R documentation

of ‘-o=depmix.dvi’ etc.

April 28, 2005

R topics documented:

depmix 1

Index **7**

---

depmix *Fitting Dependent Mixture Models*

Description

fitdmm `fitdmm` fits mixtures of hidden/latent Markov models on arbitrary length time series fitdmm mixed categorical and continuous data. This includes latent class models (for time series of length 1).

`posterior` `posterior` computes the most likely latent state sequence for a given dataset and model.

Usage

```
fitdmm(dat, dmm, printlevel = 1, post = TRUE, tdcov = 0, ses
         = TRUE, method = "nlm", der = 1, vfactor = 15,
         iterlim = 300, accuracy = "standard", kmst = !dmm$st,
         postst = TRUE)
loglike(dat, dmm, tdcov = 0, grad = FALSE, hess = FALSE, set
         = TRUE, grInd = 0, sca = 1, printlevel = 1)
posterior(dat,dmm,tdcov=0,printlevel=1)
bootstrap(object,dat,samples=100, pvalonly=0,...)
summary.fit(object,precision=3,fd=1,...)
oneliner(object,precision=3)
```

Arguments

`dat`	An object (or list of objects) of class `md`, see markovdata. If dat is a list of objects of class `md` a multigroup model is fitted on these data sets.
`dmm`	An object (or a list of objects) of class `dmm`, see dmm. If dmm is a list of objects of class `dmm`, these are taken to components of a mixture of dmm's model and will be coerced to class `mixdmm`. In any case, the model that is fitted a multigroup mixture of dmm's with default ngroups=1 and number of components=1.
`printlevel`	`printlevel` controls the output provided by the C-routines that are called to optimize the parameters. The default of 1 provides minmal output: just the initial and final loglikelihood of the model. Setting higher values will provide more output on the progress the iterations.
`post`	By default posteriors are computed, the result of which can be found in fit$post.
`method`	This is the optimization algorithm that is used. NLM is the default method. There is further support for optim and NPSOL.
`der`	Specifies whether derivatives are to be used in optimization.
`vfactor`	vfactor controls optimization in optim and nlm. Since in those routines there is no possibility for enforcing constraints, constraints are enforced by adding a penalty term to the loglikelihood. The penalty term is printed at the end of optimization if it is not close enough to zero. This may have several reasons. When parameters are estimated at bounds for example. This can be solved by fixing those parameters on their boundary values. When this is not acceptable vfactor may be increased such that the penalty is larger and the probability that they actually hold in the fitted model is correspondingly higher.
`tdcov`	Logical, when set to TRUE, given that the model and data have covariates, the corresponding parameters will be estimated.
`ses`	Logical, determines whether standard errors are computed after optimization.
`iterlim`	The iteration limit for npsol, defaults to 100, which may be too low for large models.
`accuracy`	This argument can be used to set accuracy of optimization when using `nlm` as optimizer. It can take values "standard" (the default), "high" and "best" for increasing levels of accuracy.
`grad`	logical; if TRUE the gradients are returned.
`hess`	logical; if TRUE the hessian is returned.
`set`	Whith the default value TRUE, the data and models parameters are sent to the C/C++ routines before computing the loglikelihood. When set is FALSE, this is not done. If an incorrect model was set earlier in the C-routines this may cause serious errors and/or crashes.
`sca`	If set to -1.0 the negative loglikelihood, gradients and hessian are returned.
`object`	An object of class `fit`, ie the return value of fitdmm.
`kmst,postst`	These arguments control the generation of starting values by kmeans and posterior estimates respectively.
`grInd`	Logical argument; if TRUE, individual contributions of each independent realization to the gradient vector will be returned.
`fd`	Print the finite difference based standard errors in the summary if both those and bootstrapped standard errors are available.

`samples`	The number of samples to be used in bootstrapping.
`pvalonly`	Logical, if 1 only a bootstrapped pvalue is returned and not fitted paramaters to compute standard errors, optimization is truncated when the loglikelihood is better than the original loglikelihood.
`precision`	Precision sets the number of digits to be printed in the summary functions.
`...`	Used in summary.

Details

The function `fitdmm` optimizes the parameters of a mixture of `dmms` using a general purpose optimization routine subject to linear and nonlinear constraints on the parameters.

Value

`fitdmm` returns an object of class `fit` which has a summary method that prints the summary of the fitted model, and the following fields:

`date,timeUsed,totMem`	The date that the model was fitted, the time it took to so and the memory usage.
`loglike`	The loglikelihood of the fitted model.
`aic`	The AIC of the fitted model.
`bic`	The BIC of the fitted model.
`mod`	The fitted model.
`post`	See function posterior for details.
	`loglike` returns a list of the following:
logl	The loglikelihood.
gr,grset	`gr` contains the gradients. `grset` is a logical vector giving information as to which gradients are set, currently all gradients are set except the gradients for the mixing proportions.
hs,hsset	`hs` contains the hessian. `hsset` is a logical giving information as to which elements are computed.
	`posterior` returns lists of the following:
states	A matrix of dimension 2+sum(nstates) by sum(length(ntimes)) containing in the first column the a posteriori component, in the second column the a posteriori state and in the remaining column the posterior probabilities of all states.
comp	Contains the posterior component number for each independent realization; all ones for a single component model.
	`bootstrap` returns an object of class `fit` with three extra fields, the bootstrapped standard errors, bse, a matrix with goodness-of-fit measures of the bootstrap samples, ie logl, AIC and BIC and pbetter, which is the proportion of bootstrap samples that resulted in better fits than the original model.
	`summary.fit` pretty-prints the above fields.
	`oneliner` returns a vector of loglike, aic, bic, mod$npars, mod$freepars, date.

Note

The `repeated` library by Jim Lindsey fits hidden markov models. `fitdmm` fits time series of arbitrary length and mixtures of `dmms`, where, to the best of my knowledge, other packages are limited due to the different optimization routines that are commonly used for these types of models (this is certainly so for categorical data models).

Author(s)

Ingmar Visser

References

Articles

See Also

dmm,markovdata,repeated

Examples

```
# COMBINED RT AND CORRECT/INCORRECT SCORES from a 'switching' experiment

data(rudy)
mod <- dmm(nsta=2,itemt=c(1,2)) # gaussian and binary items
fit1 <- fitdmm(dat=rudy,dmm=mod)
summary(fit1)

# add some constraints using conpat
conpat=rep(1,15)
conpat[1]=0
conpat[14:15]=0
conpat[8:9]=0
# use starting values from the previous model fit, except for the guessing
# parameters which should really be 0.5
stv=c(1,.896,.104,.084,.916,5.52,.20,.5,.5,6.39,.24,.098,.90,0,1)
mod=dmm(nstates=2,itemt=c("n",2),stval=stv,conpat=conpat)

fit2 <- fitdmm(dat=rudy,dmm=mod)
summary(fit2)

# add covariates to the model to incorporate the fact the accuracy pay off changes per tr
# 2-state model with covariates + other constraints
conpat=rep(1,15)
conpat[1]=0
conpat[8:9]=0
conpat[14:15]=0
conpat[2]=2
conpat[5]=2
stv=c(1,0.9,0.1,0.1,0.9,5.5,0.2,0.5,0.5,6.4,0.25,0.9,0.1,0,1)
tdfix=rep(0,15)
tdfix[2:5]=1
stcov=rep(0,15)
stcov[2:5]=c(-0.4,0.4,0.15,-0.15)

mod<-dmm(nstates=2,itemt=c("n",2),stval=stv,conpat=conpat,tdfix=tdfix,tdst=stcov,modname=

fit3 <- fitdmm(dat=rudy,dmm=mod,tdcov=1,der=0,ses=0,vfa=80,accu="best")
summary(fit3)

# split the data into three time series
data(rudy)
r1=markovdata(dat=rudy[1:168,],item=itemtypes(rudy))
```

```
r2=markovdata(dat=rudy[169:302,],item=itemtypes(rudy))
r3=markovdata(dat=rudy[303:439,],item=itemtypes(rudy))

# define 2-state model with constraints
conpat=rep(1,15)
conpat[1]=0
conpat[8:9]=0
conpat[14:15]=0
stv=c(1,0.9,0.1,0.1,0.9,5.5,0.2,0.5,0.5,6.4,0.25,0.9,0.1,0.5,0.5)
mod<-dmm(nstates=2,itemt=c("n",2),stval=stv,conpat=conpat)

# define 3-group model with equal transition parameters, and no
# equalities between the obser parameters
mgr <-mgdmm(dmm=mod,ng=3,trans=TRUE,obser=FALSE)

fitmg <- fitdmm(dat=list(r1,r2,r3),dmm=mgr)
summary(fitmg)

# LEARNING DATA AND MODELS (with absorbing states)

data(mdslow)
summary(mdslow)

# all or none model with error prob in the learned state
fixed = c(0,0,0,1,1,1,1,0,0,0,0)
stv = c(1,1,0,0.07,0.93,0.9,0.1,0.5,0.5,0,1)
allor <- dmm(nstates=2,itemtypes=2,fixed=fixed,stval=stv,modname="All-or-none")
fit4 <- fitdmm(dat=mdslow,dmm=allor)
summary(fit4)

# Concept identification model: learning only after an error

data(mdrat)
summary(mdrat)

# Concept identification model: learning only after an error
st=c(1,1,0,0,0,0.5,0.5,0.5,0.25,0.25,0.8,0.2,1,0,0,1,0.25,0.375,0.375)
# fix some parameters
fx=rep(0,19)
fx[8:12]=1
fx[17:19]=1
# add a couple of constraints
conr1 <- rep(0,19)
conr1[9]=1
conr1[10]=-1
conr2 <- rep(0,19)
conr2[18]=1
conr2[19]=-1
conr3 <- rep(0,19)
conr3[8]=1
conr3[17]=-2
conr=c(conr1,conr2,conr3)
cim <- dmm(nstates=3,itemtypes=2,fixed=fx,conrows=conr,stval=st,modname="CIM")

fit5 <- fitdmm(dat=mdrat,dmm=cim)
summary(fit5)
```

```
# define a mixture of the above models ...
mix <- mixdmm(dmm=list(allor,cim),modname="MixAllCim")

data(mdall)

# ... and fit it on the combined data mdall
fit6 <- fitdmm(dat=mdall,dmm=mix)
summary(fit6)
```

Index

∗Topic **models**
    depmix, 1

bootstrap *(depmix)*, 1

depmix, 1
dmm, *4*

fitdmm *(depmix)*, 1

loglike *(depmix)*, 1

markovdata, *4*

oneliner *(depmix)*, 1

posterior *(depmix)*, 1

repeated, *3*, *4*

summary.fit *(depmix)*, 1