Package ‘fmrs’

August 25, 2022

Type Package

Title Variable Selection in Finite Mixture of AFT Regression and FMR Models

Version 1.6.0

Description The package obtains parameter estimation, i.e., maximum likelihood estimators (MLE), via the Expectation-Maximization (EM) algorithm for the Finite Mixture of Regression (FMR) models with Normal distribution, and MLE for the Finite Mixture of Accelerated Failure Time Regression (FMAFTR) subject to right censoring with Log-Normal and Weibull distributions via the EM algorithm and the Newton-Raphson algorithm (for Weibull distribution).
More importantly, the package obtains the maximum penalized likelihood (MPLE) for both FMR and FMAFTR models (collectively called FMRs).
A component-wise tuning parameter selection based on a component-wise BIC is implemented in the package.
Furthermore, this package provides Ridge Regression and Elastic Net.

Depends R (>= 4.1.0)

Imports methods, survival, stats

Date 2022-03-28

biocViews Survival, Regression, DimensionReduction

Suggests BiocGenerics, testthat, knitr, utils

License GPL-3

VignetteBuilder knitr

BugReports https://github.com/shokoohi/fmrs/issues

RoxygenNote 7.1.2.9000

Encoding UTF-8

NeedsCompilation yes

git_url https://git.bioconductor.org/packages/fmrs

git_branch RELEASE_3_15

git_last_commit ffc98cb

git_last_commit_date 2022-04-26

Date/Publication 2022-08-25

Author Farhad Shokoohi [aut, cre] (<https://orcid.org/0000-0002-6224-2609>)

Maintainer Farhad Shokoohi <shokoohi@icloud.com>

R topics documented:

`fmrs-package` *Variable Selection in Finite Mixture of AFT Regression and FMR Models*

Description

The package obtains parameter estimation, i.e., maximum likelihood estimators (MLE), via the Expectation-Maximization (EM) algorithm for the Finite Mixture of Regression (FMR) models with Normal distribution, and MLE for the Finite Mixture of Accelerated Failure Time Regression (FMAFTR) subject to right censoring with Log-Normal and Weibull distributions via the EM algorithm and the Newton-Raphson algorithm (for Weibull distribution). More importantly, the package obtains the maximum penalized likelihood (MPLE) for both FMR and FMAFTR models (collectively called FMRs). A component-wise tuning parameter selection based on a component-wise BIC is implemented in the package. Furthermore, this package provides Ridge Regression and Elastic Net. The `fmrs.mle` method provides MLE for FMRs models. The `fmrs.tunsel` method provides component-wise tuning parameters. The `fmrs.varsel` method provides variable selection for FMRs models.

fmrs methods

fmrs.mle, fmrs.tunsel, fmrs.varsel, fmrs.gendata.

fmrs objects

fmrsfit-class, fmrstunpar-class

BIC — BIC method

Description

Provides the estimated BIC of an FMRs model from an fmrsfit-class

Usage

```
BIC(object, ...)

## S4 method for signature 'fmrsfit'
BIC(object, ...)
```

Arguments

object	An fmrsfit-class
...	Other possible arguments

Value

A numeric value

Author(s)

Farhad Shokoohi <shokoohi@icloud.com>

Examples

```
set.seed(1980)
K = 2
D = 10
n = 500
sig = c(1, 1)
piM = c(0.4, 0.6)
r1 = 0.5
coeff1 = c( 2,  2, -1, -2, 1, 2, 0, 0,  0, 0,  0)
coeff2 = c(-1, -1,  1,  2, 0, 0, 0, 0, -1, 2, -2)
Um = 40

dat <- fmrs.gendata(nObs = n, nComp = K, nCov = D, coeff = c(coeff1, coeff2),
```

```
dispersion = sig, mixProp = piM, rho = r1, umax = Um, disFamily = 'lnorm')

res.mle <- fmrs.mle(y = dat$y, x = dat$x, delta = dat$delta, nComp = K,
disFamily = 'lnorm', initCoeff = rnorm(K*D+K), initDispersion = rep(1, K),
initmixProp = rep(1/K, K))

BIC(res.mle)
```

---

coefficients *coefficients method*

Description

Provides the estimated regression coefficients from the fitted FMRs model from an fmrsfit-class

Usage

```
coefficients(object, ...)

## S4 method for signature 'fmrsfit'
coefficients(object, ...)
```

Arguments

object	An fmrsfit-class
...	Other possible arguments

Value

A numeric array of dimension-(nCov+1)-nComp

Author(s)

Farhad Shokoohi <shokoohi@icloud.com>

Examples

```
set.seed(1980)
K = 2
D = 10
n = 500
sig = c(1, 1)
piM = c(0.4, 0.6)
r1 = 0.5
coeff1 = c( 2,  2, -1, -2, 1, 2, 0, 0,  0, 0,  0)
coeff2 = c(-1, -1,  1,  2, 0, 0, 0, 0, -1, 2, -2)
Um = 40

dat <- fmrs.gendata(nObs = n, nComp = K, nCov = D, coeff = c(coeff1, coeff2),
```

```
dispersion = sig, mixProp = piM, rho = r1, umax = Um, disFamily = 'lnorm')

res.mle <- fmrs.mle(y = dat$y, x = dat$x, delta = dat$delta, nComp = K,
disFamily = 'lnorm', initCoeff = rnorm(K*D+K), initDispersion = rep(1, K),
initmixProp = rep(1/K, K))

coefficients(res.mle)
```

---

dispersion *dispersion method*

Description

Provides the estimated dispersions of the fitted FMRs model from an fmrsfit-class

Usage

```
dispersion(object, ...)

## S4 method for signature 'fmrsfit'
dispersion(object, ...)
```

Arguments

object	An fmrsfit-class
...	Other possible arguments

Value

A numeric array of dimension-(nCov+1)-nComp

Author(s)

Farhad Shokoohi <shokoohi@icloud.com>

Examples

```
set.seed(1980)
K = 2
D = 10
n = 500
sig = c(1, 1)
piM = c(0.4, 0.6)
r1 = 0.5
coeff1 = c( 2,  2, -1, -2, 1, 2, 0, 0,  0, 0,  0)
coeff2 = c(-1, -1,  1,  2, 0, 0, 0, 0, -1, 2, -2)
Um = 40

dat <- fmrs.gendata(nObs = n, nComp = K, nCov = D, coeff = c(coeff1, coeff2),
```

```
dispersion = sig, mixProp = piM, rho = r1, umax = Um, disFamily = 'lnorm')

res.mle <- fmrs.mle(y = dat$y, x = dat$x, delta = dat$delta, nComp = K,
disFamily = 'lnorm', initCoeff = rnorm(K*D+K), initDispersion = rep(1, K),
initmixProp = rep(1/K, K))

dispersion(res.mle)
```

fitted *fitted method*

Description

Provides the fitted response of the fitted FMRs model from an fmrsfit-class

Usage

```
fitted(object, ...)

## S4 method for signature 'fmrsfit'
fitted(object, ...)
```

Arguments

object	An fmrsfit-class
...	Other possible arguments

Value

A numeric array of dimension-nObs-nComp

Author(s)

Farhad Shokoohi <shokoohi@icloud.com>

Examples

```
set.seed(1980)
K = 2
D = 10
n = 500
sig = c(1, 1)
piM = c(0.4, 0.6)
r1 = 0.5
coeff1 = c( 2,  2, -1, -2, 1, 2, 0, 0,  0, 0,  0)
coeff2 = c(-1, -1,  1,  2, 0, 0, 0, 0, -1, 2, -2)
Um = 40

dat <- fmrs.gendata(nObs = n, nComp = K, nCov = D, coeff = c(coeff1, coeff2),
```

```
dispersion = sig, mixProp = piM, rho = r1, umax = Um, disFamily = 'lnorm')

res.mle <- fmrs.mle(y = dat$y, x = dat$x, delta = dat$delta, nComp = K,
disFamily = 'lnorm', initCoeff = rnorm(K*D+K), initDispersion = rep(1, K),
initmixProp = rep(1/K, K))

head(fitted(res.mle))
```

`fmrs.gendata` *fmrs.gendata method*

Description

Generates a data set from Finite Mixture of AFT regression models or Finite Mixture of Regression models under the specified setting.

Usage

```
fmrs.gendata(nObs, nComp, nCov, coeff, dispersion, mixProp, rho, umax, ...)

## S4 method for signature 'ANY'
fmrs.gendata(
  nObs,
  nComp,
  nCov,
  coeff,
  dispersion,
  mixProp,
  rho,
  umax,
  disFamily = "lnorm"
)
```

Arguments

`nObs`	A numeric value represents sample size
`nComp`	A numeric value represents the order mixture in `FMRs`
`nCov`	A numeric value represents the number of covariates in design matrix
`coeff`	A vector of all regression coefficients including intercepts. It must be a vector of length `nComp` *(`nCov`+1).
`dispersion`	A vector of positive values for dispersion parameters of sub-distributions in `FMRs` models
`mixProp`	A vector of mixing proportions which their sum must be one
`rho`	A numeric value in [-1, 1] which represents the correlation between covariates of design matrix

umax	A numeric value represents the upper bound in Uniform distribution for censoring
...	Other possible options
disFamily	A sub-distribution family. The options are 'norm' for FMR models, 'lnorm' for mixture of AFT regression models with Log-Normal sub-distributions,'weibull' for mixture of AFT regression models with Weibull sub-distributions

Value

A list including response, covariates and censoring variables

Author(s)

Farhad Shokoohi <shokoohi@icloud.com>

See Also

Other lnorm, norm, weibull: `fmrs.mle()`, `fmrs.tunsel()`, `fmrs.varsel()`

Examples

```
set.seed(1980)
K = 2
D = 10
n = 500
REP = 500
sig = c(1, 1)
piM = c(0.4, 0.6)
r1 = 0.5
coeff1 = c( 2,  2, -1, -2, 1, 2, 0, 0,  0, 0,  0)
coeff2 = c(-1, -1,  1,  2, 0, 0, 0, 0, -1, 2, -2)
Um = 40

dat <- fmrs.gendata(nObs = n, nComp = K, nCov = D, coeff = c(coeff1, coeff2),
dispersion = sig, mixProp = piM, rho = r1, umax = Um, disFamily = 'lnorm')
```

`fmrs.mle` *fmrs.mle method*

Description

Provides MLE for Finite Mixture of Accelerated Failure Time Regression Models or Finite Mixture of Regression Models. It also provides Ridge Regression.

Usage

```
fmrs.mle(y, delta, x, nComp, ...)

## S4 method for signature 'ANY'
fmrs.mle(
  y,
  delta,
  x,
  nComp = 2,
  disFamily = "lnorm",
  initCoeff,
  initDispersion,
  initmixProp,
  lambRidge = 0,
  nIterEM = 400,
  nIterNR = 2,
  conveps = 1e-08,
  convepsEM = 1e-08,
  convepsNR = 1e-08,
  NRpor = 2,
  activeset
)
```

Arguments

`y`	Responses (observations)
`delta`	Censoring indicator vector
`x`	Design matrix (covariates)
`nComp`	Order (Number of components) of mixture model
`...`	Other possible options
`disFamily`	A sub-distribution family. The options are `'norm'` for `FMR` models, `'lnorm'` for mixture of AFT regression models with Log-Normal sub-distributions, `'weibull'` for mixture of AFT regression models with Weibull sub-distributions
`initCoeff`	Vector of initial values for regression coefficients including intercepts
`initDispersion`	Vector of initial values for standard deviations
`initmixProp`	Vector of initial values for proportion of components
`lambRidge`	A positive value for tuning parameter in Ridge Regression or Elastic Net
`nIterEM`	Maximum number of iterations for EM algorithm
`nIterNR`	Maximum number of iterations for Newton-Raphson algorithm
`conveps`	A positive value for avoiding NaN in computing divisions
`convepsEM`	A positive value for threshold of convergence in EM algorithm
`convepsNR`	A positive value for threshold of convergence in Newton-Raphson algorithm
`NRpor`	A positive integer for maximum number of searches in NR algorithm
`activeset`	A matrix of zero-one that shows which intercepts and covariates are active in the fitted fmrs model

Details

Finite mixture of AFT regression models are represented as follows. Let X be the survival time with non-negative values, and $\boldsymbol{z} = (z_1, \ldots, z_d)^\top$ be a d-dimensional vector of covariates that may have an effect on X. If the survival time is subject to right censoring, then the observed response time is $T = \min\{Y, C\}$, where $Y = \log X$, C is logarithm of the censoring time and $\delta = I_{\{y<c\}}$ is the censoring indicator. We say that $V = (T, \delta, \boldsymbol{z})$ follows a finite mixture of AFT regression models of order K if the conditional density of (T, δ) given $\boldsymbol{z}$ has the form

$$f(t, \delta; \boldsymbol{z}, \boldsymbol{\Psi}) = \sum_{k=1}^{K} \pi_k [f_Y(t; \theta_k(\boldsymbol{z}), \sigma_k)]^{\delta} [S_Y(t; \theta_k(\boldsymbol{z}), \sigma_k)]^{1-\delta} [f_C(t)]^{1-\delta} [S_C(t)]^{\delta}$$

where $f_Y(.)$ and $S_Y(.)$ are respectively the density and survival functions of Y, $f_C(.)$ and $S_C(.)$ are respectively the density and survival functions of C; and $\theta_k(\boldsymbol{z}) = h(\beta_{0k} + \boldsymbol{z}^\top \boldsymbol{\beta}_k)$ for a known link function $h(.)$, $\boldsymbol{\Psi} = (\pi_1, \ldots, \pi_K, \beta_{01}, \ldots, \beta_{0K}, \boldsymbol{\beta}_1, \ldots, \boldsymbol{\beta}_K, \sigma_1, \ldots, \sigma_K)^\top$ with $\boldsymbol{\beta}_k = (\beta_{k1}, \beta_{k2}, \ldots, \beta_{kd})^\top$ and $0 < \pi_k < 1$ with $\sum_{k=1}^{K} \pi_k = 1$. The log-likelihood of a sample of size n is formed as

$$\ell_n(\boldsymbol{\Psi}) = \sum_{i=1}^{n} \log \sum_{k=1}^{K} \pi_k \left[f_Y(t_i, \theta_k(\boldsymbol{z}_i), \sigma_k)\right]^{\delta_i} \left[S_Y(t_i, \theta_k(\boldsymbol{z}_i), \sigma_k)\right]^{1-\delta_i}.$$

Note that we assume the censoring distribution is non-informative and hence won't play any role in the estimation process. We use EM and Newton-Raphson algorithms in our method to find the maximizer of above Log-Likelihood.

Value

An `fmrsfit-class` that includes parameter estimates of the specified `FMRs` model

Author(s)

Farhad Shokoohi <shokoohi@icloud.com>

References

Shokoohi, F., Khalili, A., Asgharian, M. and Lin, S. (2016 submitted) Variable Selection in Mixture of Survival Models for Biomedical Genomic Studies

See Also

Other lnorm, norm, weibull: `fmrs.gendata()`, `fmrs.tunsel()`, `fmrs.varsel()`

Examples

```
set.seed(1980)
K = 2
D = 10
n = 500
sig = c(1, 1)
piM = c(0.4, 0.6)
```

```
r1 = 0.5
coeff1 = c( 2,  2, -1, -2, 1, 2, 0, 0,  0, 0,  0)
coeff2 = c(-1, -1,  1,  2, 0, 0, 0, 0, -1, 2, -2)
Um = 40

dat <- fmrs.gendata(nObs = n, nComp = K, nCov = D, coeff = c(coeff1, coeff2),
dispersion = sig, mixProp = piM, rho = r1, umax = Um, disFamily = 'lnorm')

res.mle <- fmrs.mle(y = dat$y, x = dat$x, delta = dat$delta, nComp = K,
disFamily = 'lnorm', initCoeff = rnorm(K*D+K), initDispersion = rep(1, K),
initmixProp = rep(1/K, K))

summary(res.mle)
```

`fmrs.tunsel` *fmrs.tunsel method*

Description

Provides component-wise tuning parameters using BIC for Finite Mixture of Accelerated Failure Time Regression Models and Finite Mixture of Regression Models.

Usage

```
fmrs.tunsel(y, delta, x, nComp, ...)

## S4 method for signature 'ANY'
fmrs.tunsel(
  y,
  delta,
  x,
  nComp,
  disFamily = "lnorm",
  initCoeff,
  initDispersion,
  initmixProp,
  penFamily = "lasso",
  lambRidge = 0,
  nIterEM = 400,
  nIterNR = 2,
  conveps = 1e-08,
  convepsEM = 1e-08,
  convepsNR = 1e-08,
  NRpor = 2,
  gamMixPor = 1,
  activeset,
  lambMCP,
  lambSICA,
```

```
  cutpoint = 0.05,
  LambMin = 0.01,
  LambMax = 1,
  nLamb = 100
)
```

Arguments

`y`	Responses (observations)
`delta`	Censoring indicator vector
`x`	Design matrix (covariates)
`nComp`	Order (Number of components) of mixture model
`...`	Other possible options
`disFamily`	A sub-distribution family. The options are `'norm'` for `FMR` models, `'lnorm'` for mixture of AFT regression models with Log-Normal sub-distributions, `'weibull'` for mixture of AFT regression models with Weibull sub-distributions,
`initCoeff`	Vector of initial values for regression coefficients including intercepts
`initDispersion`	Vector of initial values for standard deviations
`initmixProp`	Vector of initial values for proportion of components
`penFamily`	Penalty name that is used in variable selection method. The available options are `'lasso'`, `'adplasso'`, `'mcp'`, `'scad'`, `'sica'` and `'hard'`.
`lambRidge`	A positive value for tuning parameter in Ridge Regression or Elastic Net
`nIterEM`	Maximum number of iterations for EM algorithm
`nIterNR`	Maximum number of iterations for Newton-Raphson algorithm
`conveps`	A positive value for avoiding NaN in computing divisions
`convepsEM`	A positive value for threshold of convergence in EM algorithm
`convepsNR`	A positive value for threshold of convergence in NR algorithm
`NRpor`	A positive interger for maximum number of searches in NR algorithm
`gamMixPor`	Proportion of mixing parameters in the penalty. The value must be in the interval [0,1]. If `gamMixPor = 0`, the penalty structure is no longer mixture.
`activeset`	A matrix of zero-one that shows which intercepts and covariates are active in the fitted fmrs model
`lambMCP`	A positive numbers for `mcp`'s extra tuning parameter
`lambSICA`	A positive numbers for `sica`'s extra tuning parameter
`cutpoint`	A positive value for shrinking small values of parameter estimations in the EM algorithm toward zero
`LambMin`	A positive value for minimum value of tuning parameter
`LambMax`	A positive value for maximum value of tuning parameter
`nLamb`	A positive value for number of tuning parameter

Details

The maximizer of penalized Log-Likelihood depends on selecting a set of good tuning parameters which is a rather thorny issue. We choose a value in an equally spaced set of values in $(0, \lambda_{max})$ for a pre-specified λ_{max} that maximize the component-wise BIC,

$$\hat{\lambda}_k = argmax_{\lambda_k} BIC_k(\lambda_k) = argmax_{\lambda_k} \left\{ \ell^c_{k,n}(\hat{\mathbf{\Psi}}_{\lambda_k,k}) - |d_{\lambda_k,k}| \log(n) \right\},$$

where $d_{\lambda_k,k} = \{j : \hat{\beta}_{\lambda_k,kj} \neq 0, j = 1, \ldots, d\}$ is the active set excluding the intercept and $|d_{\lambda_k,k}|$ is its size. This approach is much faster than using an `nComp` by `nComp` grid to select the set $\boldsymbol{\lambda}$ to maximize the penallized Log-Likelihood.

Value

An `fmrstunpar-class` that includes component-wise tuning parameter estimates that can be used in variable selection procedure.

Author(s)

Farhad Shokoohi <shokoohi@icloud.com>

References

Shokoohi, F., Khalili, A., Asgharian, M. and Lin, S. (2016 submitted) Variable Selection in Mixture of Survival Models for Biomedical Genomic Studies

See Also

Other lnorm, norm, weibull: `fmrs.gendata()`, `fmrs.mle()`, `fmrs.varsel()`

Examples

```
set.seed(1980)
K = 2
D = 10
n = 500
sig = c(1, 1)
piM = c(0.4, 0.6)
r1 = 0.5
coeff1 = c( 2,  2, -1, -2, 1, 2, 0, 0,  0, 0,  0)
coeff2 = c(-1, -1,  1,  2, 0, 0, 0, 0, -1, 2, -2)
Um = 40

dat <- fmrs.gendata(nObs = n, nComp = K, nCov = D, coeff = c(coeff1, coeff2),
dispersion = sig, mixProp = piM, rho = r1, umax = Um, disFamily = 'lnorm')

res.mle <- fmrs.mle(y = dat$y, x = dat$x, delta = dat$delta, nComp = K,
disFamily = 'lnorm', initCoeff = rnorm(K*D+K), initDispersion = rep(1, K),
initmixProp = rep(1/K, K))

res.lam <- fmrs.tunsel(y = dat$y, x = dat$x, delta = dat$delta, nComp = K,
disFamily = 'lnorm', initCoeff = c(coefficients(res.mle)),
```

```
initDispersion = dispersion(res.mle), initmixProp = mixProp(res.mle),
penFamily = 'adplasso')

show(res.lam)
```

fmrs.varsel *fmrs.varsel method*

Description

Provides variable selection and penalized MLE for Finite Mixture of Accelerated Failure Time Regression (FMAFTR) Models and Finite Mixture of Regression (FMR) Models. It also provide Ridge Regression and Elastic Net.

Usage

```
fmrs.varsel(y, delta, x, nComp, ...)

## S4 method for signature 'ANY'
fmrs.varsel(
  y,
  delta,
  x,
  nComp,
  disFamily = "lnorm",
  initCoeff,
  initDispersion,
  initmixProp,
  penFamily = "lasso",
  lambPen,
  lambRidge = 0,
  nIterEM = 2000,
  nIterNR = 2,
  conveps = 1e-08,
  convepsEM = 1e-08,
  convepsNR = 1e-08,
  NRpor = 2,
  gamMixPor = 1,
  activeset,
  lambMCP,
  lambSICA,
  cutpoint = 0.05
)
```

Arguments

y Responses (observations)

`delta`	Censoring indicators
`x`	Design matrix (covariates)
`nComp`	Order (Number of components) of mixture model
`...`	Other possible options
`disFamily`	A sub-distribution family. The options are `'norm'` for FMR models, `'lnorm'` for mixture of AFT regression models with Log-Normal sub-distributions, `'weibull'` for mixture of AFT regression models with Weibull sub-distributions
`initCoeff`	Vector of initial values for regression coefficients including intercepts
`initDispersion`	Vector of initial values for standard deviations
`initmixProp`	Vector of initial values for proportion of components
`penFamily`	Penalty name that is used in variable selection method The available options are `'lasso'`, `'adplasso'`, `'mcp'`, `'scad'`, `'sica'` and `'hard'`.
`lambPen`	A vector of positive numbers for tuning parameters
`lambRidge`	A positive value for tuning parameter in Ridge Regression or Elastic Net
`nIterEM`	Maximum number of iterations for EM algorithm
`nIterNR`	Maximum number of iterations for Newton-Raphson algorithm
`conveps`	A positive value for avoiding NaN in computing divisions
`convepsEM`	A positive value for threshold of convergence in EM algorithm
`convepsNR`	A positive value for threshold of convergence in NR algorithm
`NRpor`	A positive interger for maximum number of searches in NR algorithm
`gamMixPor`	Proportion of mixing parameters in the penalty. The value must be in the interval [0,1]. If `gamMixPor = 0`, the penalty structure is no longer mixture.
`activeset`	A matrix of zero-one that shows which intercepts and covariates are active in the fitted fmrs model
`lambMCP`	A positive numbers for `mcp`'s extra tuning parameter
`lambSICA`	A positive numbers for `sica`'s extra tuning parameter
`cutpoint`	A positive value for shrinking small values of parameter estimations in the EM algorithm tward zero

Details

The penalized likelihood of a finite mixture of AFT regression models is written as

$$\tilde{\ell}_n(\boldsymbol{\Psi}) = \ell_n(\boldsymbol{\Psi}) - \mathbf{p}_{\boldsymbol{\lambda}_n}(\boldsymbol{\Psi})$$

where

$$\mathbf{p}_{\boldsymbol{\lambda}_n}(\boldsymbol{\Psi}) = \sum_{k=1}^{K} \pi_k^{\alpha} \left\{ \sum_{j=1}^{d} p_{\lambda_{n,k}}(\beta_{kj}) \right\}.$$

In the M step of EM algorithm the function

$$\tilde{Q}(\boldsymbol{\Psi}, \boldsymbol{\Psi}^{(m)}) = \sum_{k=1}^{K} \tilde{Q}_k(\boldsymbol{\Psi}_k, \boldsymbol{\Psi}_k^{(m)}) = \sum_{k=1}^{K} \left[Q_k(\boldsymbol{\Psi}_k, \boldsymbol{\Psi}_k^{(m)}) - \pi_k^{\alpha} \left\{ \sum_{j=1}^{d} p_{\lambda_{n,k}}(\beta_{kj}) \right\} \right]$$

is maximized. Since the penalty function is singular at origin, we use a local quadratic approximation (LQA) for the penalty as follows,

$$\mathbf{p}^{*}_{k,\boldsymbol{\lambda}_n}(\boldsymbol{\beta},\boldsymbol{\beta}^{(m)}) = (\pi_k^{(m)})^{\alpha}\sum_{j=1}^{d}\left\{p_{\lambda_{n,k}}(\beta_{kj}^{(m)}) + \frac{p'_{\lambda_{n,k}}(\beta_{kj}^{(m)})}{2\beta_{kj}^{(m)}}(\beta_{kj}^{2} - {\beta_{kj}^{(m)}}^{2})\right\}.$$

Therefore maximizing Q is equivalent to maximizing the function

$$Q^{*}(\boldsymbol{\Psi},\boldsymbol{\Psi}^{(m)}) = \sum_{k=1}^{K} Q_k^{*}(\boldsymbol{\Psi}_k,\boldsymbol{\Psi}_k^{(m)}) = \sum_{k=1}^{K}\left[Q_k(\boldsymbol{\Psi}_k,\boldsymbol{\Psi}_k^{(m)}) - \mathbf{p}^{*}_{k,\boldsymbol{\lambda}_n}(\boldsymbol{\beta},\boldsymbol{\beta}^{(m)})\right].$$

In case of Log-Normal sub-distributions, the maximizers of Q_k functions are as follows. Given the data and current estimates of parameters, the maximizers are

$$\boldsymbol{\beta}_k^{(m+1)} = (\boldsymbol{z}'\boldsymbol{\tau}_k^{(m)}\boldsymbol{z} + \varpi_k(\boldsymbol{\beta}_{kj}^{(m)}))^{-1}\boldsymbol{z}'\boldsymbol{\tau}_k^{(m)}T_k^{(m)},$$

where $\varpi_k(\boldsymbol{\beta}_{kj}^{(m)}) = diag\left(\left(\pi_k^{(m+1)}\right)^{\alpha}\frac{p'_{\lambda_{n,k}}(\boldsymbol{\beta}_{kj}^{(m)})}{\boldsymbol{\beta}_{kj}^{(m)}}\right)$ and $\sigma_k^{(m+1)}$ is equal to

$$\sigma_k^{(m+1)} = \sqrt{\frac{\sum_{i=1}^{n}\tau_{ik}^{(m)}(t_{ik}^{(m)} - \boldsymbol{z}_i\boldsymbol{\beta}_k^{(m)})^2}{\sum_{i=1}^{n}\tau_{ik}^{(m)}\left[\delta_i + (1-\delta_i)\{A(w_{ik}^{(m)})[A(w_{ik}^{(m)}) - w_{ik}^{(m)}]\}\right]}}.$$

For the Weibull distribution, on the other hand, we have $\tilde{\boldsymbol{\Psi}}_k^{(m+1)} = \tilde{\boldsymbol{\Psi}}_k^{(m)} - 0.5^{\kappa}\left[H_k^{p,(m)}\right]^{-1} I_k^{p,(m)}$, where $H_k^p = H_k + h(\boldsymbol{\Psi}_k)$ is the penalized version of hessian matrix and $I_k^p = I_k + h(\boldsymbol{\Psi}_k)\boldsymbol{\Psi}_k$ is the penalized version of vector of first derivatives evaluated at $\tilde{\boldsymbol{\Psi}}_k^{(m)}$.

Value

`fmrsfit-class`

Author(s)

Farhad Shokoohi <shokoohi@icloud.com>

References

Shokoohi, F., Khalili, A., Asgharian, M. and Lin, S. (2016 submitted) Variable Selection in Mixture of Survival Models for Biomedical Genomic Studies

See Also

Other lnorm, norm, weibull: `fmrs.gendata()`, `fmrs.mle()`, `fmrs.tunsel()`

Examples

```
set.seed(1980)
K = 2
D = 10
n = 500
sig = c(1, 1)
piM = c(0.4, 0.6)
r1 = 0.5
coeff1 = c( 2,  2, -1, -2, 1, 2, 0, 0,  0, 0,  0)
coeff2 = c(-1, -1,  1,  2, 0, 0, 0, 0, -1, 2, -2)
Um = 40

dat <- fmrs.gendata(nObs = n, nComp = K, nCov = D, coeff = c(coeff1, coeff2),
dispersion = sig, mixProp = piM, rho = r1, umax = Um, disFamily = 'lnorm')

res.mle <- fmrs.mle(y = dat$y, x = dat$x, delta = dat$delta, nComp = K,
disFamily = 'lnorm', initCoeff = rnorm(K*D+K), initDispersion = rep(1, K),
initmixProp = rep(1/K, K))

res.lam <- fmrs.tunsel(y = dat$y, x = dat$x, delta = dat$delta,
nComp = ncomp(res.mle), disFamily = 'lnorm',
initCoeff = c(coefficients(res.mle)), initDispersion = dispersion(res.mle),
initmixProp = mixProp(res.mle), penFamily = 'adplasso')

res.var <- fmrs.varsel(y = dat$y, x = dat$x, delta = dat$delta,
nComp = ncomp(res.mle), disFamily = 'lnorm',
initCoeff = c(coefficients(res.mle)), initDispersion = dispersion(res.mle),
initmixProp = mixProp(res.mle), penFamily = 'adplasso',
lambPen = slot(res.lam, 'lambPen'))

summary(res.var)
```

---

fmrsfit-class — *An S4 class to represent a fitted FMRs model*

Description

is an S4 class represents a fitted of FMRs model resulted from running `fmrs.mle` or `fmrs.varsel`

Slots

`y` A length-nobs numeric vector

`delta` A length-nobs numeric vector

`x` A dimension-nobs-ncov numeric matrix

`nobs` A length-one numeric vector

`ncov` A length-one numeric vector

`ncomp` A length-one numeric vector

`coefficients` A length-(ncov+1)-ncomp numeric matrix

`dispersion` A length-ncomp numeric vector

`mixProp` A length-ncomp numeric vector

`logLik` A length-one numeric vector

`BIC` A length-one numeric vector

`nIterEMconv` A length-one numeric vector

`disFamily` A length-one character vector

`penFamily` A length-one character vector

`lambPen` A length-ncomp numeric vector

`lambRidge` A length-one numeric vector

`MCPGam` A length-one numeric vector

`SICAGam` A length-one numeric vector

`model` A length-one character vector

`fitted` A dimension-nobs-ncomp numeric matrix

`residuals` A dimension-nobs-ncomp numeric matrix

`weights` A dimension-nobs-ncomp numeric matrix

`activeset` A dimension-ncov+1-ncomp 0-1 matrix

`selectedset` A dimension-ncov-ncomp 0-1 matrix

`fmrstunpar-class` *An S4 class to represent estimated optimal lambdas*

Description

An S4 class to represent estimated optimal lambdas resulted from running `fmrs.tunsel`

Slots

`ncov` A length-one numeric vector

`ncomp` A length-one numeric vector

`lambPen` A dimension-one-ncomp numeric array

`MCPGam` A length-one numeric vector

`SICAGam` A length-one numeric vector

`disFamily` A length-one character vector

`penFamily` A length-one character vector

`lambRidge` A length-one numeric vector

`model` A length-one character vector

`activeset` A dimension-nobs-ncomp 0-1 matrix logLik *logLik method*

Description

Provides the estimated logLikelihood of an FMRs model from an fmrsfit-class

Usage

```
logLik(object, ...)

## S4 method for signature 'fmrsfit'
logLik(object, ...)
```

Arguments

object	An fmrsfit-class
...	Other possible arguments

Value

A numeric value

Author(s)

Farhad Shokoohi <shokoohi@icloud.com>

Examples

```
set.seed(1980)
K = 2
D = 10
n = 500
sig = c(1, 1)
piM = c(0.4, 0.6)
r1 = 0.5
coeff1 = c( 2,  2, -1, -2, 1, 2, 0, 0,  0, 0,  0)
coeff2 = c(-1, -1,  1,  2, 0, 0, 0, 0, -1, 2, -2)
Um = 40

dat <- fmrs.gendata(nObs = n, nComp = K, nCov = D, coeff = c(coeff1, coeff2),
dispersion = sig, mixProp = piM, rho = r1, umax = Um, disFamily = 'lnorm')

res.mle <- fmrs.mle(y = dat$y, x = dat$x, delta = dat$delta, nComp = K,
disFamily = 'lnorm', initCoeff = rnorm(K*D+K), initDispersion = rep(1, K),
initmixProp = rep(1/K, K))

logLik(res.mle)
```

`mixProp` *mixProp method*

Description

Provides the estimated mixing proportions of an `FMRs` model form an `fmrsfit-class`

Usage

```
mixProp(object, ...)

## S4 method for signature 'fmrsfit'
mixProp(object, ...)
```

Arguments

`object`	An `fmrsfit-class`
`...`	Other possible arguments

Value

A numeric vector of length-`nComp`

Author(s)

Farhad Shokoohi <shokoohi@icloud.com>

Examples

```
set.seed(1980)
K = 2
D = 10
n = 500
sig = c(1, 1)
piM = c(0.4, 0.6)
r1 = 0.5
coeff1 = c( 2,  2, -1, -2, 1, 2, 0, 0,  0, 0,  0)
coeff2 = c(-1, -1,  1,  2, 0, 0, 0, 0, -1, 2, -2)
Um = 40

dat <- fmrs.gendata(nObs = n, nComp = K, nCov = D, coeff = c(coeff1, coeff2),
dispersion = sig, mixProp = piM, rho = r1, umax = Um, disFamily = 'lnorm')

res.mle <- fmrs.mle(y = dat$y, x = dat$x, delta = dat$delta, nComp = K,
disFamily = 'lnorm', initCoeff = rnorm(K*D+K), initDispersion = rep(1, K),
initmixProp = rep(1/K, K))

mixProp(res.mle)
```

`ncomp` *ncomp method*

Description

Provides the order of an `FMRs` model from an `fmrsfit-class`

Usage

```
ncomp(object, ...)

## S4 method for signature 'fmrsfit'
ncomp(object, ...)
```

Arguments

`object`	An `fmrsfit-class`
`...`	Other possible arguments

Value

An integer value

Author(s)

Farhad Shokoohi <shokoohi@icloud.com>

Examples

```
set.seed(1980)
K = 2
D = 10
n = 500
sig = c(1, 1)
piM = c(0.4, 0.6)
r1 = 0.5
coeff1 = c( 2,  2, -1, -2, 1, 2, 0, 0,  0, 0,  0)
coeff2 = c(-1, -1,  1,  2, 0, 0, 0, 0, -1, 2, -2)
Um = 40

dat <- fmrs.gendata(nObs = n, nComp = K, nCov = D, coeff = c(coeff1, coeff2),
dispersion = sig, mixProp = piM, rho = r1, umax = Um, disFamily = 'lnorm')

res.mle <- fmrs.mle(y = dat$y, x = dat$x, delta = dat$delta, nComp = K,
disFamily = 'lnorm', initCoeff = rnorm(K*D+K), initDispersion = rep(1, K),
initmixProp = rep(1/K, K))

ncomp(res.mle)
```

`ncov` *ncov method*

Description

Provides the number of covariates of an FMRs model from an `fmrsfit-class`

Usage

```
ncov(object, ...)

## S4 method for signature 'fmrsfit'
ncov(object, ...)
```

Arguments

`object`	An `fmrsfit-class`
`...`	Other possible arguments

Value

An integer value

Author(s)

Farhad Shokoohi <shokoohi@icloud.com>

Examples

```
set.seed(1980)
K = 2
D = 10
n = 500
sig = c(1, 1)
piM = c(0.4, 0.6)
r1 = 0.5
coeff1 = c( 2,  2, -1, -2, 1, 2, 0, 0,  0, 0,  0)
coeff2 = c(-1, -1,  1,  2, 0, 0, 0, 0, -1, 2, -2)
Um = 40

dat <- fmrs.gendata(nObs = n, nComp = K, nCov = D, coeff = c(coeff1, coeff2),
dispersion = sig, mixProp = piM, rho = r1, umax = Um, disFamily = 'lnorm')

res.mle <- fmrs.mle(y = dat$y, x = dat$x, delta = dat$delta, nComp = K,
disFamily = 'lnorm', initCoeff = rnorm(K*D+K), initDispersion = rep(1, K),
initmixProp = rep(1/K, K))

ncov(res.mle)
```

nobs *nobs method*

Description

Provides the number of observations in an FMRs model from an fmrsfit-class

Usage

```
nobs(object, ...)

## S4 method for signature 'fmrsfit'
nobs(object, ...)
```

Arguments

object	An fmrsfit-class
...	Other possible arguments

Value

An integer value

Author(s)

Farhad Shokoohi <shokoohi@icloud.com>

Examples

```
set.seed(1980)
K = 2
D = 10
n = 500
sig = c(1, 1)
piM = c(0.4, 0.6)
r1 = 0.5
coeff1 = c( 2,  2, -1, -2, 1, 2, 0, 0,  0, 0,  0)
coeff2 = c(-1, -1,  1,  2, 0, 0, 0, 0, -1, 2, -2)
Um = 40

dat <- fmrs.gendata(nObs = n, nComp = K, nCov = D, coeff = c(coeff1, coeff2),
dispersion = sig, mixProp = piM, rho = r1, umax = Um, disFamily = 'lnorm')

res.mle <- fmrs.mle(y = dat$y, x = dat$x, delta = dat$delta, nComp = K,
disFamily = 'lnorm', initCoeff = rnorm(K*D+K), initDispersion = rep(1, K),
initmixProp = rep(1/K, K))

nobs(res.mle)
```

residuals *residuals method*

Description

Provides the residuals of the fitted FMRs model from an fmrsfit-class

Usage

```
residuals(object, ...)

## S4 method for signature 'fmrsfit'
residuals(object, ...)
```

Arguments

object	An fmrsfit-class
...	Other possible arguments

Value

A numeric array of dimension-nObs-nComp

Author(s)

Farhad Shokoohi <shokoohi@icloud.com>

Examples

```
set.seed(1980)
K = 2
D = 10
n = 500
sig = c(1, 1)
piM = c(0.4, 0.6)
r1 = 0.5
coeff1 = c( 2,  2, -1, -2, 1, 2, 0, 0,  0, 0,  0)
coeff2 = c(-1, -1,  1,  2, 0, 0, 0, 0, -1, 2, -2)
Um = 40

dat <- fmrs.gendata(nObs = n, nComp = K, nCov = D, coeff = c(coeff1, coeff2),
dispersion = sig, mixProp = piM, rho = r1, umax = Um, disFamily = 'lnorm')

res.mle <- fmrs.mle(y = dat$y, x = dat$x, delta = dat$delta, nComp = K,
disFamily = 'lnorm', initCoeff = rnorm(K*D+K), initDispersion = rep(1, K),
initmixProp = rep(1/K, K))

head(residuals(res.mle))
```

summary *summary method*

Description

Displays the fitted FMRs model by showing the estimated coefficients, dispersions and mixing proportions

Display the selected component-wise tuning parameters

Usage

```
summary(object, ...)

summary(object, ...)

## S4 method for signature 'fmrsfit'
summary(object, ...)

## S4 method for signature 'fmrstunpar'
summary(object, ...)
```

Arguments

object	An fmrsfit-class or fmrstunpar-class
...	Other possible arguments

Value

Summary of the fitted FMRs model

Summary of the selected component-wise tuning parameters

Author(s)

Farhad Shokoohi <shokoohi@icloud.com>

Farhad Shokoohi <shokoohi@icloud.com>

Examples

```
set.seed(1980)
K = 2
D = 10
n = 500
sig = c(1, 1)
piM = c(0.4, 0.6)
r1 = 0.5
coeff1 = c( 2,  2, -1, -2, 1, 2, 0, 0,  0, 0,  0)
coeff2 = c(-1, -1,  1,  2, 0, 0, 0, 0, -1, 2, -2)
```

```
Um = 40

dat <- fmrs.gendata(nObs = n, nComp = K, nCov = D, coeff = c(coeff1, coeff2),
dispersion = sig, mixProp = piM, rho = r1, umax = Um, disFamily = 'lnorm')

res.mle <- fmrs.mle(y = dat$y, x = dat$x, delta = dat$delta, nComp = K,
disFamily = 'lnorm', initCoeff = rnorm(K*D+K), initDispersion = rep(1, K),
initmixProp = rep(1/K, K))

summary(res.mle)
res.lam <- fmrs.tunsel(y = dat$y, x = dat$x, delta = dat$delta,
nComp = K, disFamily = 'lnorm', initCoeff = c(coefficients(res.mle)),
initDispersion = dispersion(res.mle), initmixProp = mixProp(res.mle),
penFamily = 'adplasso')

summary(res.lam)
```

`weights` *weights method*

Description

Provides the weights of fitted observations for each observation under all components of an `FMRs` model

Usage

```
weights(object, ...)

## S4 method for signature 'fmrsfit'
weights(object, ...)
```

Arguments

`object`	An `fmrsfit-class`
`...`	Other possible arguments

Value

A numeric array of dimension-`nObs`-`nComp`

Author(s)

Farhad Shokoohi <shokoohi@icloud.com>

Examples

```
set.seed(1980)
K = 2
D = 10
n = 500
sig = c(1, 1)
piM = c(0.4, 0.6)
r1 = 0.5
coeff1 = c( 2,  2, -1, -2, 1, 2, 0, 0,  0, 0,  0)
coeff2 = c(-1, -1,  1,  2, 0, 0, 0, 0, -1, 2, -2)
Um = 40

dat <- fmrs.gendata(nObs = n, nComp = K, nCov = D, coeff = c(coeff1, coeff2),
dispersion = sig, mixProp = piM, rho = r1, umax = Um, disFamily = 'lnorm')

res.mle <- fmrs.mle(y = dat$y, x = dat$x, delta = dat$delta, nComp = K,
disFamily = 'lnorm', initCoeff = rnorm(K*D+K), initDispersion = rep(1, K),
initmixProp = rep(1/K, K))

head(weights(res.mle))
```

Index