Package 'morpheus'

June 14, 2018

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: <http://www.theses.fr/s156435>, available here <https://www.math.u-psud.fr/~loum/IMG/pdf/these.compressed-2.pdf>.

Version 0.2-0

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 (>= 3.0.0),

Imports MASS, jointDiag, methods, pracma

Suggests devtools, flexmix, parallel, testthat, roxygen2, tensor, nloptr

License MIT + file LICENSE

RoxygenNote 5.0.1

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

NeedsCompilation yes

Repository CRAN

Date/Publication 2018-06-14 18:48:14 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: <http://www.theses.fr/s156435>, available here <https://www.math.u-psud.fr/~loum/IMG/pdf/these.compressed-2.pdf>.

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. 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;
- jointDiag allows to solve a joint diagonalization problem, providing a more robust solution compared to a single diagonalization;
- parallel (generally) permits to run the bootstrap method faster.

The three main functions are located in R/main.R:

- getParamsDirs_ref: reference method to estimate parameters directions;
- getParamsDirs: method of choice to estimate parameters directions, using a spectral decomposition of inputs/outputs;
- getBootstrapParams: run getParamsDirs on B bootstrap replicates.

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)
```

Arguments

`Ms`	A list of matrices, all of same size DxK
`ref`	Either a reference matrix or "mean" to align on empirical mean
`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

`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

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 ranks 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

`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 - shared signature
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

```
beta <- matrix(c(1,-2,3,1),ncol=2)

# Bootstrap + computeMu, morpheus VS flexmix ; assumes fargs first 3 elts X,Y,K
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,optargs=list(K=2,jd_nvects=0)), list(
  # morpheus
  function(fargs) {
    library(morpheus)
    ind <- fargs$ind
    computeMu(fargs$X[ind,],fargs$Y[ind],fargs$optargs)
  },
  # flexmix
  function(fargs) {
    library(flexmix)
    ind <- fargs$ind
    K <- fargs$optargs$K
    dat = as.data.frame( cbind(fargs$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 ; first args n,p,beta,b
```

```
res <- multiRun(list(n=1000,p=1/2,beta=beta,b=c(0,0),optargs=list(link="logit")),list(
  # morpheus
  function(fargs) {
    library(morpheus)
    K <- fargs$optargs$K
    mu <- computeMu(fargs$X, fargs$Y, fargs$optargs)
    V <- list( p=rep(1/K,K-1), beta=mu, b=c(0,0) )
    optimParams(V,fargs$optargs)$beta
  },
  # flexmix
  function(fargs) {
    library(flexmix)
    K <- fargs$optargs$K
    dat <- as.data.frame( cbind(fargs$Y,fargs$X) )
    out <- refit( flexmix( cbind(V1, 1 - V1) ~ 0+., data=dat, k=K,
      model=FLXMRglm(family="binomial") ) )
    sapply( seq_len(K), function(i) as.double( out@components[[1]][[i]][,1] ) )
  } ),
  prepareArgs = function(fargs,index) {
    library(morpheus)
    io = generateSampleIO(fargs$n, fargs$p, fargs$beta, fargs$b, fargs$optargs$link)
    fargs$X = io$X
    fargs$Y = io$Y
    fargs$optargs$K = ncol(fargs$beta)
    fargs$optargs$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")
```

`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)

optimParams *Optimize parameters*

Description

Optimize the parameters of a mixture of logistic regressions model, possibly using `mu <- computeMu(...)` as a partial starting point.

Usage

```
optimParams(K, link = c("logit", "probit"), optargs = list())
```

Arguments

K	Number of populations.
link	The link type, 'logit' or 'probit'.
optargs	a list with optional arguments: • 'M' : list of moments of order 1,2,3: will be computed if not provided. • 'X,Y' : input/output, mandatory if moments not given • 'exact': use exact formulas when available?

Value

An object 'op' of class OptimParams, initialized so that `op$run(x0)` outputs the list of optimized parameters

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

x0 is a vector containing respectively the K-1 first elements of p, then beta by columns, and finally b: `x0 = c(p[1:(K-1)],as.double(beta),b)`.

See Also

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

Examples

```
# Optimize parameters from estimated mu
io = generateSampleIO(10000, 1/2, matrix(c(1,-2,3,1),ncol=2), c(0,0), "logit")
mu = computeMu(io$X, io$Y, list(K=2))
M <- computeMoments(io$X, io$Y)
o <- optimParams(2, "logit", list(M=M))
x0 <- c(1/2, as.double(mu), c(0,0))
par0 <- o$run(x0)
# Compare with another starting point
```

```
x1 <- c(1/2, 2*as.double(mu), c(0,0))
par1 <- o$run(x1)
o$f( o$linArgs(par0) )
o$f( o$linArgs(par1) )
```

---

plotBox *plotBox*

Description

Draw boxplot

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

Examples

```
#See example in ?plotHist
```

---

plotCoefs *plotCoefs*

Description

Draw coefs estimations + standard deviations

Usage

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

Arguments

mr	Output of multiRun(), list of lists of functions results
params	True value of parameters matrix

Examples

```
#See example in ?plotHist
```

plotHist *plotHist*

Description

Plot histogram

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

Examples

```
beta <- matrix(c(1,-2,3,1),ncol=2)
mr <- multiRun(...) #see bootstrap example in ?multiRun : return lists of mu_hat
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
```

plotQn *plotQn*

Description

Draw 3D map of objective function values

Usage

```
plotQn(N, n, p, beta, b, link)
```

Arguments

N	Number of starting points
n	Number of points in sample
p	Vector of proportions
beta	Regression matrix (target)
b	Vector of biases
link	Link function (logit or probit)

Index