Package 'moc'

February 28, 2019

Version 2.0

Date 2019-02-15

Title General Nonlinear Multivariate Finite Mixtures

Depends R (>= 3.5.0)

Description Fits and vizualize user defined finite mixture models for multivariate observations using maximum likelihood. (McLachlan, G., Peel, D. (2000) Finite Mixture Models. Wiley-Interscience.)

License GPL-2

NeedsCompilation yes

Author Bernard Boulerice [aut, cre]

Maintainer Bernard Boulerice <bernard.boulerice.bb@gmail.com>

Repository CRAN

Date/Publication 2019-02-28 22:32:16 UTC

R topics documented:

AIC.moc *Information criterions for MOC models.*

Description

AIC.moc generates a table of $\log(Likelihood)$, AIC, BIC, ICL-BIC and entropy values along with the degrees of freedom of multiple moc objects.

logLik returns an object of class logLik containing the $\log(Likelihood)$, degrees of freedom and number of observations.

loglike.moc computes the $\log(Likelihood)$ of a moc object evaluated at the supplied parameters values, contrary to logLik above which uses the estimated parameter values. It gives the option to re-evaluate the model in which case the supplied parameter values are used as new starting values.

entropy is a generic method to compute the entropy of sets of probabilities.

The entropy of a set of k probabilities $(\pi_1, \ldots, \pi_k)$ is computed as $entropy = -\sum_{i=1}^{k}(\pi_i \log(\pi_i))$, it reaches its minimum of 0 when one of the $\pi_i = 1$ (minimum uncertainty) and its maximum of $\log(k)$ when all probabilities are equal to $\pi_i = 1/k$ (maximum uncertainty). Standardized entropy is just $entropy/\log(k)$ which lies in the interval $[0, 1]$. The total and mean mixture entropy are the weighted sum and mean of the mixture probabilities entropy of all subjects. These are computed for both the prior (without knowledge of the response patterns) and the posterior mixture probabilities (with knowledge of the responses).

The default method entropy.default compute entropy and standardized entropy of a set of probabilities.

entropy.moc generates a table containing weighted total and mean standardized entropy of prior and posterior mixture probabilities of moc models.

Usage

```
## S3 method for class 'moc'
AIC(object, ..., k = 2)

## S3 method for class 'moc'
logLik(object, ...)

loglike.moc(object, parm = object$coef, evaluate = FALSE)

## S3 method for class 'moc'
entropy(object, ...)
```

Arguments

object, ...	Objects of class moc.
k	Can be any real number or the string "BIC".

`parm`	Parameters values at which the $\log(Likelihood)$ is evaluated.
`evaluate`	Boolean indicating whether re-evaluation of the model is desired. If `TRUE` `parm` will be used as new starting values.

Details

The computed value in `AIC.moc` is $-2 \cdot \log(Likelihood) + k \cdot npar$. Specific treatment is carried for BIC ($k = \log(nsubject \cdot nvar)$), AIC ($k = 2$) and $\log(Likelihood)$ ($k = 0$). Setting k = "BIC", will produce a table with BIC, mixture posterior $entropy = -\sum_{i,k}(wt_i \cdot \hat{\tau}_{i,k} \log(\hat{\tau}_{i,k}))$ which is an indicator of mixture separation, df and $ICL - BIC = BIC + 2 \cdot entropy$ which is an entropy corrected BIC, see McLachlan, G. and Peel, D. (2000) and Biernacki, C. et al. (2000).

Value

`AIC.moc` returns a data frame with the relevant information for one or more `moc` objects.

The likelihood methods works on a single `moc` object: `logLik.moc` returns an object of class `logLik` with attributes *df*, *nobs* and *moc.name* while `loglike.moc` returns a matrix containing $\log(Likelihood)$ and corresponding estimated parameters with attributes *moc.name* and *parameters*.

`entropy.moc` returns a `data.frame` with number of groups, total and mean standardized prior and posterior entropy of multiple `moc` objects. The percentage of reduction from prior to posterior entropy within a model is also supplied.

Note

Be aware that degrees of freedom (df) for mixture models are usually useless (if not meaningless) and likelihood-ratio of *apparently* nested models often doesn't converge to a Chi-Square with corresponding df.

Author(s)

Bernard Boulerice <<bernard.boulerice.bb@gmail.com>>

References

McLachlan, G. and Peel, D. (2000) *Finite mixture models*, Wiley-Interscience, New York.

Biernacki, C., Celeux, G., Govaert, G. (2000) *Assessing a Mixture Model with the Integrated Completed Likelihood*, IEEE Transaction on Pattern Analysis and Machine Learning, **22**, pp. 719–725.

See Also

`moc`, `confint.moc`, `profiles.postCI`, `entropyplot.moc`, `npmle.gradient`

`confint.moc` *Parameter and profiles confidence intervals and likelihood profiling of MOC models.*

Description

`confint.moc` computes confidence intervals (CI) of specified function of the parameters based on crude Wald asymptotics. More precise CI for the original parameters are obtained through profiling of the likelihood function (that is evaluation of the likelihood over a wide range of values in the parameter space). When profiling is requested the deviance over different parameters' values is also returned.

`profiles.postCI` computes data values for which the empirical probability of observing such subject values, given mixture group, lies between the confidence bounds (see details).

`density.moc` computes the estimated mixture density at data points along some factor and optionally plot it.

Usage

```
## S3 method for class 'moc'
confint(object, parm = list(), level = 0.95,
        profiling = c("none","simple","complete"), ...)

profiles.postCI(object, data = NULL, level = 0.95,
                interpolate = TRUE)

## S3 method for class 'moc'
density(x, var = NULL, along = NULL,
            plot = c("none","pp-plot","density","pq-plot"),
            type = "l", ...)
```

Arguments

`object, x`	A fitted `moc` object.
`parm`	A list of formulas beginning with `~` or expressions of the parameters denoted $p1, p2, \ldots$ for which confidence intervals are requested. For example, `parm = list(~p1,~exp(p2)/(1+exp(p1+p3)))`.
`level`	Alpha level in the interval $(0, 1)$ for the confidence bounds $[(1 - level)/2, (1 + level)/2]$. In `profiles.postCI`, you can also directly specify the bounds like `level = c(0.02,0.98)`.
`profiling`	A string that specifies the desired type of likelihood profiling. This can be one of **none:** (default) no profiling. **simple:** evaluate the likelihood on a grid of values, one parameters at a time holding the other parameters fixed.

complete: fix one parameter at a time over a grid of values and re-estimate the other parameters.

`data` An optional `data.frame`, `matrix` or `vector` of length `nsubject` containing the values for which the confidence limits are requested. The default is to take the original response profile.

`interpolate` A logical value indicating whether interpolation of data values must be performed to achieve the probability limits. When "FALSE", the data points with the probabilities nearest to the given bounds are taken (thus using the corresponding step function).

`var` A vector of integer values specifying the response variables for density evaluation.

`along` A factor used to split the density estimator.

`plot` A string that specifies the kind of desired plot. Allowed values are

none: the default.

pp-plot: plot the estimated mixture cumulative probability (CDF) against its empirical counterpart.

pq-plot: plot the estimated CDF against the quintiles.

density: plot the estimated mixture distribution at observed data points.

`type` The type of lines in the plot, see `plot` for details.

`...` Used in `density.moc` to pass arguments directly to the plotting function. In `confint.moc` iterlim will be passed to `update.moc` and offscal will change the profiling parameters search range.

Details

The methods included here primarily exploit the empirical estimators of the conditional expectation given mixture group for some appropriately chosen function of the data $g()$, that is

$$\hat{g}_k = \frac{\sum_i wt_i\, \hat{\tau}_{i,k}\, g(y_i)}{\sum_i wt_i\, \hat{\tau}_{i,k}}.$$

Profiles confidence intervals and density estimates are defined by choosing $g()$ as the indicator function over appropriate sets. See `print.moc` and `residuals.moc`.

Value

`confint.moc` returns a list containing a list of arrays with likelihood deviance for each parameters configuration of the requested profiling, a function `ellip` corresponding to the asymptotic elliptic distance

$$ellip(p) = (p - \hat{p})\hat{\Sigma}^{-1}(p - \hat{p})$$

where $\hat{p}$ is the maximum likelihood estimator of the parameters and $\hat{\Sigma}$ its asymptotic covariance matrix. It also returns univariate, joint conditional and likelihood rejection confidence intervals for the parameters (when profiling has been requested).

`profiles.postCI` returns a list of array with upper and lower bounds on *data* profiles for each mixture group.

`density.moc` returns nothing when a plot is requested, otherwise an array with mixture density estimate and data points is returned.

Note

Mixture models are powerful tools to capture and describe the variability present in the data. The methods `profiles.postCI` and `density.moc` are especially intended to this purpose. The method `confint.moc` is quite different in essence since it is used to describe parameters' uncertainty that depends on sampling scheme and size, estimation method and goodness-of-fit of the model. A model with small parameters' uncertainty can poorly describe data variability while a model with large parameters' uncertainty can be very good at describing data variability.

Author(s)

Bernard Boulerice <<bernard.boulerice.bb@gmail.com>>

See Also

`moc`, `print.moc`, `residuals.moc`, `post.moc`, `loglike.moc`, `profilesplot`

`moc` *Fit a General Nonlinear Multivariate Mixture Model (MOC)*

Description

`moc` fits user-specified mixture models with one, two and three parameters distributions to multivariate data that can be of discrete or continuous type and a mix of both types. The likelihood for the vector of observations or repeated measurements for subject i has the form

$$f(Y_i = y_i |\, Z_i = z_i, X_i = x_i) = \sum_k \pi_k(z_i, x_i)\, h_k(y_i |\, x_i)$$

Here, $\pi_k()$ represent the mixture probability function and $h_k()$ the conditional joint density of the observations Y_i given the covariates X_i and mixture k. The user supplies either the joint or marginal conditional density(ies) of the components of Y_i. In the latter case, the joint conditional density is constructed by taking the product of the marginal densities (assuming conditional independence of the components).

The `update.moc` function allows to update or modify an already fitted `moc` object and to put constraint on its parameters.

Usage

```
moc(y, density = NULL, joint = FALSE, groups = 1,
    gmu = NULL, gshape = NULL, gextra = NULL,
    gmixture = inv.glogit, expected = NULL,
    pgmu = NULL, pgshape = NULL, pgextra = NULL, pgmix = NULL,
    check.length = TRUE, scale.weight = FALSE, wt = 1, data = NULL,
    ndigit = 10, gradtol = 0.0001, steptol = gradtol,
    iterlim = 100, print.level = 1, ...)
```

```
## S3 method for class 'moc'
update(object, groups = 1:object$groups, parm = object$coef,
           what = NULL, evaluate = FALSE, ...)
```

Arguments

y	A matrix or data frame giving the vectors of observations (of length $nvar$) for each subject.
object	A moc object to update.
density	A function returning the conditional joint or marginal density of the observations and calling the location, shape and extra functions.
joint	Specify if the density gives the joint or common marginal density of the vector of observations. When using a joint density remember that this density will receive its parameters as matrices.
groups	Number of mixtures.
gmu, gshape, gextra	User-specified lists of functions returning the location, shape and extra density parameters for each mixture group and observation as a function of the parameters pgmu, pgshape, pgextra and covariates. These functions should return values in the proper range expected by the density function.
gmixture	A user-specified function of pgmix, giving the regression function of the mixture probabilities. The default is the inverse generalized logit with respect to the first group.
expected	A list of functions returning the expected response value that depends on the combined parameters c(pgmu, pgshape, pgextra, pgmix) for each mixture groups. Defaults to gmu.
pgmu, pgshape, pgextra, pgmix	Vector of initial estimates for the parameters of the location, shape, extra and mixture functions. Parameters always assume real values from (-Inf,Inf).
wt	Vector of subjects sampling weights. Currently the program uses standard sample-weighted $\log(Likelihood)$ assuming fixed weights.
scale.weight	Logical value specifying if the vector of weights wt should be rescaled to sum to the sample size.
check.length	Logical value specifying check of rows length returned by the functions in gextra against the number of variables in y. Especially useful when the density requires more parameters than the number of variables like covariance parameters for multivariate normal.
data	An optional data frame or list containing some or all variables and functions required to fit the model. Due to changes from *CRAN*, moc may have difficulties finding all the objects defined in data. You might prefer attaching it before running moc and its methods.
ndigit, gradtol, steptol, iterlim, print.level, ...	Arguments controlling nlm.

parm	new parameter starting values for update.moc, you can put constraints and fix values with argument what. It should be of the same length as the number of parameters in the moc object.
what	vector of integer values telling *what* to do with the parameters: **0** values correspond to free parameters. **Negative values** are fixed, parameters corresponding to the same negative numbers are fixed to the same values given in parm. **Positive values** are used for free parameters but the same positive values are constrained to be equal.
evaluate	boolean indicating if evaluation of the updated model should be performed (TRUE) or simply return a call (FALSE) for the new model.

Details

The procedure minimizes the resulting $-\log(Likelihood)$ without constraints, the parameters are all assumed to be real numbers. Thus the user should supply appropriate link functions and parameterize the density and parameters functions accordingly (see the examples). By default missing values in the response variables y are assumed to be missing at random, that is the likelihood for the subset of valid observations is just the marginal likelihood for this subset in each mixture. Specific treatment of missing values in the response variables can be achieved by handling them explicitly in the functions density, gmixture, gmu, gshape and gextra. The function density can return NA and yields the default treatment of missing values in the response. The functions gmixture, gmu, gshape and gextra, cannot return NA thus missing values in the covariates should be treated explicitly by these functions.

The lists of functions gmu, gshape, gextra returns the location, shape and extra parameters to the density for each observation and mixture group as a function of pgmu, pgshape and pgextra and covariates. Each function should return a vector of length $nvar$ or a matrix of such vectors (one vector for each subject). The first function in the list is for the first group, the second function for the second group and so on. The functions in the same list share the same parameters but the different lists have different parameters (see the examples).

Setting the attributes *parameters* for functions gmu, gshape, gextra and gmixture will generate parameter labels in the printout of the object.

The residuals, fitted values and posterior probabilities are obtained through the use of the methods residuals, fitted and post.

Value

A list of class moc is returned that contains all of the relevant information calculated, including error code generated by nlm. The printed output includes $-2\log(Likelihood)$, the corresponding df, AIC, BIC, entropy and ICL-BIC (entropy corrected BIC, see AIC.moc), mean mixture probabilities, mean expected and observed values for each mixture group, the maximum likelihood estimates and standard errors.

Note

The expected function is used to compute the fitted values and response residuals (not deviance). It is especially useful when the expected value differs from the location parameters as for censored normal or zero inflated Poisson distributions.

The method of fixed sample-weight provides design-consistent parameters estimates. However, for the moment the program does not provide any methods to include sampling variances resulting from weights estimation. If the user wants to incorporate weights estimation sampling variances it could be achieved, for example, by including `moc` model estimation in a *jackknife* loop.

Be aware that degrees of freedom (df) for mixture models may be useless (if not meaningless) and likelihood-ratio of *apparently* nested models may not converge to a Chi-Square with corresponding df.

Author(s)

Bernard Boulerice <<bernard.boulerice.bb@gmail.com>>

References

McLachlan, G. and Peel, D. (2000) *Finite mixture models*, Wiley-Interscience, New York.

Lindsay, B. G. (1983) *The Geometry of Mixture Likelihoods: A General Theory*, Annals of Statistics, **11**, pp. 86–94.

Biernacki, C., Celeux, G., Govaert, G. (2000) *Assessing a Mixture Model with the Integrated Completed Likelihood*, IEEE Transaction on Pattern Analysis and Machine Learning, **22**, pp. 719–725.

Lindsay, B. G. and Roeder, K. (1992) *Residual diagnostics for mixture models*, Journal of the American Statistical Association, **87**, pp. 785–794.

See Also

`print.moc`, `plot.moc`, `residuals.moc`, `plot.residuals.moc`, `fitted.moc`, `post.moc`, `AIC.moc`, `logLik.moc`, `obsfit.moc`, `nlm`

Examples

```
data(moc.dat)

cnorm.dat<-list()   #This is used as a container for functions and data

# Censored Normal (marginal density)

cnorm<-function(x,mu,sig,min,max)
{mi<-(x == min)*1
ma<-(x == max)*1
mi*pnorm((min-mu)/sig)+ma*(1-pnorm((max-mu)/sig))+
(1-mi-ma)*dnorm((x-mu)/sig)/sig}

# For this data set the range of the dependent variables is [0,14]

cnorm.dat$cnorm1<-function(x,mu,sig,...) {cnorm(x,mu,sig,0,14)}

# We have 4 observations

cnorm.dat$gmu1<- list(
  Group1 = function(pmu) {t(1)%*%rep(pmu[1],4)},
```

```
  Group2 = function(pmu) {t(1)%*%rep(pmu[2],4)},
  Group3 = function(pmu) {t(1)%*%rep(pmu[3],4)})

attr(cnorm.dat$gmu1,"parameters")<-c("  cons1","  cons2","  cons3")

# Expected value of a general censored normal

cmean<-function(mu,sig,min,max) {
max-(max-mu)*pnorm((max-mu)/sig)+(min-mu)*pnorm((min-mu)/sig)-
sig*(dnorm((max-mu)/sig)-dnorm((min-mu)/sig)) }

# Homogeneous variances

cnorm.dat$gshape1<- list(
  Group1 = function(psh) {t(1)%*%rep(exp(psh[1]),4)},
  Group2 = function(psh) {t(1)%*%rep(exp(psh[1]),4)},
  Group3 = function(psh) {t(1)%*%rep(exp(psh[1]),4)})

attr(cnorm.dat$gshape1,"parameters")<-c("  log(std.dev)")

cnorm.dat$cmean1<- list(
  Group1 = function(p) {cmean(cnorm.dat$gmu1[[1]](p[1:3]),cnorm.dat$gshape1[[1]](p[4]),0,14) },
  Group2 = function(p) {cmean(cnorm.dat$gmu1[[2]](p[1:3]),cnorm.dat$gshape1[[2]](p[4]),0,14) },
  Group3 = function(p) {cmean(cnorm.dat$gmu1[[3]](p[1:3]),cnorm.dat$gshape1[[3]](p[4]),0,14) })

moc1<-
moc(moc.dat[,1:4],density=cnorm1,groups=3,gmu=gmu1,gshape=gshape1,
expected=cmean1,pgmu=c(0.5, 2, 5),pgshape=c(0.7),pgmix=c(-0.6, -2.0),
data=cnorm.dat,gradtol=1E-4)

print(moc1)

## Not run:
# Heterogeneous variances across mixture groups

cnorm.dat$gshape2<-list(
  Group1 = function(psh) {t(1)%*%rep(exp(psh[1]),4)},
  Group2 = function(psh) {t(1)%*%rep(exp(psh[2]),4)},
  Group3 = function(psh) {t(1)%*%rep(exp(psh[3]),4)})

cnorm.dat$cmean2<-list(
  Group1 = function(p) {cmean(cnorm.dat$gmu1[[1]](p[1:3]),cnorm.dat$gshape2[[1]](p[4:6]),0,14) },
  Group2 = function(p) {cmean(cnorm.dat$gmu1[[2]](p[1:3]),cnorm.dat$gshape2[[2]](p[4:6]),0,14) },
  Group3 = function(p) {cmean(cnorm.dat$gmu1[[3]](p[1:3]),cnorm.dat$gshape2[[3]](p[4:6]),0,14) })

moc2<-
moc(moc.dat[,1:4],density=cnorm1,groups=3,gmu=gmu1,gshape=gshape2,
expected=cmean2,pgmu=moc1$coef[1:3],pgshape=c(rep(moc1$coef[4],3)),
pgmix=moc1$coef[5:6],data=cnorm.dat,gradtol=1E-4)


## End(Not run)
# Heterogeneous variances across time
```

```
cnorm.dat$gshape3<-list(
  Group1 = function(psh) {exp(t(1)%*%psh[1:4])},
  Group2 = function(psh) {exp(t(1)%*%psh[1:4])},
  Group3 = function(psh) {exp(t(1)%*%psh[1:4])})

cnorm.dat$cmean3<-list(
  Group1 = function(p) {cmean(cnorm.dat$gmu1[[1]](p[1:3]),cnorm.dat$gshape3[[1]](p[4:7]),0,14)},
  Group2 = function(p) {cmean(cnorm.dat$gmu1[[2]](p[1:3]),cnorm.dat$gshape3[[2]](p[4:7]),0,14)},
  Group3 = function(p) {cmean(cnorm.dat$gmu1[[3]](p[1:3]),cnorm.dat$gshape3[[3]](p[4:7]),0,14)})

moc3<-
moc(moc.dat[,1:4],density=cnorm1,groups=3,gmu=gmu1,gshape=gshape3,
expected=cmean3,pgmu=moc1$coef[1:3],pgshape=c(rep(moc1$coef[4],4)),
pgmix=moc1$coef[5:6],data=cnorm.dat,gradtol=1E-4)

print(moc3)

cnorm.dat$ages<-cbind(1.7,3,4.2,5.6)

## Not run:
# Last group is a linear function of time

cnorm.dat$gmu1t<-list(
  Group1 = function(pmu) {pmu[1]*cnorm.dat$ages^0},
  Group2 = function(pmu) {pmu[2]+pmu[3]*cnorm.dat$ages},
  Group3 = function(pmu) {pmu[4]*cnorm.dat$ages^0})

cnorm.dat$cmean1t<-list(
  Group1 = function(p) {cmean(cnorm.dat$gmu1t[[1]](p[1:4]),cnorm.dat$gshape1[[1]](p[5]),0,14)},
  Group2 = function(p) {cmean(cnorm.dat$gmu1t[[2]](p[1:4]),cnorm.dat$gshape1[[2]](p[5]),0,14)},
  Group3 = function(p) {cmean(cnorm.dat$gmu1t[[3]](p[1:4]),cnorm.dat$gshape1[[3]](p[5]),0,14)})

moc4<-
moc(moc.dat[,1:4],density=cnorm1,groups=3,gmu=gmu1t,gshape=gshape1,
expected=cmean1t,pgmu=append(moc1$coef[1:3],0,after=2),
pgshape=c(moc1$coef[4]),pgmix=moc1$coef[5:6],data=cnorm.dat,gradtol=1E-4)


# Zero inflated Poisson log-linear in time for the third group
# Be careful dpois requires integer values

zip<- function(x,la,shape=1,extra)
{ mix<- exp(extra)/(1+exp(extra))
  mix*(x == 0)+(1-mix)*dpois(x,la) }


## End(Not run)


gmup<-list(
  Group1 = function(pmu) {exp(pmu[1]*cnorm.dat$ages^0)},
  Group2 = function(pmu) {exp(pmu[2]+pmu[3]*cnorm.dat$ages)},
```

```
  Group3 = function(pmu) {exp(pmu[4]*cnorm.dat$ages^0)})

## Not run:
zipfit<-list(
  Group1 = function(p) { gmup[[1]](p)/(1+exp(p[5]))},
  Group2 = function(p) { gmup[[2]](p)/(1+exp(p[5]))},
  Group3 = function(p) { gmup[[3]](p)/(1+exp(p[5]))})

gextrap<-list(
  Group1 = function(pxt) {t(1)%*%rep(pxt[1],4)},
  Group2 = function(pxt) {t(1)%*%rep(pxt[1],4)},
  Group3 = function(pxt) {t(1)%*%rep(pxt[1],4)})

moc5<-
moc(moc.dat[,1:4],density=zip,groups=3,gmu=gmup,gextra=gextrap,
expected = zipfit,pgmu=c(-0.6, 0.64,0, 1.6),pgextra=c(-3),
pgmix=c(-0.7, -2), gradtol=1E-4)


## End(Not run)

# Standard Poisson with mixture depending on time independent
# dichotomous covariate
# Be aware that dpoiss require integer values

dumm<-moc.dat[,5]-1
gmixt<-function(pm){
mix<-cbind(1,dumm)%*%matrix(pm[1:4],2,2)
cbind(1,exp(mix))/(1+apply(exp(mix),1,sum))}

poiss<-function(x,la,...) {dpois(x,la)}

moc6<-
moc(moc.dat[,1:4],density=poiss,groups=3,gmu=gmup,gmixture=gmixt,
pgmu=c(-0.7,2.0, 0, 1.5),pgmix=c(-0.2,-1, -1 ,-2),gradtol=1E-4)

print(moc6)

obsfit.moc(moc6,along=dumm)

entropy(moc1,moc3,moc6)

## Not run:
plot(moc6,against=cnorm.dat$ages,main="MOC profiles",xlab="age",ylab="Y")
plot(residuals(moc6))


## End(Not run)

#More extended examples are available in the Examples directory of the package.
```

moc.dat *Example data for MOC*

Description

Simulated data from a mixture model.

Usage

```
data(moc.dat)
```

Format

This data frame contains 500 rows (subjects) with the following columns (variables) :

Y1, Y2, Y3, Y4 simulated observation at 4 occasions.

X1 a dichotomous covariate.

Author(s)

Bernard Boulerice <<bernard.boulerice.bb@gmail.com>>

plot.moc *Plotting methods for MOC models.*

Description

`plot.moc` plots the fitted and observed values of a fitted `moc` model.

`profilesplot` is a generic method to plot subject profiles of fitted model.

`profilesplot.moc` plots the subject response profiles (variables or posterior) of each subject of a fitted `moc` object with colors that are a mix of group colors (see `mix.colors.moc`).

`plot.residuals.moc` nicely plots and object of type `residuals.moc`.

`entropyplot` is a generic method to plot subject entropy.

`entropyplot.moc` plots the prior and posterior subject entropy (standardized or not) corresponding to a `moc` model. The subject entropy are first sorted by prior then by posterior within prior.

Usage

```
## S3 method for class 'moc'
plot(x, against = 1:x$nvar,main = "",
     xlab = "", ylab = "", prob.legend = TRUE, scale = FALSE,
     group.colors = rainbow(x$groups), ...)

## S3 method for class 'residuals.moc'
plot(x, against = "Index", groups = 1:dim(x)[3],
     sunflower = FALSE, group.colors = NULL, ...)

profilesplot(x, ...)

## S3 method for class 'moc'
profilesplot(x, against = 1:x$nvar, main = NULL,
             xlab = "", ylab = "", col.legend = TRUE,
             scale = FALSE, group.colors = rainbow(x$groups),
             type = "subject", ...)

entropyplot(x, ...)

## S3 method for class 'moc'
entropyplot(x, main = NULL, std = TRUE, lwd = 1.5,
            col = c("red3","green3","gray95"),
            shade.gr.col = gray(1-0.5*(0:(x$groups-1))/(x$groups-1)),
            legend = TRUE, ...)
```

Arguments

`x`	Objects of class `moc` or `residuals.moc`.
`against`	x axis for plotting the profiles. A variable against which to plot the residuals or the strings **Index:** The default, use the index of the residuals array. **Observation:** Use the column (variable) index of the response matrix. **Subject:** Use the row (subject) index of the response matrix.
`main, xlab, ylab, ...`	Arguments to be passed to `plot`, `matplot`.
`prob.legend, col.legend, legend`	Add mixture probabilities, color legend and lines to the plot.
`sunflower`	Specify if a sunflower or standard plot is requested.
`scale`	Specify if each variable should be scaled.(see `scale`)
`groups`	Specify for which groups `residuals.moc` plot is requested.
`type`	A string giving the type of profiles to plot **subject:** (default) for subject profiles. **variable:** for pairs plot of response variables.

	posterior: for pairs of posterior probabilities.
`std`	If TRUE standardized entropy are plotted.
`lwd, col`	`entropyplot.moc` will plot the prior and posterior entropy with lines of width lwd and colors `col[1:2]`, the third color is used for the area between the two curves.
`shade.gr.col`	`entropyplot.moc` will mix these colors with `mix.colors.moc` and paint the area between the two curves accordingly.
`group.colors`	The groups base colors.

Value

`plot.moc` invisibly returns a `list` containing the plotted values and scaling information.

`plot.residuals.moc` invisibly returns the plotted residual values.

Author(s)

Bernard Boulerice <<bernard.boulerice.bb@gmail.com>>

See Also

`moc`, `residuals.moc`, `print.moc`, `AIC.moc`

`print.moc` *Summary methods for fitted MOC models*

Description

`print.moc` prints information contained in a fitted `moc` object. The `attributes` *parameters* of the functions `gmu`, `gshape`, `gextra` and `gmixture` will be used to label the output.

`coef.moc` returns the coefficients (estimated parameters) of a fitted `moc` object.

`fitted.moc` computes the expected values for each observation of a `moc` object using its `expected` function.

`obsfit.moc` computes and prints the mean posterior probabilities and the posterior means of a user specified function of the expected and observed values, separated with respect to the specified variable.

Usage

```
## S3 method for class 'moc'
print(x, digits = 5, expand = TRUE, transpose = FALSE, ...)

## S3 method for class 'moc'
coef(object, split=FALSE, ...)
```

```
## S3 method for class 'moc'
fitted(object, ...)

obsfit.moc(object, along = list(cons = rep(1, object$nsubject)),
           FUN = function(x) x)
```

Arguments

x, object	Objects of class moc.
split	If split is TRUE, returns a list with elements corresponding to mu, shape, extra and mixture parameters.
digits	Number of digits to be printed.
expand	Expand density, gmu, gshape, gextra, gmixture function body in the print.
transpose	Transpose fitted.mean and observed.mean in the print.
along	Splitting variable.
FUN	User defined function to apply to observed and expected values.
...	Unused.

Details

obsfit.moc will first compute the posterior probabilities for all subjects in each mixture using post.moc and then the weighted posterior mean probabilities

$$\hat{\bar{\tau}}_k = \frac{\sum_i wt_i \, \hat{\tau}_{i,k}}{\sum_i wt_i}$$

The weighted posterior means of a function $g()$ of the data (which are the empirical estimators of the conditional expectation given mixture group) are computed as

$$\frac{\sum_i wt_i \, \hat{\tau}_{i,k} \, g(y_i)}{\sum_i wt_i \, \hat{\tau}_{i,k}}$$

where both sums are taken over index of valid data y_i.

Value

All these methods return their results invisibly.

Author(s)

Bernard Boulerice <<bernard.boulerice.bb@gmail.com>>

See Also

moc, residuals.moc, post.moc, plot.moc, AIC.moc

`residuals.moc` *Posterior and residuals methods for MOC models*

Description

`post` is a generic method for computing posterior probabilities of a fitted model.

`post.moc` computes the posterior mixture probabilities for each subject of a fitted `moc` model.

`residuals.moc` computes response, deviance, gradient and mixture residuals. The residuals are optionally weighted by the posterior mixture probabilities, globally (with `post`) or within each group (in that case `post` is divided by its mean for each group).

`npmle.gradient` computes the components and average of the mixture gradient function at some specified parameters values

$$\frac{1}{n}\sum_{i=1}^{n}\Big(\frac{\hat{h}_j(y_i|\,x_i)}{\sum_k \hat{\pi}_k(z_i,x_i)\,\hat{h}_k(y_i|\,x_i)}-1\Big)$$

or the empirical mixture gradient which is

$$\frac{1}{n}\sum_{i=1}^{n}\Big(\frac{d\hat{F}(y_i)}{\sum_k \hat{\pi}_k(z_i,x_i)\,\hat{h}_k(y_i|\,x_i)}-1\Big)$$

where $d\hat{F}()$ is the empirical measure.

Usage

```
post(object, ...)

## S3 method for class 'moc'
post(object, ...)

## S3 method for class 'moc'
residuals(object, ...,
          type = c("deviance","response","mixture","gradient"),
          post.weight = TRUE, within = FALSE)

npmle.gradient(object, parm = object$coef, gradient = TRUE,
               average = FALSE)
```

Arguments

`object`	Object of class `moc`.
`type`	Type of residuals: either "deviance" (the default), "response", "gradient" and "mixture" (see description and details).
`post.weight`	Specify if the residuals must be weighted by the posterior mixture probabilities. Weighting is preferable, it is the default.

within	Boolean that specify if the posterior weights are rescaled within each mixture group.
parm	Parameters values at which evaluation of the gradient takes place.
gradient	Boolean specifying if you require the density for each mixture group (*TRUE*) or the empirical measure (*FALSE*) in the numerator of npmle.gradient (see the description above).
average	Boolean that specify if npmle.gradient should return the individual components or the average.
...	Unused.

Details

Posterior probabilities are the conditional probabilities of mixture groups given the subject response observations and are computed using the formula:

$$\hat{\tau}_{i,k} = \frac{\hat{\pi}_k(z_i, x_i)\, \hat{h}_k(y_i|x_i)}{\sum_k \hat{\pi}_k(z_i, x_i)\, \hat{h}_k(y_i|x_i)}$$

Response residuals are simply the difference between the observed and expected values,

$$response = y - expected$$

Deviance residuals are defined as properly scaled difference in the log likelihood at the observed and fitted value.

$$deviance = \left\{ 2 \cdot wt \cdot \log\left(\frac{\text{density}(y, y, shape, extra)}{\text{density}(y, mu, shape, extra)} \right) \right\}^{1/2} \cdot \text{sign}(response)$$

The npmle.gradient function is primarily intended to compute the components which are used to define the mixture and gradient residuals in residuals.moc. Mixture residuals uses the empirical gradient components while gradient residuals uses the mixture gradient components defined in the description above. The average of the first form above which we call the gradient can be used to check some minimal requirement about a proposed solution given with parm: if it is a maximum likelihood (not necessarily a non-parametric maximum likelihood) the average should be 0. See Lindsay, B.G. (1983) for details about the use of the gradient function for finite mixture and non-parametric mixture and Lindsay, B.G. and Roeder, K. (1992) for residuals diagnostics specific to mixture models.

Globally weighted residuals are preferable to detect influential data, wrong number of groups and differences between groups. Rescaled weight residuals are more useful when plotted against some variables or variable index to detect misspecified regression function or density.

Value

residuals.moc returns an array of class residuals.moc and residuals with attributes *type*, *post.weight* and *within*. All these methods return their values invisibly.

Author(s)

Bernard Boulerice <<bernard.boulerice.bb@gmail.com>>

References

McLachlan, G. and Peel, D. (2000) *Finite mixture models*, Wiley-Interscience, New York.

Lindsay, B. G. and Roeder, K. (1992) *Residual diagnostics for mixture models*, Journal of the American Statistical Association, **87**, pp. 785–794.

See Also

`moc`, `plot.moc`, `print.moc`, `AIC.moc`, `plot.residuals.moc`

`utils.moc` *MOC utility functions.*

Description

Functions to compute generalized logit and inverse logit with respect to a reference group.

`mix.colors.moc` computes subject colors useful for plotting by mixing group base colors according to the subject posterior probabilities.

`mocUtils` lists or returns an environment (which can be attached) containing the utility functions from *Utils* subdirectory of *moc* package.

Usage

```
inv.glogit(gmix, ref = 1)
glogit(p, ref = 1)

mix.colors.moc(object, group.colors = rainbow(object$groups))

mocUtils(filename)
```

Arguments

`object`	A fitted moc object.
`p`	Vector of probabilities to transform in generalized logit (log-odds) with ref as a reference group.
`gmix`	Vector of generalized logit with respect to ref to transform in probabilities.
`ref`	Reference group.
`group.colors`	The groups base colors to be mixed in proportion corresponding to posterior probabilities.
`filename`	The filename containing the source of utility functions or empty for a list of available source codes.

Details

These functions do not perform any check on their arguments.

Value

A vector of probabilities or generalized logit. `mix.colors.moc` invisibly returns a vector of subject mixed colors in hexadecimal *RGB* format.

Note

More utility functions are available in the *Utils* directory of the `moc` distribution.

Author(s)

Bernard Boulerice <<bernard.boulerice.bb@gmail.com>>

Index