Package 'epiforecast'

July 6, 2016

Type Package

Title Tools for forecasting semi-regular seasonal epidemic curves and similar time series

Version 0.0.1

Date 2015-12-09

Author Logan C. Brooks, David C. Farrow, Sangwon Hyun, Ryan J. Tibshirani, Roni Rosenfeld

Maintainer Logan C. Brooks <lcbrooks@cs.cmu.edu>

Imports stats,
utils,
lubridate,
splines,
glmgen,
glmnet,
httr,
Rcpp

Description Tools for forecasting semi-regular seasonal epidemic curves and similar time series. Includes an empirical Bayes approach that forms a prior by transforming historical curves, a basis regression approach that balances matching observations from the current season and matching historical seasons' measurements for future weeks, and timestep-by-timestep weighted kernel density estimation on backward differences parameterized by both the time series measurments and the current time.

License GPL-2 | GPL-3

RoxygenNote 5.0.1

Collate 'RcppExports.R'
'match.R'
'utils.R'
'br.R'
'checks.R'
'delphi_epidata.R'
'eb_dists.R'
'weeks.R'

'loaders.R'
'interface.R'
'eb.R'
'twkde.R'

LinkingTo Rcpp

SystemRequirements C++14

Suggests testthat

R topics documented:

`augmentWeeklyDF` *Augment df with year/week/season/model.week, make weekly, full seasons*

Description

Given a df with with either (a) $year and $week, or (b) $date, fills in (if missing) $year, $week, $date, $season, and $model.week. Fills in missing weekly data from all seasons so that each season in `df$season` has all of its model weeks in `df$model.week`. Assumes epi week convention.

Usage

```
augmentWeeklyDF(df, first.week.of.season = NULL)
```

Arguments

`df`	`data.frame` with week numbers and other data
`first.week.of.season`	the first week number in each season or `NULL` (the default); if `NULL`, then the first week of the season is assumed to be the week of the first data point.

Details

Entries in `data.frame` are assumed without any checks to be sorted and weekly (potentially with some skipped weeks).

`br.sim` *Function for making forecasts with the basis regression method with output matching the format of distributional forecasting methods.*

Description

Function for making forecasts with the basis regression method with output matching the format of distributional forecasting methods.

Usage

```
br.sim(dat, new.dat.sim, baseline = NA_real_, n.sims = NULL, ...)
```

Arguments

dat	a list of numeric vectors, one per past season, containing historical trajectories; must not contain any NA's.
new.dat.sim	a numeric vector (trajectory), numeric matrix (cbound trajectories), or sim object (list with $ys a numeric matrix (cbound trajectories) and $weights a numeric vector (associated weights)), with NA's for all future or missing data points to forecast or infer; currently only supports NA's at future points, not mixed in between non-NA data
baseline	a single numeric: a "baseline level" for this dataset; roughly speaking, data below this level does not grow like an epidemic.
n.sims	single non-NA integer value or NULL: the number of curves to sample from the inferred distribution, or NULL to match the number of trajectories in new.dat.sim
...	arguments to forward to br.smoothedCurve.

Value

a list with two components:

ys: a numeric matrix; in most other methods, each column is a different possible trajectory for the current season, with NA's in new.dat filled in with random draws from the forecasted distribution, and non-NA's (observed data) filled in with an imagined resampling of noise based on the model. For the basis regression method, there is a single column per trajectory in new.dat containing the smoothed curve outputted by br.smoothedCurve, unless n.sims is non-NULL, in which case, it is a resampling of these smoothed curves.

weights: a numeric vector; assigns a weight to each column of ys, which is used by methods relying on importance sampling. For the basis regresion method, this is just the number 1.

Author(s)

Logan C. Brooks, David C. Farrow, Sangwon Hyun, Ryan J. Tibshirani, Roni Rosenfeld

Examples

```
fluview.nat.recent.df =
   trimPartialPastSeasons(fetchEpidataDF("fluview", "nat",
                          first.week.of.season=21L,
                          cache.file="fluview_nat_allfetch.Rdata"),
           "wili", min.points.in.season=52L)
## Recent historical seasons + current season, minus 2009 (nonseasonal
## pandemic) season:
full.dat = split(fluview.nat.recent.df$wili, fluview.nat.recent.df$season)
names(full.dat) <- sprintf("S%s", names(full.dat))
full.dat <- full.dat[names(full.dat)!="S2009"]
## Recent historical seasons minus 2009:
dat = head(full.dat, -1L)
## Current season:
new.dat = tail(full.dat, 1L)[[1]]
## Basis regression curve using CDC's 2015 national %wILI onset threshold baseline of 2.1:
sim = br.sim(dat, new.dat, 2.1)
```

br.smoothedCurve *Function for making forecasts with the basis regression method*

Description

Estimates missing values in dat.obj[[cur.season]] by regressing the mean of "psuedo-trajectories" formed from non-NA observations from dat.obj[[cur.season]] and "pseudo-observations" formed from dat.obj[-cur.season] on a set of basis elements.

Usage

```
br.smoothedCurve(dat.obj, cur.season, smooth = TRUE, basis = "bs",
  scale.method = c("none", "max", "last"), baseline = NA_real_,
  max.scale.factor = 3, df = 10, w = 1, max.match.length = NULL,
  cv.rule = c("min", "1se"), verbose = FALSE)
```

Arguments

dat.obj	assumed to be a list, of length equal to number of past seasons. Each item here is itself a list, each component containing a vector of "signals" for that seasons.
cur.season	the number of the season to be forecast. Must be in between 1 and the length of dat.obj.
smooth	logical; if TRUE, past observations and future "pseudo-observations" (predictions) will be smoothed; if FALSE, the observations and pseudo-observations will be returned unsmoothed.
basis	type of basis to use. So far only "bs" (B-splines) are implemented.
scale.method	whether and how to scale past seasons to match data from the cur.seasonth trajectory: "none" performs no scaling; "max" scales the maximum of each other season's trajectory — restricted to times which correspond to non-NA values in the cur.seasonth trajectory — so that it matches the maximum of the cur.seasonth trajectory; and "last" performs the same scaling using data at the time corresponding to the latest observation in the cur.seasonth trajectory
baseline	the anchoring point used for scaling past season's data; data above the baseline are scaled about the baseline. The default value, NA, indicates to scale about 0 (regardless of sign).
max.scale.factor	single numeric: a limit on the amount of scaling performed by the scaling method: scale factors over max.scale.factor and under 1/max.scale.factor will be clipped.
df	the degrees of freedom for the basis. Default is 10.
w	the mixing weight between the two loss terms, as in: sum over obs times (yobs - f)^2 + w * sum over unobs times (ypast - f)^2, where yobs is the current season's observed data, ypast is the past season's data, suitably transformed, and f is the function to be estimated.

`max.match.length`
: the maximum number of past data points to which the spline is fitted. The default, NULL, indicates to use all past data points when fitting the spline.

`cv.rule`
: one of "min" or "1se", where "min" gives the usual rule, and "1se" uses the 1-standard-error rule.

`verbose`
: logical; if TRUE, progress information will be printed out to the terminal.

Details

First, constructs a pseudo-trajectory for each training trajectory (`dat.obj[-cur.season]`) by shifting the training trajectory so that the maximum of its observations at times where `dat.obj[[cur.season]]` is non-`NA` aligns more closely with the maximum of `dat.obj[[cur.season]]` (where it is non-`NA`); the alignment procedure consists of a time shift (so that the partial maximum of the training and test trajectories are the same) and a scale (controlled by `scale.method`, `baseline`, and `max.scale.factor`). The pseudo-trajectory is formed by taking `dat.obj[[cur.season]]` where it is non-`NA` and the aligned training trajectory where `dat.obj[[cur.season]]` is `NA`.

Second, the mean of the pseudo-trajectories is regressed on a collection of basis elements to produce a single curve that provides estimates for `dat.obj[[cur.season]]` where it is `NA`.

Value

a numeric vector containing a smoothed version of the past observations and future "pseudo-observations" (predictions).

Author(s)

Logan C. Brooks, David C. Farrow, Sangwon Hyun, Ryan J. Tibshirani, Roni Rosenfeld

---

bw.SJnrd0 *bw.SJ with bw.nrd0 as fallback*

Description

At least sometimes when calling bw.SJ on backfill updates (lag.info$residual), an error is generated ("sample is too sparse to find TD"). bw.SJnrd0 uses bw.SJ if it succeeds, and falls back to bw.nrd0 if it generates any error.

Usage

```
bw.SJnrd0(x)
```

Arguments

`x`
: numeric vector: the observations

Value

single numeric: the bandwidth selection

dat.to.matrix *Numeric matrix of the first* n *elements of each numeric vector in* dat.

Description

A more efficient implementation of `sapply(dat, `[`, seq_len(n))`. Any vectors in `dat` with length less than `n` are extended with `NA_real_`'s at the end.

Usage

```
dat.to.matrix(dat, n)
```

Arguments

dat	a list of numeric vectors
n	a single integer: the number of elements to take from each vector

Value

a `n`-by-`length(dat)` numeric matrix

Examples

```
dat = list(11:15, 21:26)
dat.to.matrix(dat, 5) # (5x2: dat[[2]] is cut off)
dat.to.matrix(dat, 6) # (6x2: dat[[1]] is extended with NA_real_)
n = 3
identical(c(n, length(dat)), dim(dat.to.matrix(dat, n)))
```

DatesOfSeason *Get the first weekday of every week in the given seasons*

Description

Get the first weekday of every week in the given seasons

Usage

```
DatesOfSeason(season, first.week, first.wday, owning.wday)
```

Arguments

season	integer-valued vector: season numbers
first.week	integer-valued vector: first week number of each season
first.wday	integer-valued vector of weekday numbers: first weekday number in each week
owning.wday	integer-valued vector of weekday numbers: weekday that determines the year to which a week will be assigned

Value

$length(season)$-length named list of 52/53-length Date vectors

---

DateToYearWeekWdayDF *Convert dates to a data frame of year-week-wday according to a specified convention.*

Description

Convert dates to a data frame of year-week-wday according to a specified convention.

Usage

```
DateToYearWeekWdayDF(date, first.wday, owning.wday)
```

Arguments

date	object compatible with as.Date: the dates to convert
first.wday	weekday number(s) (0–7, Sunday can be 0 or 7): the weekday that is considered the beginning of the week; typically Sunday or Monday
owning.wday	weekday number(s) (0–7, Sunday can be 0 or 7): a week is assigned to a given year if the owning weekday of that week falls in that year; typically first.wday or first.wday+3

Value

a data frame with three columns, $year, $week, and $wday, corresponding to the given dates using the convention specified by first.wday and owning.wday; each wday entry will be %in% 0:6.

---

eb.createForecasts *Function for making forecasts with the empirical Bayes method.*

Description

Function for making forecasts with the empirical Bayes method.

Usage

```
eb.createForecasts(dat, new.dat, fit.obj = NULL, time.of.forecast = NULL,
  control.list = get.eb.control.list())
```

Arguments

dat	a list of numeric vectors, one per past season, containing historical trajectories.
new.dat	a single numeric vector containing the observations for the current season so far, and possibly future data points as well (when performing retrospective analysis); should not contain any NA's.
fit.obj	a collection of fit curves and noise level estimates to use when forming the prior; defaults to `smooth.curves.to.fit(eb.fitSmoothCurves(dat))`; while the smoothing method is quite fast, repeated calls to `eb.createForecasts` may benefit from caching the smoothed curves and feeding them in each time.
time.of.forecast	integer in [0..`length(new.dat.partial)`]; if specified, the forecast is prepared as if `new.dat.partial[seq_len(time.of.forecast)]` was fed in.
control.list	optional control list to forward to `eb.createForecasts`.

Value

a list with two components:

ys: a numeric matrix; in most other methods, each column is a different possible trajectory for the current season, with NA's in new.dat filled in with random draws from the forecasted distribution, and non-NA's (observed data) filled in with an imagined resampling of noise based on the model.

weights: a numeric vector; assigns a weight to each column of ys, which is used by methods relying on importance sampling.

---

eb.fitSmoothCurves *Function for fitting smooth curves to past seasons' data*

Description

Arguments:

Usage

```
eb.fitSmoothCurves(dat.obj, method = c("ss", "tf"), cv.rule = c("min",
  "1se", "gcv"), tf.ord = 2, verbose = FALSE)
```

Arguments

dat.obj	assumed to be a list, of length equal to number of past seasons. Each item here is itself a list, each component containing a vector of "signals" for that seasons
method	one of "ss" and "tf". The former uses R's built-in smoothing spline method; the latter uses the glmgen package
cv.rule	one of "min", "1se", or "gcv": the rule for selecting the smoothing parameter, where "min" gives the CV usual rule, and "1se" uses the CV 1-standard-error rule, and "gcv" uses generalized cross-validation (more efficient, less accurate); the "ss" method accepts only "min" and "gcv"

tf.ord	the order of the piecewise polynomial fit by trend filtering. Default is 2
verbose	logical; if TRUE, progress information will be printed out to the terminal.
cv	if TRUE, uses cross-validation to find the smoothing parameter; if FALSE, uses generalized cross-validation (more efficient, less accurate)

Value

a list with components

smooth.obj: a list of the same dimension as dat.obj, except all observed signal values have all been replaced by smoothed values

sigma.hat: a vector of length equal to the number of seasons, each component giving an estimate of the standard deviation of the noise in that season's data

Examples

```
## author_header begin
## Copyright (C) 2016 Ryan Tibshirani
##
## This file is part of epiforecast.  Algorithms included in epiforecast were developed by Logan C. Brooks, David
##
## Research reported in this publication was supported by the National Institute Of General Medical Sciences of th
## author_header end
## license_header begin
## epiforecast is free software: you can redistribute it and/or modify
## it under the terms of the GNU General Public License as published by
## the Free Software Foundation, either version 2 of the License, or
## (at your option) any later version.
##
## epiforecast is distributed in the hope that it will be useful,
## but WITHOUT ANY WARRANTY; without even the implied warranty of
## MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE.  See the
## GNU General Public License for more details.
##
## You should have received a copy of the GNU General Public License
## along with epiforecast.  If not, see <http://www.gnu.org/licenses/>.
## license_header end

## source("eb.fitSmoothCurves.R")
## source("br.makeForecasts.R")

# Create some fake data

ns = 8
dat.obj = vector(mode="list",length=ns)
set.seed(0)

nt = 52
x = 1:nt

for (i in 1:ns) {
  y = sin(2*pi*runif(1,1,2.5)*x/max(x)) + rnorm(length(x),sd=0.5)
```

```
  dat.obj[[i]] = y
}

out1 = eb.fitSmoothCurves(dat.obj, method="ss", cv.rule="min", verb=TRUE)
out2 = eb.fitSmoothCurves(dat.obj, method="ss", cv.rule="gcv", verb=TRUE)
out3 = eb.fitSmoothCurves(dat.obj, method="tf", cv.rule="min", verb=TRUE)
out4 = eb.fitSmoothCurves(dat.obj, method="tf", cv.rule="gcv", verb=TRUE)

par(mfrow=c(1,ns))
for (i in 1:ns) {
  plot(x,dat.obj[[i]])
  lines(x,out1$smooth.obj[[i]],col=1)
  lines(x,out2$smooth.obj[[i]],col=2)
  lines(x,out3$smooth.obj[[i]],col=3)
  lines(x,out4$smooth.obj[[i]],col=4)
  legend("topright",lty=1,col=1:4,
         legend=c("SS, CV", "SS, GCV", "TF, CV", "TF, GCV"))
}

####

cur.season = 8
plot(dat.obj[[cur.season]])

t0 = 10
dat.obj.2 = dat.obj
dat.obj.2[[8]][-(1:t0)] = NA
points(br.smoothedCurve(dat.obj,cur.season,verbose=TRUE),col="red")
```

`eb.simulate` *Function for making forecasts with the empirical Bayes method.*

Description

Function for making forecasts with the empirical Bayes method.

Usage

```
eb.simulate(dat, new.dat, baseline, fit.obj = NULL,
  control.list = get.eb.control.list())
```

Arguments

`dat`	a list of numeric vectors, one per past season, containing historical trajectories.
`new.dat`	a single numeric vector with NA's for all future or missing data points to forecast or infer; currently only supports NA's for future data points, not mixed in with past data.
`baseline`	a "baseline level" for this dataset; roughly speaking, data below this level does not grow like an epidemic; currently ignored, but can be used as the `y.scale.baseline` by passing it through the `control.list` argument.

fit.obj	optional argument to forward to eb.createForecasts
control.list	optional argument to forward to eb.createForecasts; n.out is overridden with length(new.dat)

Value

a list with two components:

ys: a numeric matrix; each column is a different possible trajectory for the current season, with NA's in new.dat filled in with random draws from the forecasted distribution, and non-NA's (observed data) filled in with an imagined resampling of noise based on the model.

weights: a numeric vector; assigns a weight to each column of ys; generated by importance sampling.

Author(s)

Logan C. Brooks, David C. Farrow, Sangwon Hyun, Ryan J. Tibshirani, Roni Rosenfeld

---

fetchEpidataDF *Fetch & cache* R*hrefhttps://github.com/undefx/delphi-epidatadelphi-epidata data, convert it to a* data.frame

---

Description

Fetch & cache delphi-epidata data, convert it to a data.frame

Usage

```
fetchEpidataDF(source, area, lag = NULL, first.week.of.season = NULL,
  first.epiweek = NULL, last.epiweek = NULL, cache.file = NULL,
  cache.invalidation.period = as.difftime(1L, units = "days"),
  force.cache.invalidation = FALSE)
```

Arguments

source	length-1 character vector; name of data source; one of • "fluview":U.S. Outpatient Influenza-like Illness Surveillance Network (ILINet) data from CDC FluView reports; weekly, national or by HHS region; updated weekly • "ilinet":Estimates of ILINet data at state level; weekly; static • "gft":Google Flu Trends data • "ght":Google Health Trends data; currently restricted, not supported by this function • "twitter"Twitter data; currently restricted, not supported by this function • "wiki"Wikipedia access log data; currently not supported by this function • "nidss.flu"Taiwan National Infectious Disease Statistics System (NIDSS) outpatient ILI data.

- "nidss.flu"Taiwan National Infectious Disease Statistics System (NIDSS) dengue incidence data.

`area` length-1 character vector; name of area; possibilities by `source` are:

- "fluview":"hhs1", "hhs2", ..., "hhs10", "nat"
- "ilinet":2-character state abbreviation or "DC"
- "gft":listed at https://github.com/undefx/delphi-epidata/blob/master/labels/cities.txt
- "nidss.flu", "nidss.dengue":one of "nationwide", "central", "eastern", "kaoping", "northern", "southern", "taipei"

`lag` single integer value or `NULL`; for supported data sources for which observations for a particular time are revised at later times as more data is received, gives access to some of the older versions of the data. `NULL` gives the latest revision of the observation. An integer value gives the value for each observation `lag` weeks after its initial report; the value `0` corresponds to using the initial report for each data point. (This function does not reconstruct what a the data looked like at any particular point in the past, but can be used to do so.)

`first.week.of.season` single integer value giving the week number that seasons start with, or `NULL` to have seasons start with the week number of the first data point

`first.epiweek` year-epiweek as string in form "YYYYww" specifying a lower limit on times for which data will be fetched, or `NULL` to not impose a lower limit

`last.epiweek` year-epiweek as string in form "YYYYww" specifying an upper limit on times for which data will be fetched, or `NULL` to not impose an upper limit

`cache.file` single string; file path of Rdata file to create to cache the data from the delphi-epidata server, or `NULL` to not cache

`cache.invalidation.period` single `difftime`: time duration that must pass from last fetch for a new fetch to be performed instead of reading from the cache; default is one day

`force.cache.invalidation` single non-`NA` logical; if `TRUE`, then the `cache.invalidation.period` and a fetch will always be performed (the cache will not be read)

Value

`data.frame` with `$date`, `$year`, `$week`, and corresponding data (fields differ based on `source`)

Examples

```
## All fluview data at the national level:
fluview.nat.all.df =
   trimPartialPastSeasons(fetchEpidataDF("fluview", "nat",
                          first.week.of.season=21L,
                          cache.file="fluview_nat_allfetch.Rdata"),
           "wili", min.points.in.season=33L)
## Fluview data at the national level for more recent seasons for which data
## was reported for all weeks (not just those in the influenza season), plus
## the current season:
```

```
fluview.nat.recent.df =
   trimPartialPastSeasons(fetchEpidataDF("fluview", "nat",
                          first.week.of.season=21L,
                          cache.file="fluview_nat_allfetch.Rdata"),
           "wili", min.points.in.season=52L)
```

`fetchEpidataFullDat` *Fetch data from delphi-epidata, trim partial seasons, and convert to list of trajectories*

Description

Fetch data from delphi-epidata, trim partial seasons, and convert to list of trajectories

Usage

```
fetchEpidataFullDat(source, area, signal.ind, min.points.in.season,
  lag = NULL, first.week.of.season = NULL, first.epiweek = NULL,
  last.epiweek = NULL, cache.file = NULL,
  cache.invalidation.period = as.difftime(1L, units = "days"),
  force.cache.invalidation = FALSE)
```

Arguments

`source` length-1 character vector; name of data source; one of

- "fluview":U.S. Outpatient Influenza-like Illness Surveillance Network (ILINet) data from CDC FluView reports; weekly, national or by HHS region; updated weekly
- "ilinet":Estimates of ILINet data at state level; weekly; static
- "gft":Google Flu Trends data
- "ght":Google Health Trends data; currently restricted, not supported by this function
- "twitter"Twitter data; currently restricted, not supported by this function
- "wiki"Wikipedia access log data; currently not supported by this function
- "nidss.flu"Taiwan National Infectious Disease Statistics System (NIDSS) outpatient ILI data.
- "nidss.flu"Taiwan National Infectious Disease Statistics System (NIDSS) dengue incidence data.

`area` length-1 character vector; name of area; possibilities by `source` are:

- "fluview":"hhs1", "hhs2", ..., "hhs10", "nat"
- "ilinet":2-character state abbreviation or "DC"
- "gft":listed at https://github.com/undefx/delphi-epidata/blob/master/labels/cities.txt
- "nidss.flu", "nidss.dengue":one of "nationwide", "central", "eastern", "kaoping", "northern", "southern", "taipei"

`signal.ind` single non-NA character/integer-valued index for column of `df`

`min.points.in.season`
the minimum number of non-`NA` values for `signal.ind` that a season must have in order to be retained; all rows corresponding to seasons containing less observations will be removed from `df`

`lag` single integer value or `NULL`; for supported data sources for which observations for a particular time are revised at later times as more data is received, gives access to some of the older versions of the data. `NULL` gives the latest revision of the observation. An integer value gives the value for each observation `lag` weeks after its initial report; the value `0` corresponds to using the initial report for each data point. (This function does not reconstruct what a the data looked like at any particular point in the past, but can be used to do so.)

`first.week.of.season`
single integer value giving the week number that seasons start with, or `NULL` to have seasons start with the week number of the first data point

`first.epiweek` year-epiweek as string in form "YYYYww" specifying a lower limit on times for which data will be fetched, or `NULL` to not impose a lower limit

`last.epiweek` year-epiweek as string in form "YYYYww" specifying an upper limit on times for which data will be fetched, or `NULL` to not impose an upper limit

`cache.file` single string; file path of Rdata file to create to cache the data from the delphi-epidata server, or `NULL` to not cache

`cache.invalidation.period`
single `difftime`: time duration that must pass from last fetch for a new fetch to be performed instead of reading from the cache; default is one day

`force.cache.invalidation`
single non-`NA` logical; if `TRUE`, then the `cache.invalidation.period` and a fetch will always be performed (the cache will not be read)

Value

named list of 52/53-length is.numeric vectors; each vector is the trajectory of the given signal for a single season, with NA's used to fill in for missing and future data; the names are of the form "SYYYY", where YYYY is the first year of the season, e.g., "S2003" corresponds to the 2003–2004 season.

Examples

```
## All fluview data at the national level:
fluview.nat.all.full.dat =
  fetchEpidataFullDat("fluview", "nat", "wili", 33L, first.week.of.season=21L,
                      cache.file="fluview_nat_allfetch.Rdata")
## Fluview data at the national level for more recent seasons for which data was
## reported for all weeks (not just those in the influenza season), plus the
## current season:
fluview.nat.recent.full.dat =
  fetchEpidataFullDat("fluview", "nat", "wili", 52L, first.week.of.season=21L,
                      cache.file="fluview_nat_allfetch.Rdata")
```

`firstEpiweekOfUniverse` *Epiweek before which no data should exist*

Description

Assume that the universe was created in 1234 AD EW01, and that no data will be from times before this.

Usage

```
firstEpiweekOfUniverse
```

Format

An object of class `integer` of length 1.

`fit.eb.control.list` *Takes an EB control list containing arguments that may require fitting to a curve object, performs any fitting, and outputs a static EB control list containing the results of the fitting procedure. The contents of the EB control list are also validated and standardized to a more rigid form, e.g, replacing some* `NULL` *values with defaults and some non-integer-class integral input with integer-class versions.*

Description

Takes an EB control list containing arguments that may require fitting to a curve object, performs any fitting, and outputs a static EB control list containing the results of the fitting procedure. The contents of the EB control list are also validated and standardized to a more rigid form, e.g, replacing some `NULL` values with defaults and some non-integer-class integral input with integer-class versions.

Usage

```
fit.eb.control.list(dat, new.dat, fit.obj, time.of.forecast, control.list)
```

`get.eb.control.list` *Generates a control list for* `eb.createForecasts`.

Description

With no arguments, returns the default control list. Optional arguments provided to the function will override these default values (currently with no validation checks).

Usage

```
get.eb.control.list(parent = NULL, n.sims = 20000L, peak.time.dist = NULL,
  x.shift.dist = NULL, x.scale.dist = NULL, y.scale.baseline = 0,
  peak.height.dist = NULL, y.scale.dist = NULL, sd.option = c("match",
  "scale", "prior"), sd.prior = "uniform", sd.scale.dist = NULL,
  reasonable.future.weight = 0, n.future.neighbors = 3L,
  inactive.seasons = NULL, n.out = 53L, ii.match.mask = NULL,
  max.match.length = NULL, n.unpinned.observations = 0L)
```

Arguments

`parent`

`n.sims` the number of simulated curves to generate in a forecast

`peak.time.dist` the distribution of smoothed-curve peak times that the prior should follow. If enabled, each smoothed curve will be x-shifted to have a peak time which is drawn from this distribution. The default setting is to disable this transformation. The default enabled distribution is a discrete uniform distribution fitted to the peak times of the smoothed curves provided to `eb.createForecasts` in the argument `fit.obj`.

`x.shift.dist` the distribution of x-shifts to apply (after any x-shift from `peak.time.dist`). The default setting is to enable this transformation. The default enabled distribution is a discrete uniform distribution centered at zero with width equal to twice the bin width of a histogram of the `fit.obj` peak times, using Sturges' rule.

`x.scale.dist`

`y.scale.baseline`

a single numeric value. Any y-scale transforms will only transform about and above this baseline value; for example, for a baseline of 4 and scaling factor of 2, the y-value 1 will not be scaled, since 1<4, and the y-value 5 will be scale to 5+(5-4)*2=6. The default is 0, which, for non-negative smoothed curves, corresponds to simply multiplying y-values by the scaling factor.

`peak.height.dist`

the distribution of smoothed-curve peak heights that the prior should follow. If enabled, each smoothed curve will be y-scaled to have a peak height which is drawn from this distribution. The default setting is to disable this transformation. The default enabled distribution is a uniform distribution fitted to

the `fit.obj` peak heights. If a smoothed curve remains completely below `y.scale.baseline` the entire time, then y-scaling will have no effect on that curve, and the peak height will remain at its original value. If a peak height selected from the distribution is lower than `y.scale.baseline`, parts of the curve above the baseline will be scaled by a negative factor so that the original peak is mapped to the drawn peak height value; however, this inversion is likely undesirable, and the resulting peak height may be any value between the drawn peak height value and the baseline value.

`y.scale.dist` the distribution of y-scales to apply (after any y-scale from `peak.height.dist`). The default setting is to enable this transformation. The default enabled transformation is a log-uniform distribution centered at 0 in the log-scale with log-scale width equal to twice the bin width of a histogram of the logarithms of the `fit.obj` peak heights, using Sturges' rule. Note that this default behavior can significantly bias the mean of the prior for the peak heights, but does not significantly affect the median of the prior for the peak heights.

`sd.option` one of `"match"`, `"scale"`, or `"prior"`, which controls the assignment of a single noise level, or distribution of possible noise levels, to a transformed curve: `"match"` chooses the `sigma.hat` associated with the selected smooth curve; `"scale"` does the same, but scales this `sigma.hat` by the y-scale factor given by the transformations selected from `peak.height.dist` and `y.scale.dist`; `"prior"` selects a noise level uniformly from the `sigma.hat`'s of all smoothed curves fed into the EB method, not just the one corresponding to the current transformed curve.

`sd.prior` controls the distribution of noise levels used when `sd.option` is `"prior"`; currently, the only choice is `"uniform"`.

`sd.scale.dist` controls the distribution of noise level scaling factors, which are applied after `sd.option` and `sd.prior` are used to select an initial noise level. The default setting is to disable this transformation. There is no default enabled distribution.

`reasonable.future.weight`
controls the coefficient of the "reasonable future" term added to the conditional log-likelihood of the observed values given a transformed curve and noise level when calculating importance weights. The default value is 0, which disables this feature.

`n.future.neighbors`
controls the number of neighbors to use when determining the "reasonable future" term: for a given transformed curve and noise parameter, the neighbors are the `n.future.neighbors` historical noisy curves from the`dat` argument in `eb.createForecasts` with the highest log-likelihoods in future weeks (after `time.of.forecast`; the reasonable future term is the average across these neighbors of the log-likelihood in future weeks.

`inactive.seasons`
is currently ignored.

`n.out` is the number of observations that each outputted noisy curve should contain; it should be less than or equal to the length of the shortest smooth curve. For weekly data and year-long seasons, this should be 52 or 53.

`ii.match.mask` is a vector of indices in `seq_len(n.out)`; only observations at these times will be considered when computing the likelihood of observations and assigning "reasonable future" terms to transformed curves and noise levels.

`max.match.length`
is a single integer controlling the maximum number of observations to use when computing the log-likelihood of `new.dat` given a transformed curve and noise level; if more than `max.match.length` observations are available at `time.of.forecast` after applying `ii.match.mask`, only the `max.match.length` most recent observations are used in the likelihood calculation.

`n.unpinned.observations`
is a single integer controlling what values in the noisy transformed curves in the posterior are "pinned" to the observed values in `new.dat`; any observations after `time.of.forecast-n.unpinned.observations` are not pinned.

Details

Most settings are single integers, single reals, or character vectors where the first entry holds the desired value. Transformation distribution settings, on the other hand, can be one of several options, with the following associated meanings:

- `NULL`: Use the default setting for this transformation, either disabling the transformation or using the default distribution
- `TRUE`: Enable this transformation, and use the default distribution for this transformation
- `FALSE`: Disable this transformation
- Single integer: Enable this transformation and use the default distribution for this transformation, but break the distribution into the specified number of bins (rather than the default) when applying the grid importance sampling algorithm used by `eb.createForecasts`
- Distribution with bins:
- Function from curve.obj to distribution with bins:

Value

a list of parameter settings used by `eb.createForecasts`

Author(s)

Logan C. Brooks, David C. Farrow, Sangwon Hyun, Ryan J. Tibshirani, Roni Rosenfeld

Examples

```
default.control.list = get.eb.control.list()
with.less.sims = get.eb.control.list(n.sims = 10000L)
with.less.sims.another.way = get.eb.control.list(default.control.list, n.sims = 10000L)
with.less.sims.and.sd.option.scale = get.eb.control.list(with.less.sims, sd.option="scale")
```

lastWeekNumber *The number of weeks assigned to a given year or years.*

Description

The length of one input should be evenly divisible by the length of the other.

Usage

```
lastWeekNumber(year, owning.wday)
```

Arguments

year	integer/NA-valued vector of years
owning.wday	integer vector of wday numbers (0–7, 0 and 7 are both Sunday): a week is assigned to a given year based on whether this weekday is contained in that year

Value

the number of weeks assigned to each year in years

Examples

```
## The number of epi weeks in each year from 1990 to 2020:
lastWeekNumber(1990:2020, 3)
```

match.arg.else.default

match.arg *variant replacing unmatched args with* choices[[1]]*, allowing non-character choices*

Description

Assumes this usage: parent_fun = function(parent_arg[=parent_choices]) ... match.arg.forgiving(parent_arg) ... (with or without "=parent_choices").

Usage

```
match.arg.else.default(arg, choices)
```

Arguments

arg	the argument to match to a choice; should
choices	a positive-length vector; if it contains NULL, first choice should should be NULL to avoid ambiguity

Details

If `arg` is `NULL`, returns `choices[[1]]`.

If `choices` is a character vector, this performs partial matches; otherwise, it checks for `arg`'s that are `all.equal` with `check.attributes=FALSE`.

Value

`arg`, the corresponding match in `choices`, or `choices[[1]]` with a warning (when `arg` fails to match a choice)

Author(s)

Logan C. Brooks, David C. Farrow, Sangwon Hyun, Ryan J. Tibshirani, Roni Rosenfeld

Examples

```
library(testthat)
## author_header begin
## Copyright (C) 2016 Logan C. Brooks
##
## This file is part of epiforecast.  Algorithms included in epiforecast were developed by Logan C. Brooks, David
##
## Research reported in this publication was supported by the National Institute Of General Medical Sciences of th
## author_header end
## license_header begin
## epiforecast is free software: you can redistribute it and/or modify
## it under the terms of the GNU General Public License as published by
## the Free Software Foundation, either version 2 of the License, or
## (at your option) any later version.
##
## epiforecast is distributed in the hope that it will be useful,
## but WITHOUT ANY WARRANTY; without even the implied warranty of
## MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE.  See the
## GNU General Public License for more details.
##
## You should have received a copy of the GNU General Public License
## along with epiforecast.  If not, see <http://www.gnu.org/licenses/>.
## license_header end

parent_function = function(ch1=letters[1:5], ch2=c("AAA","AAB","BBB"),
                           num1=1:5, int1=6L:10L,
                           list1=list(1:2,3:4), list2=list(NULL,"a",c(two=2),1:5)) {
  return (list(
    ch1=match.arg.else.default(ch1),
    ch2=match.arg.else.default(ch2),
    num1=match.arg.else.default(num1),
    int1=match.arg.else.default(int1),
    list1=match.arg.else.default(list1),
    list2=match.arg.else.default(list2)
  ))
}
```

```
## Return default on missing:
expect_equal(parent_function(),
             list(ch1="a", ch2="AAA", num1=1, int1=6L, list1=1:2, list2=NULL))

## Return default on NULL:
expect_equal(parent_function(NULL, NULL, NULL, NULL, NULL, NULL),
             list(ch1="a", ch2="AAA", num1=1, int1=6L, list1=1:2, list2=NULL))

## Allow partial matches, all.equal ignoring attributes:
expect_equal(parent_function("b", c(extraneous.name="B"), 3L, 8.00000000001, c(p=3,q=4), 2),
             list(ch1="b", ch2="BBB", num1=3.0, int1=8L, list1=3:4, list2=c(two=2)))

## Return default with warning on mismatched inputs:
expect_equal(suppressWarnings(parent_function("q", "A", "nonnumeric", 11L, 1:4, c("A","B","C"))),
             list(ch1="a", ch2="AAA", num1=1, int1=6L, list1=1:2, list2=NULL))
expect_warning(parent_function("q"))
expect_warning(parent_function(,"A"))
expect_warning(parent_function(,,"nonnumeric"))
expect_warning(parent_function(,,,11L))
expect_warning(parent_function(,,,,1:4))
expect_warning(parent_function(,,,,,c("A","B","C")))

## Produce error on inappropriate inputs:
expect_error(parent_function(3), "length-1 character")
expect_error(parent_function(letters[1:2]), "length-1 character")

## todo produce error on inappropriate =choices=
```

`match.dat` *Match dat object input*

Description

Returns a list of possibly-named numeric-class vectors given a list of possibly-named (is.)numeric vectors as input, or generates an error if the input seems inappropriate.

Usage

```
match.dat(dat)
```

Arguments

`dat`	supposed to be a list of possibly-named numeric vectors

Value

`dat` as a list of numeric-class vectors

Author(s)

Logan C. Brooks, David C. Farrow, Sangwon Hyun, Ryan J. Tibshirani, Roni Rosenfeld

---

`match.dist` *Returns a distribution that can be divided into buckets — a named list containing* `‘n‘`*, a single integer representing the number of buckets into which the distribution can be broken down;* `‘choices‘`*, an* `n`*-length vector containing a representative element from each bucket;* `‘probs‘`*, an* `n`*-length vector containing the probability mass assigned to each bucket, and* `‘sampler‘`*, a function from vectors of bucket indices (in* `seq_len(n)`*) to randomly sampled elements within the corresponding buckets. Designed for input processing and validation within another function.*

Description

Returns a distribution that can be divided into buckets — a named list containing `` `n` ``, a single integer representing the number of buckets into which the distribution can be broken down; `` `choices` ``, an n-length vector containing a representative element from each bucket; `` `probs` ``, an n-length vector containing the probability mass assigned to each bucket, and `` `sampler` ``, a function from vectors of bucket indices (in `seq_len(n)`) to randomly sampled elements within the corresponding buckets. Designed for input processing and validation within another function.

Usage

```
match.dist(curve.obj, dist, null.replacement, true.replacement,
  false.replacement, integer.replacement.fn)
```

Arguments

`curve.obj`	output of `eb.fitSmoothCurves`
`dist`	one of the following: (a) `NULL`, (b) a single boolean, (c) a single integer, (d) a function that outputs a distribution given a `curve.obj`, or (e) a distribution.
`null.replacement`	one of (b)–(e), which is used to replace `NULL` inputs
`true.replacement`	one of (c)–(e), which is used to replace `TRUE` inputs
`false.replacement`	one of (c)–(e), which is used to replace `FALSE` inputs
`integer.replacement.fn`	a function from a single integer to either (d) or (e), called on integer inputs to produce a replacement value

Value

a distribution, which incorporates any fitting procedure to `curve.obj`;

match.integer *Match integer-valued input*

Description

Returns a possibly-named integer-class vector version of the input, or produces an error if the input seems inappropriate.

Usage

```
match.integer(inp)
```

Arguments

inp	supposed to be a possibly-named numeric object with integer/NA values.

Value

inp as an integer vector

Author(s)

Logan C. Brooks, David C. Farrow, Sangwon Hyun, Ryan J. Tibshirani, Roni Rosenfeld

match.new.dat.sim *Convert trajectory, trajectory matrix, or sim to sim; error otherwise*

Description

*.sim methods should eventually all support sim objects as input for new.dat rather than just single trajectories, but, for the convenience of the user, allow other types of input as well. Specifically, we should accept:

- Trajectory: a numeric vector;
- Trajectory matrix: a numeric matrix with each column a trajectory; and
- Sim: a list with $ys a #times by #trajectories numeric matrix with #trajectories >= 1 and each row either all NA or all non-NA, and $weights a #trajectories-length numeric matrix with entries all >= 0.

Usage

```
match.new.dat.sim(new.dat.sim)
```

Arguments

new.dat.sim	trajectory / trajectory matrix / sim object

Details

This method checks that it receives such an input and outputs a corresponding sim object.

Value

sim object

Author(s)

Logan C. Brooks, David C. Farrow, Sangwon Hyun, Ryan J. Tibshirani, Roni Rosenfeld

Examples

```
match.new.dat.sim(1:5)
match.new.dat.sim(as.matrix(1:5)[,rep(1,10)])
match.new.dat.sim(list(ys=as.matrix(1:5)[,rep(1,10)], weights=0:9))
```

---

match.single.integer *Returns a possibly-named length-1 non-NA integer-class vector version of the input, or produces an error if the input seems inappropriate. Designed for input processing and validation within another function.*

Description

Returns a possibly-named length-1 non-NA integer-class vector version of the input, or produces an error if the input seems inappropriate. Designed for input processing and validation within another function.

Usage

```
match.single.integer(n)
```

---

match.single.na.or.numeric *Match length-1 numeric input*

Description

Returns a possibly-named length-1 possibly-NA numeric vector version of the input, or produces an error if the input seems inappropriate.

Usage

```
match.single.na.or.numeric(x)
```

Arguments

x	supposed to be a possibly-named length-1 numeric object

Value

x as a length-1 numeric vector

Author(s)

Logan C. Brooks, David C. Farrow, Sangwon Hyun, Ryan J. Tibshirani, Roni Rosenfeld

---

match.single.nonna.integer

Match length-1 non-NA integer-valued input

Description

Returns a possibly-named length-1 non-NA integer-class vector version of the input, or produces an error if the input seems inappropriate.

Usage

```
match.single.nonna.integer(n)
```

Arguments

n	supposed to be a possibly-named length-1 non-NA integer-valued numeric object

Value

n as a length-1 integer-class vector

Author(s)

Logan C. Brooks, David C. Farrow, Sangwon Hyun, Ryan J. Tibshirani, Roni Rosenfeld

`match.single.nonna.integer.or.null` *Match length-1 non-*`NA` *integer-valued input or* `NULL`

Description

Returns a possibly-named length-1 non-NA integer-class vector version of the input if non-`NULL`, `NULL` if input is `NULL`, or produces an error if the input seems inappropriate.

Usage

```
match.single.nonna.integer.or.null(n)
```

Arguments

`n`	supposed to be a possibly-named length-1 non-`NA` integer-valued numeric object or `NULL`

Value

`n` as a length-1 integer-class vector

Author(s)

Logan C. Brooks, David C. Farrow, Sangwon Hyun, Ryan J. Tibshirani, Roni Rosenfeld

`match.single.wday.w` *Returns a possibly-named length-1 integer-class vector of weekday numbers in 0:6 (%w format) given an integer-valued length-1 numeric vector with values in 0:7 (%w or %u format), or throws an error if the input seems inappropriate.*

Description

Returns a possibly-named length-1 integer-class vector of weekday numbers in 0:6 (%w format) given an integer-valued length-1 numeric vector with values in 0:7 (%w or %u format), or throws an error if the input seems inappropriate.

Usage

```
match.single.wday.w(wday)
```

Arguments

`wday`	vector of weekday numbers (each `%in% 0:7`)

Value

integer-class %w format weekday numbers

`match.wday.w` *Returns a possibly-named integer-class vector of weekday numbers in 0:6 (%w format) given an integer-valued numeric vector with values in 0:7 (%w or %u format), or throws an error if the input seems inappropriate.*

Description

Returns a possibly-named integer-class vector of weekday numbers in 0:6 (%w format) given an integer-valued numeric vector with values in 0:7 (%w or %u format), or throws an error if the input seems inappropriate.

Usage

```
match.wday.w(wday)
```

Arguments

`wday`	vector of weekday numbers (each `%in% 0:7`)

Value

integer-class %w format weekday numbers

`seasonModelWeekDFToYearWeekDF` *Convert season-model.week in* `data.frame` *to year-week*

Description

Like `yearWeekDFToSeasonModelWeekDF`, but in opposite direction.

Usage

```
seasonModelWeekDFToYearWeekDF(seasonModelWeek, first.week, owning.wday)
```

Arguments

`seasonModelWeek`	`data.frame` (or other list) with columns `$season`: integer-valued vector: season numbers `$model.week`: integer-valued vector: model week numbers
`first.week`	integer-valued vector: week number that the seasons start on
`owning.wday`	integer-valued vector of weekday numbers: weekday that determines the year to which a week will be assigned

Value

data.frame with two columns giving the corresponding year-week breakdown:

$year: integer-class vector: years

$week: integer-class vector: week numbers

Examples

```
dates = as.Date("2015-01-01")+seq.int(0L, 1000L, 7L)
ywwd = DateToYearWeekWdayDF(dates, 0L, 3L) # epi week convention
dates.duplicate1 = yearWeekWdayDFToDate(ywwd, 0L, 3L)
identical(dates, dates.duplicate1)
smw = yearWeekDFToSeasonModelWeekDF(ywwd, 21L, 3L) # seasons starting on week number 21
yw = seasonModelWeekDFToYearWeekDF(smw, 21L, 3L)
identical(ywwd[,c("year","week")], yw)
dates.duplicate2 = seasonModelWeekWdayDFToDate(cbind(smw, wday=ywwd$wday), 21L, 0L, 3L)
identical(dates, dates.duplicate2)
```

---

seasonModelWeekToYearWeekDF

Convert season-model.week to year-week

Description

Like yearWeekToSeasonModelWeekDF, but in opposite direction.

Usage

```
seasonModelWeekToYearWeekDF(season, model.week, first.week, owning.wday)
```

Arguments

season	integer-valued vector: season numbers
model.week	integer-valued vector: model week numbers
first.week	integer-valued vector: week number that the seasons start on
owning.wday	integer-valued vector of weekday numbers: weekday that determines the year to which a week will be assigned

Value

data.frame with two columns giving the corresponding year-week breakdown:

$year: integer-class vector: years

$week: integer-class vector: week numbers

seasonModelWeekWdayDFToDate

Convert season-model.week-wday `data.frame` *to* `Date`

Description

Convert season-model.week-wday `data.frame` to `Date`

Usage

```
seasonModelWeekWdayDFToDate(seasonModelWeekWday, first.week, first.wday,
  owning.wday, error.on.wrap = TRUE)
```

Arguments

`seasonModelWeekWday`	`data.frame` with `$season`, `model.week`, and `wday` columns
`first.week`	first week number of season
`first.wday`	first weekday number of week
`owning.wday`	integer-valued vector of weekday numbers: weekday that determines the year to which a week will be assigned
`error.on.wrap`	TRUE or FALSE: if TRUE, an error is generated if a given week falls outside the range of possible week numbers for the corresponding year (i.e., if it is nonpositive or greater than the number of weeks assigned to the corresponding year in `year`); if FALSE, the week number will wrap around to future or previous years (e.g., week 0 of 1997 will be considered the last week of 1996)

Value

`Date` vector with the corresponding dates

seasonModelWeekWdayToDate

Convert season-model.week-wday to `Date`

Description

Convert season-model.week-wday to `Date`

Usage

```
seasonModelWeekWdayToDate(season, model.week, wday, first.week, first.wday,
  owning.wday, error.on.wrap = TRUE)
```

Arguments

season	season number
model.week	model week number
wday	weekday number
first.week	first week number of season
first.wday	first weekday number of week
owning.wday	integer-valued vector of weekday numbers: weekday that determines the year to which a week will be assigned
error.on.wrap	TRUE or FALSE: if TRUE, an error is generated if a given week falls outside the range of possible week numbers for the corresponding year (i.e., if it is nonpositive or greater than the number of weeks assigned to the corresponding year in year); if FALSE, the week number will wrap around to future or previous years (e.g., week 0 of 1997 will be considered the last week of 1996)

Value

Date vector with the corresponding dates

---

seasonOfDate *Get the season number associated with a particular date*

Description

Seasons are 52/53-week-long time spans that start with a particular week number (from 1 to 52); years are a special case that start with week 1. Seasons contain weeks from two consecutive years except in the case that the season starts with week 1, in which case it coincides exactly with a single year. Seasons are numbered by the first distinct year from which they take weeks, and labeled with an "S" prefix; for example, a season containing weeks from 2015 and 2016 would be numbered 2015 and labeled "S2015". This function gives the season number associated with the dates.

Usage

```
seasonOfDate(date, first.week, first.wday, owning.wday)
```

Arguments

date	object convertible to Date
first.week	the week on which each season should start
first.wday	weekday number(s) (0–7, Sunday can be 0 or 7): the weekday that is considered the beginning of the week; typically Sunday or Monday
owning.wday	weekday number(s) (0–7, Sunday can be 0 or 7): a week is assigned to a given year if the owning weekday of that week falls in that year; typically first.wday or first.wday+3

Value

named integer-valued vector: season numbers associated with the inputted weeks with names giving the season labels

---

seasonOfYearWeek *Get the season number associated with a particular year and week*

Description

Seasons are 52/53-week-long time spans that start with a particular week number (from 1 to 52); years are a special case that start with week 1. Seasons contain weeks from two consecutive years except in the case that the season starts with week 1, in which case it coincides exactly with a single year. Each season ends the week before the next season (with the same starting week) begins. Seasons are numbered by the first distinct year from which they take weeks, and labeled with an "S" prefix; for example, a season containing weeks from 2015 and 2016 would be numbered 2015 and labeled "S2015". This function gives the season number associated with the inputted weeks specified as year-week combinations.

Usage

```
seasonOfYearWeek(year, week, first.week)
```

Arguments

year	integer-valued vector: year in which the weeks fall
week	integer-valued vector: associated week numbers (each `%in% 1:53`)
first.week	the week on which each season should start

Value

named integer-valued vector: season numbers associated with the inputted weeks with names giving the season labels

---

Seq *A* `seq` *variant that produces a 0-length vector when* `!(from <= to)`.

Description

A `seq` variant that produces a 0-length vector when `!(from <= to)`.

Usage

```
Seq(from, to, ...)
```

Arguments

from	starting number (or other object compatible with seq)
to	ending number (or other object compatible with seq)
...	arguments to forward to seq

---

trimPartialPastSeasons *Trims incomplete past seasons from a* data.frame

---

Description

Removes rows from df corresponding to "past" seasons (i.e., all but the last season in df) for which df has less than min.points.in.season non-missing entries in df[[signal.ind]].

Usage

```
trimPartialPastSeasons(df, signal.ind, min.points.in.season)
```

Arguments

df	data frame with columns df$season, and df[[signal.ind]]
signal.ind	single non-NA character/integer-valued index for column of df
min.points.in.season	the minimum number of non-NA values for signal.ind that a season must have in order to be retained; all rows corresponding to seasons containing less observations will be removed from df

---

twkde.markovian.sim *Time-parameterized kernel density estimation sim method, Markovian version*

Description

Function for making forecasts with the basic time-parameterized kernel density estimation method. This method estimates diff(new.dat)[t-1] (used to produce dat[t]) for a trajectory new.dat based on weighted kernel density estimation using values of dat at the corresponding time of season (a Markov process). The weights are based on new.dat[t-1] and the corresponding values in dat; the weighting function is a Gaussian kernel with width determined by bw.SJnrd0.

Usage

```
twkde.markovian.sim(dat, new.dat.sim, baseline = NA_real_, n.sims = 2000)
```

Arguments

dat	a list of numeric vectors, one per past season, containing historical trajectories.
new.dat.sim	a numeric vector (trajectory), numeric matrix (cbound trajectories), or sim object (list with $ys a numeric matrix (cbound trajectories) and $weights a numeric vector (associated weights)), with NA's for all future or missing data points to forecast or infer; currently only supports NA's at future points, not mixed in between non-NA data
baseline	a "baseline level" for this dataset; roughly speaking, data below this level does not grow like an epidemic; currently ignored, but can be used as the y.scale.baseline by passing it through the control.list argument.
n.sims	single non-NA integer value or NULL: the number of curves to sample from the inferred distribution, or NULL to match the number of trajectories in new.dat.sim

Value

a sim object (list with two components:

- ys: a numeric matrix; each column is a different possible trajectory for the current season, with NA's in new.dat.sim filled in with random draws from the forecasted distribution, and non-NA's (observed data) filled in with an imagined resampling of noise based on the model.
- weights: a numeric vector; assigns a weight to each column of ys, which is used by other methods that rely on importance sampling).

Author(s)

Logan C. Brooks, David C. Farrow, Sangwon Hyun, Ryan J. Tibshirani, Roni Rosenfeld

Examples

```
fluview.nat.recent.df =
   trimPartialPastSeasons(fetchEpidataDF("fluview", "nat",
                          first.week.of.season=21L,
                          cache.file="fluview_nat_allfetch.Rdata"),
           "wili", min.points.in.season=52L)
## Recent historical seasons + current season, minus 2009 (nonseasonal
## pandemic) season:
full.dat = split(fluview.nat.recent.df$wili, fluview.nat.recent.df$season)
names(full.dat) <- sprintf("S%s", names(full.dat))
full.dat <- full.dat[names(full.dat)!="S2009"]
## Recent historical seasons minus 2009:
dat = head(full.dat, -1L)
## Current season:
new.dat = tail(full.dat, 1L)[[1]]
## Sample from conditional curve distribution estimate using CDC's 2015
## national %wILI onset threshold baseline of 2.1:
sim = twkde.markovian.sim(dat, new.dat, 2.1, n.sims=100)
```

`twkde.sim` *Time-parameterized kernel density estimation sim method with heuristic adjustments*

Description

Function for making forecasts with the time-parameterized kernel density estimation method with tweaks. This method estimates `diff(new.dat)` based on weighted kernel density estimation. Weights are based on the time of season and four functions of `new.dat`: (a) the last observed value in `new.dat`, (b) the sum of observed values in `new.dat`, (c) an exponential moving average of the observed values in `new.dat`, and (d) an exponential moving average of the changes in observed values in `new.dat` (i.e., in `diff(new.dat)`). The weighting function is separable, and consists of two components: a highly weighted "base" weighting function and a lowly weighted boxcar weighting function. The base weighting function is the product of an integral Laplacian kernel with respect to time of season, and Gaussian kernels with respect to the four `new.dat`-based components (with bandwidths selected by the `bw.SJnrd0` method, and relative weighting controlled by `tradeoff.weights`). Each time a difference is drawn, simulating `diff(new.dat)[t-1]`, the corresponding result for `new.dat[t]` is linearly mixed with a randomly selected value from historical curves around that time.

Usage

```
twkde.sim(dat, new.dat.sim, baseline = NA_real_, n.sims = 2000,
  decay.factor = 0.7, diff.decay.factor = 0.5, max.shifts = c(rep(10, 20),
  10:1, rep(0, 3), 1:10, rep(10, 10)), shift.decay.factor = 0.7,
  tradeoff.weights = c(0.5, 0.25, 0.25, 0.5))
```

Arguments

`dat`	a list of numeric vectors, one per past season, containing historical trajectories.
`new.dat.sim`	a numeric vector (trajectory), numeric matrix (cbound trajectories), or sim object (list with $ys a numeric matrix (cbound trajectories) and $weights a numeric vector (associated weights)), with `NA`'s for all future or missing data points to forecast or infer; currently only supports `NA`'s at future points, not mixed in between non-`NA` data
`baseline`	a "baseline level" for this dataset; roughly speaking, data below this level does not grow like an epidemic; currently ignored, but can be used as the `y.scale.baseline` by passing it through the `control.list` argument.
`n.sims`	single non-`NA` integer value or `NULL`: the number of curves to sample from the inferred distribution, or `NULL` to match the number of trajectories in `new.dat.sim`
`decay.factor`	decay factor for the exponential moving average of covariate.
`diff.decay.factor`	decay factor for the exponential moving average of differences covariate.
`max.shifts`	numeric vector with length matching the trajectory length in `new.dat.sim`; specifies the width of the time-of-season kernel as a function of the time of season.

`shift.decay.factor`
: decay factor for the time-of-season Laplacian kernel component.

`tradeoff.weights`
: log-scale weighting factors for the four non-time-based kernel components (last observed value, sum of observed values, exponential moving average of values, exponential moving average of differences).

Value

a sim object (list with two components:

- `ys`: a numeric matrix; each column is a different possible trajectory for the current season, with `NA`'s in `new.dat.sim` filled in with random draws from the forecasted distribution, and non-`NA`'s (observed data) filled in with an imagined resampling of noise based on the model.
- `weights`: a numeric vector; assigns a weight to each column of `ys`, which is used by other methods that rely on importance sampling).

Author(s)

Logan C. Brooks, David C. Farrow, Sangwon Hyun, Ryan J. Tibshirani, Roni Rosenfeld

Examples

```
fluview.nat.recent.df =
   trimPartialPastSeasons(fetchEpidataDF("fluview", "nat",
                          first.week.of.season=21L,
                          cache.file="fluview_nat_allfetch.Rdata"),
           "wili", min.points.in.season=52L)
## Recent historical seasons + current season, minus 2009 (nonseasonal
## pandemic) season:
full.dat = split(fluview.nat.recent.df$wili, fluview.nat.recent.df$season)
names(full.dat) <- sprintf("S%s", names(full.dat))
full.dat <- full.dat[names(full.dat)!="S2009"]
## Recent historical seasons minus 2009:
dat = head(full.dat, -1L)
## Current season:
new.dat = tail(full.dat, 1L)[[1]]
## Sample from conditional curve distribution estimate using CDC's 2015
## national %wILI onset threshold baseline of 2.1:
sim = twkde.sim(dat, new.dat, 2.1, n.sims=50)
```

`unifChoicePrior` *Creates a uniform distribution over discrete choices which can be used with* `get.eb.control.list`.

Description

Creates a uniform distribution over discrete choices which can be used with `get.eb.control.list`.

Usage

```
unifChoicePrior(choices)
```

Arguments

choices	a vector of (discrete) choices

Value

a uniform discrete distribution over choices.

Examples

```
uniform.seq = unifChoicePrior(letters[1:5])

## The distributions used by EB can be broken down into buckets;
## for the uniform discrete distribution, each bucket corresponds
## (boringly) to a single choice from =choices=. However, it is
## important to have a common interface.
random.bucket.indices = sample(seq_len(uniform.seq$n), 10000, replace=TRUE, prob=uniform.seq$probs)
random.elements = uniform.seq$sampler(random.bucket.indices)
random.elements.another.way = uniform.seq$choices[random.bucket.indices] # only works for =unifChoicePrior=
random.elements.a.third.way = letters[random.bucket.indices] # only works for this example
```

---

weekConventions `first.wday` *and* `owning.wday` *for some week numbering conventions*

Description

Covers four common week numbering conventions:

- "epi": Epidemiological weeks or "epi weeks": weeks begin on Sunday, and are assigned to years based on what year the majority of days fall in (i.e., what year Wednesday falls in)
- "iso": ISO 8601 weeks: weeks begin on Monday, and are assigned to years based on what year the majority of days fall in (i.e., what year Thursday falls in)
- "usa": USA convention: weeks begin on Sunday, and are assigned to years based on what year Sunday falls in
- "uk": UK convention: weeks begin on Monday, and are assigned to years based on what year Monday falls in

Usage

```
weekConventions
```

Format

An object of class `matrix` with 2 rows and 4 columns.

Details

There are two rows, named `"first.wday"` and `"owning.wday"`. There are four columns, corresponding to the four conventions above.

`yearWeekDFToSeasonModelWeekDF` *Convert year-week in a* `data.frame` *to season-model.week*

Description

Delegates to `yearWeekToSeasonModelWeekDF`.

Usage

```
yearWeekDFToSeasonModelWeekDF(yearWeek, first.week, owning.wday)
```

Arguments

`yearWeek`	`data.frame` (or other list) with columns `$year` and `$week`: `$year` integer-valued vector: year from the year-week numbering `$week` integer-valued vector: week from the year-week numbering
`first.week`	integer-valued vector: week number that the seasons start on
`owning.wday`	integer-valued vector of weekday numbers: weekday that determines the year to which a week will be assigned

Value

`data.frame` with two columns giving the corresponding season-model.week breakdown:

`$season`: integer-class vector: season numbers

`$model.week`: integer-class vector: model week numbers

`yearWeekToSeasonModelWeekDF` *Convert year-week to season-model.week*

Description

Seasons starting with a week number `n` other than 1 contain week numbers `n` to 52/53 from the season's first year and 1 to `n-1` of the season's second year. Sometimes we would like a numbering of weeks within a season that coincides with the week number when possible, but does not have a jump down from 52/53 to 1. Model weeks fulfill this purpose: they begin with the starting week of a season and increase by 1 for each subsequent week; they coincide with week numbers in the starting year, and are 52/53 plus the week number in the second year.

Usage

```
yearWeekToSeasonModelWeekDF(year, week, first.week, owning.wday)
```

Arguments

`year`	integer-valued vector: year from the year-week numbering
`week`	integer-valued vector: week from the year-week numbering
`first.week`	integer-valued vector: week number that the seasons start on
`owning.wday`	integer-valued vector of weekday numbers: weekday that determines the year to which a week will be assigned

Value

`data.frame` with two columns giving the corresponding season-model.week breakdown:

`$season`: integer-class vector: season numbers

`$model.week`: integer-class vector: model week numbers

`yearWeekWdayDFToDate` *Convert a structure containing year, week, and weekday numbers into the corresponding dates under a specified week numbering convention. Reverses* `DateToYearWeekWdayDF`.

Description

Convert a structure containing year, week, and weekday numbers into the corresponding dates under a specified week numbering convention. Reverses `DateToYearWeekWdayDF`.

Usage

```
yearWeekWdayDFToDate(ywwd, first.wday, owning.wday, error.on.wrap = TRUE)
```

Arguments

ywwd	data frame with columns $year, $week, and $wday containing the year, week, and weekday numbers respectively (weekday numbers are 0–7; Sunday can be inputted either as 0 or as 7)
first.wday	wday number corresponding to the first weekday of any week
owning.wday	wday number; a week is assigned to a given year if the owning weekday of that week falls in the given year
error.on.wrap	TRUE or FALSE: if TRUE, an error is generated if a given week falls outside the range of possible week numbers for the corresponding year (i.e., if it is nonpositive or greater than the number of weeks assigned to the corresponding year in year); if FALSE, the week number will wrap around to future or previous years (e.g., week 0 of 1997 will be considered the last week of 1996)

Value

a Date vector: the dates corresponding to the year-week-wdays and week numbering convention

---

yearWeekWdayListsToDate

Like yearWeekWdayDFToDate, but allows parameters to be provided in several (is.)list structures and vectors.

Description

Like yearWeekWdayDFToDate, but allows parameters to be provided in several (is.)list structures and vectors.

Usage

```
yearWeekWdayListsToDate(...)
```

Arguments

...	vectors and (is.)lists to feed to yearWeekWdayVecsToDate; each entry in a list (e.g., column in a data.frame) is treated as a vector. Names can be provided by the names attribute of list arguments or optional parameter names for the vector arguments. (With no names, parameters of yearWeekWdayVecsToDate are matched sequentially.): year integer-valued vector: the year numbers week integer-valued vector: the week numbers wday integer-valued vector: the wday numbers (0–7, either 0 or 7 can be used for Sunday) first.wday wday number of the first day of each week owning.wday if the owning weekday of a week falls in a particular year, the entire week is assigned to that year error.on.wrap if TRUE, throws errors when week numbers do not fall in the specified years; when FALSE, wraps them around into other years

Value

the corresponding dates

`yearWeekWdayVecsToDate` *Like* `yearWeekWdayDFToDate`*, but with each column of* `ywwd` *provided as separate parameters.*

Description

Like `yearWeekWdayDFToDate`, but with each column of `ywwd` provided as separate parameters.

Usage

```
yearWeekWdayVecsToDate(year, week, wday, first.wday, owning.wday,
  error.on.wrap = TRUE)
```

Arguments

`year`	integer-valued vector: the year numbers
`week`	integer-valued vector: the week numbers
`wday`	integer-valued vector: the wday numbers (0–7, either 0 or 7 can be used for Sunday)
`first.wday`	wday number of the first day of each week
`owning.wday`	if the owning weekday of a week falls in a particular year, the entire week is assigned to that year
`error.on.wrap`	if TRUE, throws errors when week numbers do not fall in the specified years; when FALSE, wraps them around into other years

Value

the corresponding dates

Index