Package ‘npfixedcompR’

December 30, 2020

Title Non-parametric estimation of mixing distribution with possibily fixed components

Version 0.0.1.0008

Description Non-parametric estimation of mixing distribution with possibly fixed components.

Imports lsei (>= 1.2-0), R6 (>= 2.4.1), goftest (>= 1.2-2), nloptr (>= 1.2.2.1)

License GPL-3

Encoding UTF-8

LazyData true

Roxygen list(markdown = TRUE)

RoxygenNote 7.1.1

R topics documented:

`bin` *Bin the continuous data.*

Description

Bin the continuous data.

Usage

```
bin(data, order = -5)
```

Arguments

`data`	the observation to be binned.
`order`	see the details

Details

This function bins the continuous data using the equal-width bin in order to speed up some functions in this package.

h is taken as 10^(-order)

the observations are rounded down to the bins ..., -h, 0, h, ...

To further speed up the process, omit the bin that has 0 count.

Value

a list with v be the representative value of each bin and w be the count in the corresponding bin.

`computemixdist` *Computing non-parametric mixing distribution*

Description

Computing non-parametric mixing distribution

Usage

```
computemixdist(x, ...)
```

Arguments

`x`	a object from implemented family generated by `makeobject`.
`...`	parameters above passed to the specific method

Details

The full list of implemented family is in `makeobject`.

The avaliable parameters are listed as follows:

- mix: The initial proper mixing distribution
- tol: tolerance to stop the code
- maxiter: maximum iterations allowed.
- verbose: logical; whether to print the intermediate results.

This function essentially calls the class method in the object.

Examples

```
data = rnorm(500, c(0, 2))
pi0 = 0.5
x = makeobject(data, pi0 = pi0, method = "npnormll")
computemixdist(x)
x = makeobject(data, pi0 = pi0, method = "npnormllw")
computemixdist(x)
x = makeobject(data, pi0 = pi0, method = "npnormcvm")
computemixdist(x)
x = makeobject(data, pi0 = pi0, method = "npnormcvmw")
computemixdist(x)
x = makeobject(data, pi0 = pi0, method = "npnormad")
computemixdist(x)
x = makeobject(data, pi0 = pi0, method = "nptll")
computemixdist(x)
```

`covestEB` *Estimating Covariance Matrix using Empirical Bayes*

Description

Estimating covariance matrix using Empirical Bayes

Usage

```
covestEB(X, estpi0 = FALSE, order = -3, verbose = FALSE, force.nonbin = FALSE)

covestEB.cor(X, verbose = FALSE)
```

Arguments

`X`	a matrix of size n * p, where n is the number of observations and p is the number of variables
`estpi0`	logical; if TRUE, the NPMLE is estimated based on the estimation of pi0, which in this case can be used to detect sparsity or assume sparsity.
`order`	the level of binning to use when the number of observations passed to the computation is greater than 5000.
`verbose`	logical; If TRUE, the intermediate results will be shown.
`force.nonbin`	logical; If TRUE, no binning is performce by force.

Details

The function covestEB performs covariance matrix estimation using Fisher transformation, while the function covestEB.cor performs covariance estimation directly on sample correlation coefficients using one-parameter normal approximation.

Covariance matrix estimation using Fisher transformation supports estimation sparsity as well as large-scale computation, while estimation on the original scale supports neither and it is for comparison only. It is recommended to perform estimation on Fisher-transformed sample correlation coefficients.

Value

a list. a covariance matrix estimate of size p * p is given in mat, whether correction is done is given in correction, and the method for computing the density of sample correlation coefficients is given in method.

Examples

```
n = 100; p = 50
X = matrix(rnorm(n * p), nrow = n, ncol = p)
r = covestEB(X)
r2 = covestEB.cor(X)
```

---

ddiscnorm *(non-parametric) discrete normal distribution*

Description

The density and the distribution function of (non-parametric) discrete normal distribution

Usage

```
ddiscnorm(x, mean = 0, sd = 1, h = 1, log = FALSE)

dnpdiscnorm(
  x,
  mu0 = 0,
  pi0 = 0,
  sd = 1,
  h = 1,
  lower.tail = TRUE,
  log.p = FALSE
)

pdiscnorm(x, mean = 0, sd = 1, h = 1, lower.tail = TRUE, log.p = FALSE)

pnpdiscnorm(
  x,
  mu0 = 0,
  pi0 = 0,
  sd = 1,
  h = 1,
```

```
  lower.tail = TRUE,
  log.p = FALSE
)
```

Arguments

x	vector of observations, vector of quantiles
sd	standard deviation.
h	the discretisation parameter.
log, log.p	logical; if TRUE, the result will be given in log scale.
mu0, mean	the vector of support points
pi0	the vector of weights correponding to the support points mu0
lower.tail	logical; if TRUE, the lower probability is computed

Details

ddiscnorm gives the density, pdiscnorm gives the distribution function of the discrete normal distribution. dnpdiscnorm gives the density, pnpdiscnorm gives the distribution function of the non-parametric discrete normal distribution.

The function pnpdiscnorm uses pnpnorm1 to compute the distribution function.

---

dnpnorm *non-parametric normal distribution*

Description

The density and the distribution function of non-parametric normal distribution

Usage

```
dnpnorm(x, mu0 = 0, pi0 = 1, sd = 1, log = FALSE)

pnpnorm(x, mu0 = 0, pi0 = 1, sd = 1, lower.tail = TRUE, log.p = FALSE)

dnpnorm1(x, mu0 = 0, pi0 = 1, sd = 1, lower.tail = TRUE, log.p = FALSE)

pnpnorm1(x, mu0 = 0, pi0 = 1, sd = 1, lower.tail = TRUE, log.p = FALSE)
```

Arguments

x	vector of observations, vector of quantiles
mu0	the vector of support points
pi0	the vector of weights correponding to the support points
sd	standard deviation.
log, log.p	logical; if TRUE, the result will be given in log scale.
lower.tail	logical; if TRUE, the lower probability is computed

Details

dnpnorm gives the density, pnpnorm gives the distribution function, pnpnorm1 focus on the more accurate but slower distribution function of a non-parametric normal distribution dnpt *non-parametric t distribution*

Description

The density and the distribution function of non-parametric t distribution

Usage

```
dnpt(x, mu0 = 0, pi0 = 0, df, log = FALSE)

pnpt(x, mu0 = 0, pi0 = 0, df, lower.tail = TRUE, log.p = FALSE)
```

Arguments

x	vector of observations, vector of quantiles
mu0	the vector of support points
pi0	the vector of weights correponding to the support points
df	degree of freedom.
log, log.p	logical; if TRUE, the result will be given in log scale.
lower.tail	logical; if TRUE, the lower probability is computed

Details

`dnpnorm` gives the density, `pnpnorm` gives the distribution function,

estpi0 *Computing non-parametric mixing distribution with estimated proportion at 0*

Description

computing non-parametric mixing distribution with estimated proportion at 0

Usage

```
estpi0(x, ...)
```

Arguments

x	a object from implemented family
...	parameters above passed to the specific method.

Details

This is a function for computing non-parametric mixing distribution with estimated proportion at 0. Different families will have different threshold values.

The parameters are listed as follows:

- val: Threshold value
- mix: The initial proper mixing distribution.
- tol: tolerance to stop the code.
- maxiter: maximum iterations allowed.
- verbose: logical; Whether to print the intermediate results.

It is not shown in the parameter section since various method have different default thresold values and this function essentially calls the class method in the object.

The full list of implemented families is in `makeobject`.

Examples

```
data = rnorm(500, c(0, 2))
pi0 = 0.5
x = makeobject(data, method = "npnormll")
estpi0(x)
x = makeobject(data, method = "npnormllw")
estpi0(x)
x = makeobject(data, method = "npnormcvm")
estpi0(x)
x = makeobject(data, method = "npnormcvm")
estpi0(x)
x = makeobject(data, method = "npnormad")
estpi0(x)
x = makeobject(data, method = "nptll")
estpi0(x)
```

`extractlower` *Extract or returning the lower triangular part of the matrix*

Description

The function `extractlower` is to extract the strict lower triangular part of a squared matrix and the function `returnlower` is to return the vector value into a symmetric matrix with diagonal 1.

Usage

```
extractlower(A)

returnlower(v)
```

Arguments

A	a matrix to be extracted the lower triangular part
v	a vector to be returned to a symmetric matrix with diagonal 1.

Examples

```
a = matrix(1:100, 10, 10)
b = extractlower(a)
d = returnlower(b)
```

FDRcontrol *FDR controlling procedures*

Description

Implementation of several FDR-controlling procedure in R.

Usage

```
adaptive.stepdown(pval, alpha = 0.05)

BH(pval, alpha = 0.05)
```

Arguments

pval	a vector of p-values (no necessarily sorted)
alpha	given FDR level

Details

The function `adaptive.stepdown` is a simple implementation of the adaptive step-down procedure described in Gavrilov et al. (2009)

The function `BH` is a direct implementation of the procedure described in Benjamini and Hochberg (1995).

Value

a list with num.rejection, the number of rejections computed by this function, and classifer, a vector of TRUE and FALSE; if TRUE, the corresponding input is regarded as null, and as non-null if otherwise.

Examples

```
adaptive.stepdown(pnorm(-abs(rnorm(1000, c(0, 2)))) * 2)
BH(pnorm(-abs(rnorm(1000, c(0, 2)))) * 2)
```

makeobject *Making object for computation*

Description

These functions are used to make the object for computing the non-paramtric mixing distribution or estimating the proportion of zero using non-parametric methods.

Usage

```
makeobject(v, method = "npnormll", ...)

makeobject.npnormad(v, mu0, pi0, beta, order)

makeobject.npnormadw(v, mu0, pi0, beta, order = -3)

makeobject.npnormcll(v, mu0, pi0, beta, order)

makeobject.npnormcvm(v, mu0, pi0, beta, order)

makeobject.npnormcvmw(v, mu0, pi0, beta, order = -3)

makeobject.npnormll(v, mu0, pi0, beta, order)

makeobject.npnormllw(v, mu0, pi0, beta, order = -3)

makeobject.nptll(v, mu0, pi0, beta, order)
```

Arguments

v	the object either numeric or the implmented family
method	An implemented family; see details
...	other parameter passed to the constructor.
mu0	A vector of support points
pi0	A vector of weights corresponding to the support points
beta	structual parameter.
order	the parameter for the binned version.

Details

This is a generic function for making the object for computing the non-parametric mixing distribution or estimating the proportion of zero.

current implemented families are:

- npnormll : normal density using maximum likelihood (Chapter 3). The default beta is 1.
- npnormllw : Binned version of normal density using maximum likelihood (Chapter 6). The default beta is 1 and the default order is -3.
- npnormcvm : normal density using the Cramer-von Mises distance (Chapter 5). The default beta is 1.

- npnormcvmw : Binned version of normal density using the Cramer-von Mises distance (Chapter 6). The default beta is 1 and the default order is -3.
- npnormad : normal density using the Anderson-Darling distance (Chapter 5). The default beta is 1.
- npnormadw : Binned version of normal density using the Anderson-Darling distance (Chapter 6) The default beta is 1 and the default order is -3.
- nptll : t-density using maximum likelihood (Chapter 3). The default beta is infinity (normal distribution).
- npnormcll : the one-parameter normal distribution used for approximating the sample correlation coefficients using maximum likelihood. This does not have a corresponding estimation of zero due to incompleted theory (Chapter 8). There is no default beta. The structure beta is the number of observations.

The default method used is npnormll.

The detailed description of the npfixedcompRobject class is in `npfixedcompRobject`.

`npfixedcompR` *npfixedcompR*

Description

npfixedcompR

Author(s)

Xiangjie Xue

`npfixedcompRobject` *The npfixedcompR object*

Description

This documentation gives a detailed description of the npfixedcompR object. This implementation uses the R6 object.

Details

The structure of the object currently has three layers: The foundation layer is the npfixedcompR class, which has components common to all the specific classes; The second layer is the distributional layer, which has components common to all the classes with the same distribution (the npnorm class); The third layer is the specific layer (the npnormll class), which contains components specific to this particular loss. Since the nptll and the npnormcll class only has one loss implemented, they are implemented in the second layer.

For the following, if a component is listed as public, then it can be accessed as needed. If a component is listed as private, then it can not be accessed. There might be functions listed as public can be used to modified the private values.

The component in the npfixedcompR class are as follows.

- `mu0fixed` (public) : The vector of the locations of fixed support points. This can be set by `initialize` or `modified` implemented in the last year.
- `pi0fixed` (public) : The vector of the weights of fixed support points. This can be set by `initialize` or `modified` implemented in the last year.
- `data` (public) : The observations. The observations can only be set by `makeobject` via `initialzed`. Once the object is defined, the observations can not be modified.
- `len` (public) : The length of the observations. This is different to the effective sample size when dealing with the binned version.
- `result` (public) : The estimated mixing distribution. This is used to store the result computed by `computemixdist` or `estpi0`.
- `methodflag` (public) : This function is used to print or set the algorithm used for finding the new support points. The default argument is `NULL`, which prints the private component `mflag` implemented in the final layer. Other inputs can be `d0`, `d1` and `d2`, which change the algorithm used. The algorithm used should depend on the final layer (hence `mflag` is implemented in the final layer).
- `checklossfunction` (public) : The function used for Armijo rule; see Wang (2007).
- `collapsemix` (public) : The function used for collapsing mixing distributions; see `cnm`.
- `compareattr` (public) : The function used for comparing the attributes `mu0` and `pi0` for performence purposes. If the geometry of `mu0` and `pi0` is close to the stored ones, there is no need to recompute the information related to this pair.
- `solvegradientmultiple` (public) : The function used for computing the new support points. This is function calls for `solvegradientmultipled0`, `solvegradientmultipled1` and `solvegradientmultipled` according to the private component `mflag`. The detailed descriptions of the algorithms used can be found in Appendix B of the thesis. The break points of the smaller intervals are computed through `printgridpoints` implemented in the distributional layer.
- `computemixdist` (public) : The function used for computing the mixing distribution given `mu0fixed` and `pi0fixd`. The result is stored in `result` component.
- `estpi0d` (public) : The function used for computing the derivative when estimate the null proportion. This default implementation can be used in the Newton-Raphson method. This is overwritten by the espti0d in the last layer, which can be used in the Halley method.
- `solvegradientmultipled0` (private), `solvegradientmultipled1` (private) and `solvegradientmultipled2` (private) : The functions for computing the new support points. The underlying computing functions `solvegradientsingled1` and `solvegradientsingled2` are vectorised implementations while `solvegradientsingled0` uses the `optimize`.
- `solveestpi0` (private) : The function for computing the null proportion. The algofithm used depends on `estpi0d`. This is essentially the root-finding algorithm and the formulation can be found in Appendix B of the thesis.
- `gridpoints` (private) : The break points for the smaller interval when computing the new support points. This is initialised by `printgridpoints` in the distributional layer.

The components in the distributional layer are as follows.

- `setgridpoints` (public) : The function used to initialise `gridpoints`. The range of the support can be found in Chapter 3 of the thesis.
- `initpoints` (public) : The function for generating the starting mixing distribution, if not specified in `computemixdist`; see `link[nspmix]{npnorm}` for example.

The componets in the final layer are as follows.

- beta (public) : The structural parameter. The structural parameter is always considered as fixed and can be changed using initisalize and modified.
- type (public) : The flag for component distributions.
- initialize (public) : The function for initialising the object for computing. It sets the data, the length, fixed support points, structural parameter and any precomputed values.
- modified (public) : The function for modifying the object for further computing. It sets the fixed support points, the structural parameter and any precomputed values related to the change of the fixed support points.
- lossfunction (public) : THe function for computing the loss function given a mixing distribution.
- setflexden (public) : This function is used to set the precomputed values for performance purposes.
- gradientfunction (public) : This function computes the gradient for the implemented families. The output is a list of length 3: The gradient d0, the first derivative with respect to the location parameter d1 and the second derivateive with respect to the location parameter d2.
- computeweights (public) : This function computes the new weights given fixed support points, which is needed in computemixdist in the first layer.
- estpi0dS (public) : Generating precomputed values for estimating the null proportion. This is primarily for performance purposes.
- estpi0d (public) : This function overwrites the function of the same name in the first layer for a faster computation of the null proportions.
- estpi0 (public) : This function computes the null proportions depending on the given loss/distance.
- precompute (private), flexden (private), S1 (private) : The object for storing the precomputed values for performance purposes.

References

Wang, Yong. "On Fast Computation of the Non-Parametric Maximum Likelihood Estimate of a Mixing Distribution." Journal of the Royal Statistical Society. Series B (Statistical Methodology) 69, no. 2 (2007): 185-98. http://www.jstor.org/stable/4623262.

---

plotposteriormapping *plot the posterier map*

Description

plot the mapping between the original observations and its posterior mean

Usage

```
plotposteriormapping(x, result, result2 = NULL, ...)
```

Arguments

x	a vector of observations
result	an object of class nspmix to show in the top
result2	an object of class nspmix to show in the bottom. If NULL, then the density in the bottom is not drawn.
...	other parameter passed to plot

Details

This function explicity considers that map between the transformed sample correlation coefficients to the posterior mean of tanh(x) under normal cases. `result` typically is the mixing distribution of the transformed sample correlations and `result2` is the mixing distribution on the sample correlation scale.

Value

none

Examples

```
n = 100; p = 50
X = matrix(rnorm(n * p), nrow = n, ncol = p)
r = cor(X)
x = makeobject(atanh(extractlower(r)), beta = 1 / sqrt(n - 3))
r1 = computemixdist(x)
plotposteriormapping(atanh(extractlower(r)), r1)
```

`posteriormean` *Find the posterior mean*

Description

Find the posterior mean given the observations and the mixing distribution based on the family in the result

Usage

```
posteriormean(x, result, fun = function(x) x)

posteriormean.npnorm(x, result, fun = function(x) x)

posteriormean.npt(x, result, fun = function(x) x)

posteriormean.npnormc(x, result, fun = function(x) x)
```

Arguments

`x`	a vector of observations
`result`	an object of class nspmix
`fun`	the function to transform the mean. It finds the posterior mean of `fun(x)`. The function `fun` must be vectorised.

Examples

```
data = rnorm(500, c(0, 2))
x = makeobject(data, pi0 = 0.5)
r1 = computemixdist(x)
posteriormean(data, r1)
x2 = makeobject(data, pi0 = 0.5, method = "nptll") # equivalent to normal
r2 = computemixdist(x2)
```

```
posteriormean(data, r2, fun = function(x) x^2)
data = runif(500, min = -0.5, max = 0.5)
x3 = makeobject(data, method = "npnormcll", beta = 100)
r3 = computemixdist(x3)
posteriormean(data, r3)
```

`rejectregion.npnorm` *Find the rejection region*

Description

Find the rejection region

Usage

```
rejectregion.npnorm(result, alpha = 0.05)

rejectregion.npt(result, alpha = 0.05)

rejectregion(result, alpha = 0.05)
```

Arguments

`result`	an object class nspmix
`alpha`	the FDR controlling rate.

Details

Find the rejection region based on the family in the result The rejection region is calculated using the density estimate rather than data points hence robust. The rejection is based on the hypothesis is located at 0. The optimisation is done via NLopt library (The package nloptr)

Value

a list with par is the boundary for rejection and area is the propotion of rejection

Examples

```
data = rnorm(500, c(0, 2))
x = makeobject(data, pi0 = 0.5)
r1 = computemixdist(x)
rejectregion(r1)
x2 = makeobject(data, pi0 = 0.5, method = "nptll") # equivalent to normal
r2 = computemixdist(x2)
rejectregion(r2)
```

Index