Package 'informR'

March 9, 2015

Version 1.0-5

Date 2015-03-09

Title Sequence Statistics for Relational Event Models

Author Christopher Steven Marcum <`cmarcum@uci.edu`>

Maintainer Christopher Steven Marcum <`cmarcum@uci.edu`>

Description Aids in creating sequence statistics for Butts's 'relevent' software.

License GPL (>= 2)

Depends R (>= 2.12), abind, relevent

LazyLoad yes

NeedsCompilation no

Repository CRAN

Date/Publication 2015-03-09 20:38:36

R topics documented:

`informR-package` *R Tools for Creating Sequence Statistics*

Description

Tools for creating sequence statistics for Butts's egocentric relational event model fitting software in the `library(relevent)` R package.

Details

Package:	informR
Type:	Package
Version:	1.0
Date:	2011-04-17
License:	GPL 2.0 or greater
LazyLoad:	yes

Use this package to create variables and assemble statslists and eventlists from regular expressions for egocentric relational event model fitting using Butts's `library(relevent)` package.

Author(s)

Author and Maintainer:Christopher Steven Marcum <cmarcum@uci.edu>

See Also

rem

Examples

```
rawevents<-sample(rep(c("ran","eat","stay","eat","ran","play"),50))
actors<-rep(c("Jim","Bill","Pete"),100)
evmat<-cbind(rawevents,actors)
eventlist<-gen.evl(evmat)

#See event-type alphabetic token codes
eventlist$event.key

alpha.ints<-gen.intercepts(eventlist,basecat="ran")

#Create some inertial s-form statistics and fit the models
beta.sforms<-gen.sformlist(eventlist,c("aa","bb","cc","dd"))
```

```
#Combine s-form matrices with intercepts
beta.ints<-slbind(beta.sforms,alpha.ints,new.names=TRUE,event.key=eventlist$event.key)
fitalpha.rem<-rem(eventlist=eventlist$eventlist,statslist=alpha.ints,estimator="BPM")
fitbeta.rem<-rem(eventlist=eventlist$eventlist,statslist=beta.ints,estimator="BPM")
summary(fitalpha.rem)
summary(fitbeta.rem)

for(i in c("aa","bb","cc","dd")) count.sform(eventlist,i)
```

`atus80int` *Interval Events Subset of the American Time Use Survey*

Description

Event histories from respondents over the age of 80 in the pooled 2003–2008 American Time Use Survey.

Usage

```
data(atus80int)
```

Format

A data frame with 124,704 observations on the following 4 variables:

`Events` Type of event. See details.

`Time` Cumulative time that an event occurred. See details.

`TUCASEID` Respondent unique identification number.

`SEX` Sex of respondent. 1=Males.

Details

Each general sequential activity spell in the American Time Use Survey for respondents aged 80 and over was split into “starting” and “stopping” events.

The timing variable marks the instantaneous “time” of event occurrence in minutes beginning at time 0 and ending at time t (which is usually 1400). Because simultaneous events are not allowed one thousandth of a minute is added to the starting time of each activity spell after the first spell.

Source

Bureau of Labor Statistics. The American Time Use Survey. Available online at: http://www.bls.gov/tus/.

Examples

```
data(atus80int)
#Types of Events
unique(atus80int$Events)
```

`atus80ord` *Ordinal Events Subset of the American Time Use Survey*

Description

Event histories from respondents over the age of 80 in the pooled 2003–2008 American Time Use Survey.

Usage

```
data(atus80ord)
```

Format

A data frame with 62,352 observations on the following 3 variables.

`Activities` Type of activity spell. See details.

`TUCASEID` Respondent unique identification number.

`SEX` Sex of respondent. 1=Males.

Details

Each activity was recoded into a general activity class with 14 possible values (including “missing” as NA).

Source

Bureau of Labor Statistics. The American Time Use Survey. Available online at: `http://www.bls.gov/tus/`.

Examples

```
data(atus80ord)
#Activity Spell Frequencies by Sex
table(atus80ord$Activities,atus80ord$SEX)
```

`count.sform` *Count Sform Instances*

Description

Counts and prints instances of an sform in a statslist.

Usage

```
count.sform(evls, sform, new.name)
```

Arguments

`evls`	An eventlist with eventlist attribute "char", as well as element "event.key".
`sform`	An sform regular expression using the `evls$event.key[,1]` uids.
`new.name`	Logical. Should count.sform guess at a descriptive name for the sform?. Default is TRUE.

Value

A list containing:

`list`	A list of length `length(evls$eventlist)`, each numeric element of which is a count of the instances of the sform in the respective eventlist.

Author(s)

Christopher Steven Marcum

Examples

```
rawevents<-sample(rep(c("ran","eat","stay","eat","ran","play"),100))
actors<-rep(c("Jim","Bill","Pete"),10)
eventlist<-cbind(rawevents,actors)
evls<-gen.evl(eventlist)

sform<-"ab"
count.sform(evls,sform)
count.sform(evls,sform)$Bill

#Compare with
count.sform(evls,sform,new.name=FALSE)
```

gen.evl *Generate numeric categories of events.*

Description

Generate numeric categories of events in an idXevent list.

Usage

```
gen.evl(eventlist, null.events=NULL)
```

Arguments

`eventlist`	Either a two (2) or three (3) column matrix or data.frame. See Details.
`null.events`	A character vector of event values that should be treated as 0 in the eventlist. Default is NULL.

Details

For interval (or continuous) relational event models, `eventlist` must be a three column matrix where the first column indexes the events, the second column indexes the temporal information of when the event transpired, and the third column indexes an event history grouping factor (possibly, for instance, the name(s) of the actor(s)). For each unique value of the grouping factor, the corresponding events must be given in ascending temporal order. For ordinal (or discrete) relational event models, `eventlist` must be a two column matrix with the first column indexing the events and the second column indexing the event history grouping factor.

Value

A list containing:

`eventlist`	Vector of integers representing numeric codes for events. Has `attr(,"char")`, which represents the character representation of those values.
`event.key`	A key identifying the unique ids of each verbose event type.
`null.events`	`if(!is.null(null.events))`, a vector of null events.

Author(s)

Christopher Steven Marcum

Examples

```
rawevents<-sample(rep(c("ran","eat","stay","eat","ran","play"),100))
actors<-rep(c("Jim","Bill","Pete"),10)
eventlist<-cbind(rawevents,actors)
evls<-gen.evl(eventlist)
evls$eventlist$Bill
evls$event.key
```

```
#Compare with:
evls<-gen.evl(eventlist,null.events=c("ran","eat"))
evls$eventlist$Bill
evls$event.key
evls$null.events
```

gen.id *Generate Unique Id's for Events*

Description

Internal function for generating unique single character id's for each unique event in an eventlist. May be useful for a user to call if, for some reason, event.key is lost from the eventlist.

Usage

```
gen.id(rawevents, print = TRUE)
```

Arguments

`rawevents`	a character string of events
`print`	logical, print the single character codes corresponding to the unique event values to stdout. Defaults to TRUE.

Details

Currently, the rawevents string must not contain more than 52 unique event types (i.e., 2Xlength(a:z)

Value

A character string of single character identifiers for each unique event.

Author(s)

Christopher Steven Marcum

Examples

```
rawevents<-sample(rep(c("ran","eat","stay","eat","ran","play"),5))
actors<-rep(c("Jim","Bill","Pete"),10)
idevents<-gen.id(rawevents)
idevents
```

gen.intercepts *Generate Baserate Statistics*

Description

Generates baserate statistics in statslist form for direct input into `rem()` in Butts's relevent R package.

Usage

```
gen.intercepts(evl, basecat=NULL, type=1, contr=TRUE)
```

Arguments

`evl`	An eventlist. Possibly passed from `gen.evl()`
`basecat`	A string indicating which event to treat as the baseline group. Default is NULL, which uses the first event in `evl$event.key`.
`type`	An integer indicating the location of the statistics in the statslist. 1 for global, 2 for local. Default is global.
`contr`	Logical. Should the routine use SAS-like contrasts to generate the statistics? Defaults to true (which is faster than the alternative).

Value

A list containing:

`list`	An i,j,k array consisting of the i'th event's j by k identity contrast matrix.

Author(s)

Christopher Steven Marcum

See Also

`slbind`,`gen.id`

Examples

```
rawevents<-sample(rep(c("ran","eat","stay","eat","ran","play"),5))
actors<-rep(c("Jim","Bill","Pete"),10)
eventlist<-gen.evl(cbind(rawevents,actors))
baserates<-gen.intercepts(eventlist)
baserates[[1]][[1]][1,,]

#Compare with:
baserates<-gen.intercepts(eventlist,contr=FALSE)
baserates[[1]][[1]][1,,]
```

`gen.sform` *Generates Sform Statistics from Event Sequences*

Description

Internal function for generating sequence statistics based on limited regular expressions.

Usage

```
gen.sform(a, sform, olev = NULL)
```

Arguments

`a`	A character vector of events, possibly passed from `gen.id`. Must have `attr(a,"a.uid")`.
`sform`	A regular expression representing the sequence statistic, see details
`olev`	a character vector, see details.

Details

This function is typically not called by the end user. Regular expressions must adhere to PERL standards and, at this time, must make use of the alphabetic event.key values in an eventlist object.

The parameter `olev` is a placeholder for future functionality, however can currently be used to truncate valid event types not found in the null.events values in an eventlist object. This parameter will rarely ever be non-null.

Value

A list of `length(a)` containing single row matrices of sufficient statistics for matches/non-matches of the S-form.

Note

Currently, `a` must consist of single character ids. This is not checked.

Author(s)

Christopher Steven Marcum

gen.sformlist *Generate a list of S-form sequence statistics.*

Description

Generates a list of S-form sequence statistics based on sform regular expressions. Output is not used directly but is passed to other methods.

Usage

```
gen.sformlist(evl, sforms,cond=FALSE, interval=FALSE, warn=TRUE,...)
```

Arguments

`evl`	An eventlist passed from `gen.evl()`
`sforms`	A character string of S-form regular expressions representing the sequence statistics.
`cond`	Should the resulting statistics condition out the prefix? Defaults to FALSE.
`interval`	When cond=TRUE, are elements of sforms paired as start-stop terms?. Defaults to FALSE.
`warn`	Should warnings be issued? Defaults to TRUE.
`...`	Additional arguments to pass to `gen.sform`

Details

Regular expressions must adhere to PERL standards and, at this time, must make use of the alphabetic `evl$event.key` values in an eventlist object. Two special regex operators are permitted in S-form expressions: the "|" (OR operator) and the "+" (AND operator). In S-form expressions, the OR operator is used to differentiate between two possible event paths and the AND operator is used to indicate persistence of the last event.

Useful expressions include:

[,1]	aa	inertial term: S-form of the type "event a predicts event a"
[,2]	ab	basic digram transition term: S-form of the type "event a predicts event b"
[,3]	a+b	transition term with persistence: S-form of the type "some series of events a predicts event b"
[,4]	abc	basic trigram term: S-form of the of the type "event a followed by event b predicts event c"
[,5]	aab	tuples term: S-form of the of the type "event a followed by event a predicts event c"
[,6]	(a|b)c	complex term with divergence: S-form of the of the type "event a OR event b predicts event c"
[,7]	(a|b+d)c	complex term with divergence and persistence: S-form of the of the type "event a predicts event c OR some

There may be more than one way to form an S-form expression. For example, aa is equivalent to a+a; however, the former is preferred because the search methods used to update the sufficient statistics matrix are much faster in that case. This is also important to note because the informR package does not check for affine collinearity between the sufficient statistics (i.e., as would be the case in `gen.sformlist(evl, c("aa","a+a","aaa"))`.

To generate S-form statistics that condition out the prefix set `cond=TRUE`. This will result in statistics of the form "(aa)b" or the likelihood of "b" given that "aa" occurred; thus, only the hazard of "b" is affected. This is useful when only the suffix event is of interest. When elements of each S-form are "paired," such as interval likelihood cases where each spell class can be characterized as having "starting" and "stopping" elements, setting `interval=TRUE` will result in two statistics per s-form: one that models the hazard of starting the suffix spell and one that models the duration of the suffix spell. In the general case, this is equivalent to a model with terms for "(ab)c" and "(ab)cd", respectively.

Value

A list containing idXevent, iXj matrices of sform sequence statistics.

Note

A notice will be issued if special regex characters are found in the sform vector.

Note

This routine will complain about poorly formed regular expressions.

Note

The `cond` parameter affects all S-forms in `sform`.

Note

S-form regular expressions that contain repitition or divergence ("+" or "|") flags are not allowed when `cond=TRUE`. See the example for how to do this manually.

Author(s)

Christopher Steven Marcum

See Also

`gen.sform`,`sfl2sl`,`glb.sformlist`

Examples

```
set.seed(57391)
rawevents<-sample(rep(c("ran","eat","stay","eat","ran","play"),50))
actors<-rep(c("Jim","Bill","Pete"),100)
evmat<-cbind(rawevents,actors)
eventlist<-gen.evl(evmat)
```

```
#See event-type alphabetic token codes
eventlist$event.key

alpha.ints<-gen.intercepts(eventlist,basecat="ran")

#Create some inertial s-form statistics and fit the models
beta.sforms<-gen.sformlist(eventlist,c("aa","bb","cc","dd"))

#Condition out the effects of the prefix:
gamma.sforms<-gen.sformlist(eventlist,c("aab","abb","acc","add"),cond=TRUE)

#Manual example of the above
sforms1<-c("aab","abb","acc","add")
sforms2<-sapply(sforms1,function(x) substr(x,1,nchar(x)-1))

sforms1.sf<-gen.sformlist(eventlist, sforms1)
sforms2.sf<-gen.sformlist(eventlist, sforms2)

for(i in 1:length(sforms1.sf)){
   for(j in 1: dim(sforms1.sf[[1]])[[3]]){
      sforms1.sf[[i]][,,j]<-abs(sforms1.sf[[i]][,,j]-sforms2.sf[[i]][,,j])
   }
}

#Note the difference:
gamma.sforms2<-gen.sformlist(eventlist,c("aab","abb","acc","add"))
gamma.sforms2[[1]][75:85,,1]
gamma.sforms[[1]][75:85,,1]
```

get.regpos *Get Positions of Events Within Regular Expression Match*

Description

A convenience function that obtains event positions from inside a regular expression match.

Usage

```
get.regpos(gregx)
```

Arguments

gregx	A regular expression of type `gregexpr()`

Author(s)

Christopher Steven Marcum

See Also

gregexpr

---

glapply *Group Level Apply*

Description

Apply a function on a variable over levels of an grouping factor. A wrapper for `split()` and is similar in functionality to `by()` but with more options for output.

Usage

```
glapply(x, id, FUN, regroup = TRUE, ...)
```

Arguments

`x`	a string or numeric vector.
`id`	grouping factor with `length(x)`
`FUN`	Function to apply over `x`
`regroup`	logical, if TRUE returns result in the same structure as `x`, otherwise, returns a list of length `unique(id)`.
`...`	Additional arguments passed to FUN

Author(s)

Christopher Steven Marcum

See Also

split,unsplit,lapply

Examples

```
data(Indometh)
glapply(Indometh$conc,Indometh$Subject,FUN=mean)
x<-rep(sample(1:10),2)
id<-rep(letters[1:5],4)
glapply(x,id,FUN=function(x)sum((x-mean(x))^2),regroup=FALSE)
glapply(x,id,FUN=function(x)sum((x-mean(x))^2),regroup=TRUE)
#Compare with output produced by by()
by(x,id,function(x)sum((x-mean(x))^2))
```

`glb.sformlist` *Generate a list of global S-form sequence statistics.*

Description

Generates a list of S-form sequence statistics constructed by pooling multiple sform regular expressions into single statistics. Output is not used directly but is passed to other methods.

Usage

```
glb.sformlist(evl, sforms, new.names, dichot, cond=FALSE, interval=FALSE, warn=TRUE)
```

Arguments

`evl`	An eventlist passed from `gen.evl()`
`sforms`	A list of character strings of S-form regular expressions grouped by a common attribute. See details.
`new.names`	A character string equal to the length of sforms ideally representing the common attributes of each pooled set of sforms.
`dichot`	Logical. Should the resulting statistics be dichotomized? Defaults to TRUE. See details.
`cond`	Logical. Should the prefix be dropped from the resulting statistics? Defaults to FALSE.
`interval`	Logical. Pass interval flag to gen.sformlits? Defaults to FALSE.
`warn`	Logical. Should warnings be issued? Defaults to TRUE.

Details

As with `gen.sformlist()`, regular expressions in `sforms` must adhere to PERL standards and, at this time, must make use of the alphabetic `evl$event.key` values in an eventlist object.

If `dichot` is set to FALSE then the components of the global sform that overlap will result in statistics that have been multiplied by a scalar equal to the number of overlaps. For example, in a situation where a global s-form is composed of the following terms: `c("aab","aac")`, any event “a” that occurs in an eventlist will have a corresponding value of 2 in the respective “a” column of the resulting statslist. In general, this is not the desired output and defaults to dichotomous statistics.

To generate S-form statistics that condition out the prefix set `cond=TRUE`. This will result in statistics of the form "(aa)b" or the likelihood of "b" given that "aa" occurred; thus, only the hazard of "b" is affected. This is useful when only the suffix event is of interest. When elements of each S-form are "paired," such as interval likelihood cases where each spell class can be characterized as having “starting” and “stopping” elements, setting `interval=TRUE` will result in two statistics per s-form: one that models the hazard of starting the suffix spell and one that models the duration of the suffix spell. In the general case, this is equivalent to a model with terms for "(ab)c" and "(ab)cd", respectively.

The `interval` parameter is a placeholder for future functionality. It currently passes the flag to `gen.sformlist()` and then fails to return what you think it should return.

Value

A list containing idXevent, iXj matrices of sform sequence statistics.

Note

A notice will be issued if special regex characters are found in any sform vector.

Note

This routine will complain about poorly formed regular expressions.

Note

The `cond` parameter affects all S-forms in `sform`.

Note

S-form regular expressions that contain repitition or divergence ("+" or "|") flags are not allowed when `cond=TRUE`. See the example in `help("gen.sformlist")` for how to do this manually.

Author(s)

Christopher Steven Marcum

See Also

`gen.sform`,`gen.sformlist`,`sfl2sl`

Examples

```
example(gen.sformlist)
x<-list(c("aa","bb","cc"),c("ba","ca"))
tb1<-glb.sformlist(eventlist,sforms=x,new.names=c("inertia","afollows"))
rem(eventlist$eventlist,slbind(tb1,alpha.ints))
```

`regmat.ind` *Regular Expresstion Matrix Index*

Description

Internal function for generating a matrix of regular expression position matches, usually mapped to a vector expansion of the input string.

Usage

```
regmat.ind(reg,evl,...)
```

Arguments

`reg`	character, The regular expression.
`evl`	eventlist, The eventlist on which regex in `reg` is to be performed.
`...`	additional parameters to pass to `gregexpr()`.

Value

A three column matrix of integers.

Author(s)

Christopher Steven Marcum

`sf2nms` *Translate S-form Regular Expressions*

Description

Translates S-forms into verbose names by regex substitution using the event.key in an eventlist.

Usage

```
sf2nms(event.key, sform)
```

Arguments

`event.key`	A two column matrix, possibly passed from `evls$event.key`
`sform`	An S-form regular expression using the `evls$event.key[,1]` uids.

Value

A list containing:

`character`	A “translation” by substitution of the sform in unique id form to its event type representation

Author(s)

Christopher Steven Marcum

Examples

```
rawevents<-sample(rep(c("ran","eat","stay","eat","ran","play"),100))
actors<-rep(c("Jim","Bill","Pete"),10)
eventlist<-cbind(rawevents,actors)
evls<-gen.evl(eventlist)

sform<-"ab"
sf2nms(evls$event.key,sform)
```

`sfl2sl` *Convert an sformlist object to a statslist object*

Description

Converts an sformlist object to a statslist object, which is compatible with `rem()` in Butts's relevent package.

Usage

```
sfl2sl(sformlist,exclude=NULL,eventlist=NULL)
sfl2statslist(sformlist,type=1)
```

Arguments

`sformlist`	an object of `class(sformlist)`
`exclude`	character vector, optional but useful if excluding events later. Must supply `eventlist` if non-null.
`eventlist`	optional, a two column matrix of events
`type`	integer indicating global or local statslist position. Defaults to 1.

Details

`sfl2sl` is an internal function used by `gen.sformlist` and should not normally be of any use to users.

`sfl2statslist`, however, will nicely convert any object returned by `gen.sformlist` or `glb.sformlist` into a statslist object. This is useful for constructing s-form models without intercepts passed from `gen.intercepts`.

Author(s)

Christopher Steven Marcum

See Also

`gen.sformlist`, `slbind`

`slbind` *Combine Statlist Arrays*

Description

Combines statslist arrays using a wrapper for `abind()` in the abind package.

Usage

```
slbind(sformstats, statslist, type = 1, new.names=FALSE,...)
```

Arguments

`sformstats`	An sformstats object, possibly passed from `sfl2sl`
`statslist`	A statslist object, possibly passed from `gen.intercepts`
`type`	Indicates where the combining is going to occur in the output statslist. 1 for global, 2 for local.
`new.names`	Either logical or character string. Choose the ouput variable names. See details.
`...`	Additional arguments passed to `sf2nms()` if new.names=TRUE. See details

Details

The `new.names` parameter defaults to FALSE, which sets the variable names in the output to be whatever was passed from the names of the kth elements of each ijk array in `sformstats`, as in the default behavior in `abind()`. Setting `new.names` to TRUE tries to guess variable names using `sf2nms`. If TRUE, then `...` must contain an event.key from the eventlist object that `statslist` was built upon (e.g, `event.key=eventlist$event.key`). If `new.names` is passed as a character vector, then its length must be equal to the number of the kth elements of each ijk array in `sformstats`.

In the case that `new.names` is TRUE, it is possible to retrieve the original variable names by: `names(dimnames(output[[x]][[type]]))`, where output is the statslist generated by `slbind()`, x is any index, and type is `type` as defined above.

Note

`slbind` can accept `abind` arguments.

Author(s)

Christopher Steven Marcum

See Also

`sfl2sl`,`gen.intercepts`,`sf2nms`

Examples

```
rawevents<-sample(rep(c("ran","eat","stay","eat","ran","play"),50))
actors<-rep(c("Jim","Bill","Pete"),100)
evmat<-cbind(rawevents,actors)
eventlist<-gen.evl(evmat)
beta.ints<-gen.intercepts(eventlist)
beta.sforms<-gen.sformlist(eventlist,c("a+b","bb"))
statslist<-slbind(beta.sforms,beta.ints)
statslist[[1]][[1]][1:3,,]

#Compare with:
statslist<-slbind(beta.sforms,beta.ints,new.names=TRUE,event.key=eventlist$event.key)
dimnames(statslist[[1]][[1]])
```

---

slbind.cond — *Add ActorXEvent Conditional or Interaction Variables to a Statslist Array*

Description

Combines a single actor-level attribute with sufficient statistics from a statslist array using a wrapper for `abind()` in the abind package.

Usage

```
slbind.cond(intvar, statslist, var.suffix, sl.ind=NULL,who.evs=NULL,type = 1,...)
```

Arguments

`intvar`	An actor-level numeric variable. See Details.
`statslist`	A statslist object, possibly passed from `gen.intercepts`.
`var.suffix`	A character string naming the new variable(s).
`sl.ind`	A numeric vector containing the statslist column indices to be interacted with intvar. Defaults to all columns. See Details.
`who.evs`	If type=2, a numeric vector indexing where to apply the interaction. Optional.
`type`	Indicates where the combining is going to occur in statslist. 1 for global, 2 for local.
`...`	Additional methods passed to `abind()`.

Details

For global statistics, `length(intvar)==length(statslist)` must be true. Because current functionality allows for only single vectors, the user must iterate over all levels of factors with more than two levels. Thus, this is truly a _single variable_ function and care must be taken.

The `sl.ind` parameter should be carefully specified as improper interactions can result in over-identified or unidentifiable models.

Note

slbind.cond can accept abind arguments.

Author(s)

Christopher Steven Marcum

See Also

slbind, abind,slbind.cov

Examples

```
rawevents<-sample(rep(c("ran","eat","stay","eat","ran","play"),50))
actors<-rep(c("Jim","Bill","Pete"),100)
evmat<-cbind(rawevents,actors)
eventlist<-gen.evl(evmat)
beta.ints<-gen.intercepts(eventlist)


statslist.new<-slbind.cond(intvar=c(1,0,0),beta.ints,var.suffix="Jim")
statslist.new[[1]][[1]][1,,]
statslist.new[[2]][[1]][1,,]
statslist.new[[3]][[1]][1,,]
```

---

slbind.cov *Add Covariates to a Statslist Array*

Description

Combines scalars and vectors to statslist arrays using a wrapper for abind() in the abind package.

Usage

```
slbind.cov(covar, statslist, type = 1,...)
```

Arguments

covar	A list of lists containing the covariates, see details
statslist	A statslist object, possibly passed from gen.intercepts
type	Indicates where the combining is going to occur in statslist. 1 for global, 2 for local.
...	Additional methods passed to abind().

Details

The length AND actor-order of the covar list must equal that of the statslist. Currently, no check is made to ensure that the actor-order is maintained, though an object length error will be returned in many faulty cases. Also, note that the number of covariates in each element of covar (i.e., for each actor) should be the same for proper model identification (also not currently checked).

Actor-level covariates are passed as single values and event-level covariates are passed as vectors. That is, each sub-element (covariate) of the covar list must either be a scalar or a vector with length equal to `length(eventlist$eventlist$actor)`.

If `names(covar$actor)` (for any and each actor) is NULL, then `slbind.cov()` will generate names using the `make.names=TRUE` parameter, as discussed in the `?abind` documentation.

Note

`slbind.cov` can accept `abind` arguments.

Author(s)

Christopher Steven Marcum

See Also

`slbind`, `abind`, `slbind.cond`

Examples

```
rawevents<-sample(rep(c("ran","eat","stay","eat","ran","play"),50))
actors<-rep(c("Jim","Bill","Pete"),100)
evmat<-cbind(rawevents,actors)
eventlist<-gen.evl(evmat)
beta.ints<-gen.intercepts(eventlist)

##Make up some covariates
covar<-list()
covar$Bill$rate<-sort(rexp(length(eventlist$eventlist$Bill)))
covar$Bill$smokes<-1
covar$Jim$rate<-sort(rexp(length(eventlist$eventlist$Jim)))
covar$Jim$smokes<-0
covar$Pete$rate<-sort(rexp(length(eventlist$eventlist$Pete)))
covar$Pete$smokes<-0

statslist.new<-slbind.cov(covar,beta.ints)
statslist.new[[1]][[1]][,1,]
```

sldrop — *Drop S-form Statistics or Covariates from a Statslist Array*

Description

Removes variables from a statslist.

Usage

```
sldrop(statslist, varname, type = 1)
```

Arguments

`statslist`	A statslist object, possibly passed from `gen.intercepts` or other methods
`varname`	A vector of variable names to remove from statslist. See details.
`type`	Indicates where the combining is going to occur in statslist. 1 for global, 2 for local.

Details

Removes the kth element(s) from the [i,j,k] statslist array based upon regex match of the k dimnames(statslist[[x]][[type]])[[3]].

Author(s)

Christopher Steven Marcum

See Also

`slbind`, `slbind.cov`, `abind`

Examples

```

#Take example from slbind.cov
example(slbind.cov)
statslist.new<-slbind.cov(covar,beta.ints)
statslist.new[[1]][[1]][,1,]

#And removes the "smokes" variable
dimnames(statslist.new[[1]][[1]])[[3]]
statslist.old<-sldrop(statslist.new,"smokes")
dimnames(statslist.old[[1]][[1]])[[3]]
```

Index