Package 'epiforecast.cpp14funs'

September 11, 2019

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>

Recommends epiforecast

Imports stats,
utils,
lubridate,
splines,
glmgen,
glmnet,
httr,
Rcpp,
rlist,
Hmisc,
weights,
hexbin,
plotrix,
tibble,
dplyr,
pipeR,
matrixStats,
quantreg,
lpSolve,
parallel,
R.utils,
RCurl,
pbmcapply

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 measurements and the current time.

License GPL-2

RoxygenNote 6.1.1

Collate 'RcppExports.R'
'checks.R'
'curve-fits.R'
'eb_dists.R'
'interface.R'
'eb.R'

LinkingTo Rcpp

SystemRequirements C++14

Suggests testthat

R topics documented:

`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

---

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

Description

Function for making forecasts with the empirical Bayes method.

Usage

```
eb.sim(full.dat, baseline = 0, time.of.forecast = NULL,
  max.n.sims = 1000L, fit.obj = NULL,
  control.list = get_eb_control_list())
```

Arguments

max.n.sims	maximum number of (weighted) simulated curves to produce. Defaults to 1000.
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 sim object — a list with two components:

ys: a numeric matrix, typically with multiple columns; each column is a different possible trajectory for the current season, with NA's in the input for the current season 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 some models, the non-NA values will remain unchanged).

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

Author(s)

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

Examples

```
library("epiforecast")
## National-level ILINet weighted %ILI data for recent seasons, excluding 2009 pandemic:
area.name = "nat"
full.dat = fetchEpidataFullDat("fluview", area.name, "wili",
                               min.points.in.season = 52L,
                               first.week.of.season = 31L,
                               cache.file.prefix=sprintf("fluview_%s_fetch", area.name))
full.dat <- full.dat[names(full.dat)!="S2009"]
## Sample from conditional curve distribution estimate using the above data
## and CDC's 2015 national %wILI onset threshold baseline of 2.1:
sim = eb.sim(full.dat, baseline=2.1, max.n.sims=100)
print(sim)
plot(sim, type="lineplot")
```

`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, max.n.sims = 2000L,
  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, bias_peaktime_weight = NA_real_,
  bias_peaktime_mean = NA_real_, bias_peaktime_sd = NA_real_,
  bias_peakheight_weight = NA_real_, bias_peakheight_mean = NA_real_,
  bias_peakheight_sd = NA_real_, inactive.seasons = NULL,
  n.out = 53L, ii.match.mask = NULL, max.match.length = NULL,
  n.unpinned.observations = 0L, model = "Empirical Bayes")
```

Arguments

`parent`

`max.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 unde-

sirable, 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.

`bias_peaktime_weight`
is the coefficient of a "peak time bias" 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 `NA`, which disables this feature. When it is enabled, the peak time bias term is the log-likelihood of the underlying peak time (before adding noise) of a curve, according to a Gaussian distribution with mean `bias_peaktime_mean` and standard deviation `bias_peaktime_sd`, multiplied by `bias_peaktime_weight`.

`bias_peaktime_mean`
the mean of the peak time bias Gaussian distribution

`bias_peaktime_sd`
the standard deviation of the peak time bias distribution

`bias_peakheight_weight`
: is the coefficient of a "peak height bias" 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 `NA`, which disables this feature. When it is enabled, the peak height bias term is the log-likelihood of the underlying peak height (before adding noise) of a curve, according to a Gaussian distribution with mean `bias_peakheight_mean` and standard deviation `bias_peakheight_sd`, multiplied by `bias_peakheight_weight`.

`bias_peakheight_mean`
: the mean of the peak height bias Gaussian distribution

`bias_peakheight_sd`
: the standard deviation of the peak height bias distribution

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

`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.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)
```

`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
```

Index