Package ‘morpheus’

February 9, 2023

Title Estimate Parameters of Mixtures of Logistic Regressions

Description Mixture of logistic regressions parameters (H)estimation with
(U)spectral methods. The main methods take d-dimensional inputs and a vector
of binary outputs, and return parameters according to the GLMs mixture model
(General Linear Model). For more details see chapter 3 in the PhD thesis of
Mor-Absa Loum: <https://www.theses.fr/s156435>, available here
<https://theses.hal.science/tel-01877796/document>.

Version 1.0-4

Author Benjamin Auder <Benjamin.Auder@u-psud.fr> [aut,cre],
Mor-Absa Loum <Mor-Absa.Loum@u-psud.fr> [aut]

Maintainer Benjamin Auder <Benjamin.Auder@u-psud.fr>

Depends R (>= 3.5.0),

Imports MASS, jointDiag, methods, pracma

Suggests devtools, flexmix, parallel, testthat (>= 3.0.0), roxygen2

License MIT + file LICENSE

RoxygenNote 7.2.0

URL https://github.com/yagu0/morpheus

Collate 'utils.R' 'A_NAMESPACE.R' 'computeMu.R' 'multiRun.R'
'optimParams.R' 'plot.R' 'sampleIO.R'

Config/testthat/edition 3

NeedsCompilation yes

Repository CRAN

Date/Publication 2023-02-09 11:20:12 UTC

R topics documented:

`morpheus-package` *Estimate Parameters of Mixtures of Logistic Regressions*

Description

Mixture of logistic regressions parameters (H)estimation with (U)spectral methods. The main methods take d-dimensional inputs and a vector of binary outputs, and return parameters according to the GLMs mixture model (General Linear Model). For more details see chapter 3 in the PhD thesis of Mor-Absa Loum: <https://www.theses.fr/s156435>, available here <https://theses.hal.science/tel-01877796/document>.

Details

The package devtools should be useful in development stage, since we rely on testthat for unit tests, and roxygen2 for documentation. knitr is used to generate the package vignette. jointDiag allows to solve a joint diagonalization problem, providing a more robust solution compared to a single diagonalization. Concerning the other suggested packages:

- tensor is used for comparing to some reference functions initially coded in R; it should not be required in further package versions;
- parallel (generally) permits to run the bootstrap method faster.

The two main functions are located in R/computeMu.R and R/optimParams.R:

- computeMu(): estimation of parameters directions;
- optimParams(): builds an object `o` to estimate all other parameters when calling `o$run()`, starting from the directions obtained by previous function

See also `multiRun()`, which is a flexible method to run Monte-Carlo or bootstrap estimations using different models in various contexts.

Author(s)

Benjamin Auder <Benjamin.Auder@u-psud.fr> [aut,cre], Mor-Absa Loum <Mor-Absa.Loum@u-psud.fr> [aut]

Maintainer: Benjamin Auder <Benjamin.Auder@u-psud.fr>

alignMatrices *alignMatrices*

Description

Align a set of parameters matrices, with potential permutations.

Usage

```
alignMatrices(Ms, ref, ls_mode = c("exact", "approx1", "approx2"))
```

Arguments

`Ms`	A list of matrices, all of same size DxK
`ref`	A reference matrix to align other matrices with
`ls_mode`	How to compute the labels assignment: "exact" for exact algorithm (default, but might be time-consuming, complexity is O(K^3)), or "approx1", or "approx2" to apply a greedy matching algorithm (heuristic) which for each column in reference (resp. in current row) compare to all unassigned columns in current row (resp. in reference)

Value

The aligned list (of matrices), of same size as Ms

Examples

```
m1 <- matrix(c(1,1,0,0),ncol=2)
m2 <- matrix(c(0,0,1,1),ncol=2)
ref <- m1
Ms <- list(m1, m2, m1, m2)
a <- alignMatrices(Ms, ref, "exact")
# a[[i]] is expected to contain m1 for all i
```

computeMoments *computeMoments*

Description

Compute cross-moments of order 1,2,3 from X,Y

Usage

```
computeMoments(X, Y)
```

Arguments

`X`	Matrix of input data (size nxd)
`Y`	Vector of binary outputs (size n)

Value

A list L where L[[i]] is the i-th cross-moment

Examples

```
X <- matrix(rnorm(100), ncol=2)
Y <- rbinom(100, 1, .5)
M <- computeMoments(X, Y)
```

`computeMu` *Compute mu*

Description

Estimate the normalized columns mu of the beta matrix parameter in a mixture of logistic regressions models, with a spectral method described in the package vignette.

Usage

```
computeMu(X, Y, optargs = list())
```

Arguments

`X`	Matrix of input data (size nxd)
`Y`	Vector of binary outputs (size n)
`optargs`	List of optional argument: • 'jd_method', joint diagonalization method from the package jointDiag: 'uwedge' (default) or 'jedi'. • 'jd_nvects', number of random vectors for joint-diagonalization (or 0 for p=d, canonical basis by default) • 'M', moments of order 1,2,3: will be computed if not provided. • 'K', number of populations (estimated with rank of M2 if not given)

Value

The estimated normalized parameters as columns of a matrix mu of size dxK

See Also

`multiRun` to estimate statistics based on mu, and `generateSampleIO` for I/O random generation.

Examples

```
io <- generateSampleIO(10000, 1/2, matrix(c(1,0,0,1),ncol=2), c(0,0), "probit")
mu <- computeMu(io$X, io$Y, list(K=2)) #or just X and Y for estimated K
```

`generateSampleIO` *Generate sample inputs-outputs*

Description

Generate input matrix X of size nxd and binary output of size n, where Y is subdivided into K groups of proportions p. Inside one group, the probability law P(Y=1) is described by the corresponding column parameter in the matrix beta + intercept b.

Usage

```
generateSampleIO(n, p, beta, b, link)
```

Arguments

`n`	Number of individuals
`p`	Vector of K(-1) populations relative proportions (sum (<)= 1)
`beta`	Vectors of model parameters for each population, of size dxK
`b`	Vector of intercept values (use rep(0,K) for no intercept)
`link`	Link type; "logit" or "probit"

Value

A list with

- X: the input matrix (size nxd)
- Y: the output vector (size n)
- index: the population index (in 1:K) for each row in X

Examples

```
# K = 3 so we give first two components of p: 0.3 and 0.3 (p[3] = 0.4)
io <- generateSampleIO(1000, c(.3,.3),
  matrix(c(1,3,-1,1,2,1),ncol=3), c(.5,-1,0), "logit")
io$index[1] #number of the group of X[1,] and Y[1] (in 1...K)
```

multiRun *multiRun*

Description

Estimate N times some parameters, outputs of some list of functions. This method is thus very generic, allowing typically bootstrap or Monte-Carlo estimations of matrices mu or beta. Passing a list of functions opens the possibility to compare them on a fair basis (exact same inputs). It's even possible to compare methods on some deterministic design of experiments.

Usage

```
multiRun(
  fargs,
  estimParams,
  prepareArgs = function(x, i) x,
  N = 10,
  ncores = 3,
  agg = lapply,
  verbose = FALSE
)
```

Arguments

`fargs`	List of arguments for the estimation functions
`estimParams`	List of nf function(s) to apply on fargs
`prepareArgs`	Prepare arguments for the functions inside estimParams
`N`	Number of runs
`ncores`	Number of cores for parallel runs (<=1: sequential)
`agg`	Aggregation method (default: lapply)
`verbose`	TRUE to indicate runs + methods numbers

Value

A list of nf aggregates of N results (matrices).

Examples

```
## Not run:
beta <- matrix(c(1,-2,3,1),ncol=2)

# Bootstrap + computeMu, morpheus VS flexmix
io <- generateSampleIO(n=1000, p=1/2, beta=beta, b=c(0,0), "logit")
mu <- normalize(beta)
res <- multiRun(list(X=io$X,Y=io$Y,K=2), list(
  # morpheus
```

```
    function(fargs) {
      library(morpheus)
      ind <- fargs$ind
      computeMu(fargs$X[ind,], fargs$Y[ind], list(K=fargs$K))
    },
    # flexmix
    function(fargs) {
      library(flexmix)
      ind <- fargs$ind
      K <- fargs$K
      dat <- as.data.frame( cbind(fargs$Y[ind],fargs$X[ind,]) )
      out <- refit( flexmix( cbind(V1, 1 - V1) ~ 0+., data=dat, k=K,
        model=FLXMRglm(family="binomial") ) )
      normalize( matrix(out@coef[1:(ncol(fargs$X)*K)], ncol=K) )
    } ),
    prepareArgs = function(fargs,index) {
      if (index == 1)
        fargs$ind <- 1:nrow(fargs$X)
      else
        fargs$ind <- sample(1:nrow(fargs$X),replace=TRUE)
      fargs
    }, N=10, ncores=3)
  for (i in 1:2)
    res[[i]] <- alignMatrices(res[[i]], ref=mu, ls_mode="exact")

  # Monte-Carlo + optimParams from X,Y, morpheus VS flexmix
  res <- multiRun(list(n=1000,p=1/2,beta=beta,b=c(0,0),link="logit"), list(
    # morpheus
    function(fargs) {
      library(morpheus)
      K <- fargs$K
      mu <- computeMu(fargs$X, fargs$Y, list(K=fargs$K))
      o <- optimParams(fargs$X, fargs$Y, fargs$K, fargs$link, fargs$M)
      o$run(list(beta=mu))$beta
    },
    # flexmix
    function(fargs) {
      library(flexmix)
      K <- fargs$K
      dat <- as.data.frame( cbind(fargs$Y,fargs$X) )
      out <- refit( flexmix( cbind(V1, 1 - V1) ~ ., data=dat, k=K,
        model=FLXMRglm(family="binomial") ) )
      sapply( seq_len(K), function(i)
        as.double( out@components[[1]][[i]][2:(1+ncol(fargs$X)),1] ) )
    } ),
    prepareArgs = function(fargs,index) {
      library(morpheus)
      io <- generateSampleIO(fargs$n, fargs$p, fargs$beta, fargs$b, fargs$link)
      fargs$X <- io$X
      fargs$Y <- io$Y
      fargs$K <- ncol(fargs$beta)
      fargs$link <- fargs$link
      fargs$M <- computeMoments(io$X,io$Y)
      fargs
    }, N=10, ncores=3)
  for (i in 1:2)
    res[[i]] <- alignMatrices(res[[i]], ref=beta, ls_mode="exact")
  ## End(Not run)
```

---

normalize *normalize*

Description

Normalize a vector or a matrix (by columns), using euclidian norm

Usage

```
normalize(x)
```

Arguments

x	Vector or matrix to be normalized

Value

The normalized matrix (1 column if x is a vector)

Examples

```
x <- matrix(c(1,2,-1,3), ncol=2)
normalize(x) #column 1 is 1/sqrt(5) (1 2),
             #and column 2 is 1/sqrt(10) (-1, 3)
```

---

optimParams *optimParams*

Description

Wrapper function for OptimParams class

Usage

```
optimParams(X, Y, K, link = c("logit", "probit"), M = NULL, nc = 0)
```

Arguments

`X`	Data matrix of covariables
`Y`	Output as a binary vector
`K`	Number of populations.
`link`	The link type, 'logit' or 'probit'.
`M`	the empirical cross-moments between X and Y (optional)
`nc`	Number of cores (default: 0 to use all)

Value

An object 'op' of class OptimParams, initialized so that `op$run(theta0)` outputs the list of optimized parameters

- p: proportions, size K
- beta: regression matrix, size dxK
- b: intercepts, size K

theta0 is a list containing the initial parameters. Only beta is required (p would be set to (1/K,...,1/K) and b to (0,...0)).

See Also

`multiRun` to estimate statistics based on beta, and `generateSampleIO` for I/O random generation.

Examples

```
# Optimize parameters from estimated mu
io <- generateSampleIO(100,
  1/2, matrix(c(1,-2,3,1),ncol=2), c(0,0), "logit")
mu <- computeMu(io$X, io$Y, list(K=2))
o <- optimParams(io$X, io$Y, 2, "logit")
## Not run:
theta0 <- list(p=1/2, beta=mu, b=c(0,0))
par0 <- o$run(theta0)
# Compare with another starting point
theta1 <- list(p=1/2, beta=2*mu, b=c(0,0))
par1 <- o$run(theta1)
# Look at the function values at par0 and par1:
o$f( o$linArgs(par0) )
o$f( o$linArgs(par1) )
## End(Not run)
```

plotBox *plotBox*

Description

Draw compared boxplots of a single parameter (scalar)

Usage

```
plotBox(mr, x, y, ...)
```

Arguments

mr	Output of multiRun(), list of lists of functions results
x	Row index of the element inside the aggregated parameter
y	Column index of the element inside the aggregated parameter
...	Additional graphical parameters (xlab, ylab, ...)

Examples

```
## Not run:
beta <- matrix(c(1,-2,3,1),ncol=2)
mr <- multiRun(...) #see bootstrap example in ?multiRun
                    #mr[[i]] is a list of estimated parameters matrices
mu <- normalize(beta)
for (i in 1:2)
  mr[[i]] <- alignMatrices(res[[i]], ref=mu, ls_mode="exact")
plotBox(mr, 2, 1) #second row, first column
## End(Not run)
```

plotCoefs *plotCoefs*

Description

Draw a graph of (averaged) coefficients estimations with their standard, deviations ordered by mean values. Note that the drawing does not correspond to a function; it is just a convenient way to visualize the estimated parameters.

Usage

```
plotCoefs(mr, params, ...)
```

Arguments

`mr`	List of parameters matrices
`params`	True value of the parameters matrix
`...`	Additional graphical parameters

Examples

```
## Not run:
beta <- matrix(c(1,-2,3,1),ncol=2)
mr <- multiRun(...) #see bootstrap example in ?multiRun
                    #mr[[i]] is a list of estimated parameters matrices
mu <- normalize(beta)
for (i in 1:2)
  mr[[i]] <- alignMatrices(res[[i]], ref=mu, ls_mode="exact")
params <- rbind( c(.5,.5), beta, c(0,0) ) #p, beta, b stacked in a matrix
plotCoefs(mr[[1]], params)
## End(Not run)
```

`plotHist` *plotHist*

Description

Plot compared histograms of a single parameter (scalar)

Usage

```
plotHist(mr, x, y, ...)
```

Arguments

`mr`	Output of multiRun(), list of lists of functions results
`x`	Row index of the element inside the aggregated parameter
`y`	Column index of the element inside the aggregated parameter
`...`	Additional graphical parameters (xlab, ylab, ...)

Examples

```
## Not run:
beta <- matrix(c(1,-2,3,1),ncol=2)
mr <- multiRun(...) #see bootstrap example in ?multiRun
                    #mr[[i]] is a list of estimated parameters matrices
mu <- normalize(beta)
for (i in 1:2)
  mr[[i]] <- alignMatrices(res[[i]], ref=mu, ls_mode="exact")
plotHist(mr, 2, 1) #second row, first column
## End(Not run)
```

Index