nFeatures), ncol=nSamples)
groups <- factor(rep(c("A", "B"), each=10))
maha(dataset, groups)
```

`tourData09` *Tour de France 2009*

Description

Each row represents the performance of a rider in the 2009 Tour de France; the name and team of the rider are used as the row names. The four columns are the `Cost` (to include on a team in the Versus fantasy challenge), `Scores` (based on daily finishing position), `JerseyBonus` (for any days spent in one of the three main leader jerseys), and `Total` (the sum of Scores and JerseyBonus).

Usage

```
data(tourData09)
```

Format

A data frame with 102 rows and 4 columns.

Source

The data were collected in 2009 from the web site `http://www.versus.com/tdfgames`, which appears to no longer exist.

Index