Package: HIDDA.forecasting (via r-universe)

December 7, 2024

Title Forecasting Based on Surveillance Data

Version 1.1.2

Date 2023-11-29

Description The Handbook of Infectious Disease Data Analysis (``HIDDA'') contains a chapter on ``Forecasting Based on Surveillance Data''. The R package 'HIDDA.forecasting' provides the data and code to reproduce results from the two applications described in that chapter (see the corresponding vignettes): Univariate forecasting of Swiss ILI counts using 'forecast', 'glarma', 'surveillance' and 'prophet', and an age-stratified analysis of norovirus gastroenteritis in Berlin using the multivariate time-series model implemented in surveillance::hhh4().

URL https://hidda.github.io/forecasting/, https://github.com/HIDDA/forecasting

BugReports https://github.com/HIDDA/forecasting/issues

Depends R (>= 3.2.0)

Imports grDevices, graphics, stats, utils, zoo, scoringRules (>= 0.9.4)

Suggests knitr, rmarkdown, ggplot2 (>= 3.0.0), dplyr, lattice, fanplot, surveillance (>= 1.16.0), hhh4contacts, forecast, prophet (>= 0.3), glarma, MASS

License GPL (>= 2)

LazyData TRUE

VignetteBuilder knitr, rmarkdown

Roxygen list(markdown = TRUE, old_usage = TRUE)

RoxygenNote 7.2.3

Repository https://ee-lib.r-universe.dev

RemoteUrl https://github.com/HIDDA/forecasting

RemoteRef HEAD

RemoteSha 74ca2570c8fcf17314c5bca5e8a3369b3aed8c07

Contents

`CHILI` *Swiss Surveillance Data on Influenza Like Illness, 2000-2016*

Description

The `CHILI` dataset is a time series of the weekly number of ILI cases in Switzerland from 2000 to 2016, estimated from the Swiss Sentinella Reporting System.

Usage

```
data("CHILI")
```

Format

a univariate time series of class `zoo`, where the time index is of class `Date` and always refers to the Tuesday of the notification week

Source

The Swiss ILI data has been received on 19 January 2017 by courtesy of:

Swiss Federal Office of Public Health
Public Health Directorate
Communicable Diseases Division
3003 Bern
SWITZERLAND

Examples

```
summary(CHILI)
plot(CHILI)
```

dhhh4sims hhh4-*Based Forecast Distributions*

Description

The function dhhh4sims constructs a (non-vectorized) probability mass function from the result of surveillance::simulate.hhh4() (and the corresponding model), as a function of the time point within the simulation period. The distribution at each time point is obtained as a mixture of negative binomial (or Poisson) distributions based on the samples from the previous time point.

Usage

```
dhhh4sims(sims, model)
```

Arguments

sims	a "hhh4sims" object from surveillance::simulate.hhh4().
model	the "hhh4" object underlying sims.

Value

a function(x, tp = 1, log = FALSE), which takes a vector of model$nUnit counts and calculates the (log-)probability of observing these counts (given the model) at the tp'th time point of the simulation period (index or character string matching rownames(sims)).

Author(s)

Sebastian Meyer

See Also

logs_hhh4sims() where this function is used.

Examples

```
library("surveillance")
CHILI.sts <- sts(observed = CHILI,
                 epoch = as.integer(index(CHILI)), epochAsDate = TRUE)

## fit a simple hhh4 model
(f1 <- addSeason2formula(~ 1, period = 365.2425))
fit <- hhh4(
    stsObj = CHILI.sts,
    control = list(ar = list(f = f1), end = list(f = f1), family = "NegBin1")
)

## simulate the last four weeks (only 200 runs, for speed)
sims <- simulate(fit, nsim = 200, seed = 1, subset = 884:nrow(CHILI.sts),
                 y.start = observed(CHILI.sts)[883,])
```

```
if (requireNamespace("fanplot")) {
    plot(sims, "fan", fan.args = list(ln = c(5,95)/100),
         observed.args = list(pch = 19), means.args = list(type = "b"))
}

## derive the weekly forecast distributions
dfun <- dhhh4sims(sims, fit)
dfun(4000, tp = 1)
dfun(4000, tp = 4)
curve(sapply(x, dfun, tp = 4), 0, 30000, type = "h",
      main = "4-weeks-ahead forecast",
      xlab = "No. infected", ylab = "Probability")

## compare the forecast distributions with the simulated counts
par(mfrow = n2mfrow(nrow(sims)))
for (tp in 1:nrow(sims)) {
    MASS::truehist(sims[tp,,], xlab = "counts", ylab = "Probability")
    curve(sapply(x, dfun, tp = tp), add = TRUE, lwd = 2)
}
```

---

dnbmix *Simulation-Based Forecast Distributions*

Description

The function dnbmix() constructs a (vectorized) probability mass function from a matrix of (simulated) means and corresponding size parameters, as a function of the time point (row of means) within the simulation period. The distribution at each time point is obtained as a mixture of negative binomial (or Poisson) distributions.

Usage

```
dnbmix(means, size = NULL)
```

Arguments

means	a n.ahead x n.sim matrix of means.
size	the dispersion parameter of the dnbinom() distribution or NULL (Poisson forecasts). Can also be time-varying (of length n.ahead).

Value

a function(x, tp = 1, log = FALSE), which takes a vector of counts x and calculates the (log-)probabilities of observing each of these numbers at the tp'th time point of the simulation period (indexing the rows of means).

Author(s)

Sebastian Meyer

See Also

`logs_nbmix()` where this function is used.

Examples

```
## a GLARMA example
library("glarma")
y <- as.vector(CHILI)

## fit a simple NegBin-GLARMA model
X <- t(sapply(2*pi*seq_along(y)/52.1775,
              function (x) c(sin = sin(x), cos = cos(x))))
X <- cbind(intercept = 1, X)
fit <- glarma(y = y[1:883], X = X[1:883,], type = "NegBin", phiLags = 1)

## simulate the last four weeks (only 500 runs, for speed)
set.seed(1)
means <- replicate(500, {
    forecast(fit, n.ahead = 4, newdata = X[884:887,], newoffset = rep(0,4))$mu
})

## derive the weekly forecast distributions
dfun <- dnbmix(means, coef(fit, type = "NB"))
dfun(4000, tp = 1)
dfun(4000, tp = 4)
curve(dfun(x, tp = 4), 0, 30000, type = "h",
      main = "4-weeks-ahead forecast",
      xlab = "No. infected", ylab = "Probability")
```

`logs_hhh4sims` *Simulation-Based Logarithmic Score Using* `dhhh4sims`

Description

The function `logs_hhh4sims` computes the logarithmic score of the forecast distributions based on a `surveillance::hhh4()` `model` and simulations (`sims`) thereof. The forecast distributions are obtained via `dhhh4sims()` as sequential mixtures of negative binomial (or Poisson) distributions, which is different from the kernel density estimation approach employed in `scores_sample()`.

Usage

```
logs_hhh4sims(observed = NULL, sims, model)
```

Arguments

observed	a vector or matrix of observed counts during the simulation period. By default (NULL), this is taken from attr(sims, "stsObserved").
sims	a "hhh4sims" object from surveillance::simulate.hhh4().
model	the surveillance::hhh4() fit underlying sims.

Value

a vector or matrix of log-scores for the observed counts.

Author(s)

Sebastian Meyer

See Also

scores_sample() for an alternative approach of calculating the logarithmic score from simulation-based forecasts

---

logs_nbmix *Simulation-Based Logarithmic Score Via* dnbmix

Description

The function logs_nbmix computes the logarithmic score of forecasts based on mixtures of negative binomial (or Poisson) distributions via dnbmix(). This is different from the kernel density estimation approach available via scores_sample().

Usage

```
logs_nbmix(observed, means, size)
```

Arguments

observed	a vector of observed counts during the simulation period.
means	a n.ahead x n.sim matrix of means.
size	the dispersion parameter of the dnbinom() distribution or NULL (Poisson forecasts). Can also be time-varying (of length n.ahead).

Value

a vector of log-scores for the observed counts.

Author(s)

Sebastian Meyer

See Also

scores_sample() for an alternative approach of calculating the logarithmic score from simulation-based forecasts

---

osaplot *Plot (One-Step-Ahead) Forecasts with Scores*

Description

This function produces a fan chart of sequential (one-step-ahead) forecasts with dots for the observed values, using surveillance::fanplot(), which itself wraps fanplot::fan(). A matplot() of score values at each time point is added below ("slicing").

Usage

```
osaplot(quantiles, probs, means, observed, scores, start = 1,
  xlab = "Time", fan.args = list(), means.args = list(),
  observed.args = list(), key.args = list(), ..., scores.args = list(),
  legend.args = list(), heights = c(0.6, 0.4))
```

Arguments

quantiles	a time x probs matrix of forecast quantiles at each time point.
probs	numeric vector of probabilities with values between 0 and 1.
means	(optional) numeric vector of point forecasts at each time point.
observed	(optional) numeric vector of observed values.
scores	(optional) numeric vector (or matrix) of associated scores.
start	time index (x-coordinate) of the first prediction.
xlab	x-axis label.
fan.args	a list of graphical parameters for the fanplot::fan(), e.g., to employ a different colorRampPalette() as fan.col, or to enable contour lines via ln.
means.args	a list of graphical parameters for lines() to modify the plotting style of the point predictions.
observed.args	a list of graphical parameters for lines() to modify the plotting style of the observed values.
key.args	if a list, a color key (in fanplot::fan()'s "boxfan"-style) is added to the fan chart. The list may include positioning parameters start (the x-position) and ylim (the y-range of the color key), space to modify the width of the color key, and rlab to modify the labels. An alternative way of labeling the quantiles is via the argument ln in fan.args.
...	further arguments are passed to plot.default().
scores.args	a list of graphical parameters for matplot() to modify the style of the scores subplot at the bottom.
legend.args	if a list (of parameters for legend()) and ncol(scores) > 1, a legend is added to the scores subplot.
heights	numeric vector of length 2 specifying the relative height of the two subplots.

Author(s)

Sebastian Meyer

`scores_lnorm` *Proper Scoring Rules for Log-Normal Forecasts*

Description

This is a simple wrapper around functions from the **scoringRules** package for predictions with a LN(`meanlog`, `sdlog`) distribution. The function is vectorized and preserves the dimension of the input.

Usage

```
scores_lnorm(x, meanlog, sdlog, which = c("dss", "logs"))
```

Arguments

`x`	the observed counts.
`meanlog`, `sdlog`	parameters of the log-normal distribution, i.e., mean and standard deviation of the distribution on the log scale.
`which`	a character vector specifying which scoring rules to apply. The Dawid-Sebastiani score (`"dss"`) and the logarithmic score (`"logs"`) are available and both computed by default.

Value

scores for the predictions of the observations in `x` (maintaining their dimensions).

`scores_lnorm_discrete` *Proper Scoring Rules for* Discretized *Log-Normal Forecasts*

Description

Compute scores for discretized log-normal forecasts. The function is vectorized and preserves the dimension of the input.

Usage

```
scores_lnorm_discrete(x, meanlog, sdlog, which = c("dss", "logs"))
```

Arguments

x	the observed counts.
meanlog, sdlog	parameters of the log-normal distribution, i.e., mean and standard deviation of the distribution on the log scale.
which	a character vector specifying which scoring rules to apply. The Dawid-Sebastiani score ("dss") and the logarithmic score ("logs") are available and both computed by default.

Value

scores for the predictions of the observations in x (maintaining their dimensions).

---

scores_sample *Proper Scoring Rules based on Simulations*

Description

This is a simple wrapper around functions from the **scoringRules** package to calculate scoring rules from simulation-based forecasts. Calculation of the logarithmic score involves kernel density estimation, see scoringRules::logs_sample(). The function is vectorized and preserves the dimension of the input.

Usage

```
scores_sample(x, sims, which = c("dss", "logs"))
```

Arguments

x	a vector of observed counts.
sims	a matrix of simulated counts with as many rows as length(x).
which	a character vector specifying which scoring rules to apply. The Dawid-Sebastiani score ("dss") and the logarithmic score ("logs") are available and both computed by default.

Value

scores for the predictions of the observations in x (maintaining their dimensions).

`update.Arima` *Refit an ARIMA Model on a Subset of the Time Series*

Description

There seems to be no function in package **forecast** (as of version 8.2) to re-estimate an ARIMA model on a subset of the original time series. This `update` method does exactly that.

Usage

```
## S3 method for class 'Arima'
update(object, subset, ...)
```

Arguments

`object`	an object of class `"Arima"`, e.g., from `forecast::auto.arima()`.
`subset`	an integer vector selecting part of the original time series (and external regressors).
`...`	further arguments to be passed to `arima()`.

Value

the updated model.

Author(s)

Sebastian Meyer

Index