Package 'profoc'

September 15, 2021

Type Package

Title Probabilistic Forecast Combination Using CRPS Learning

Version 0.8.4

Date 2021-09-15

Description Combine probabilistic forecasts using CRPS learning algorithms proposed in Berrisch, Ziel (2021) <arXiv:2102.00968>. The package implements multiple online learning algorithms like Bernstein online aggregation; see Wintenberger (2014) <arXiv:1404.1356>. Quantile regression is also implemented for comparison purposes. Model parameters can be tuned automatically with respect to the loss of the forecast combination. Methods like predict(), update(), plot() and print() are available for convenience. This package utilizes the optim C++ library for numeric optimization <https://github.com/kthohr/optim>.

License GPL (>= 3)

Encoding UTF-8

Depends R (>= 3.0.2)

Imports Rcpp (>= 1.0.5), Matrix

LinkingTo Rcpp, RcppArmadillo, RcppProgress, splines2 (>= 0.4.4)

URL https://profoc.berrisch.biz/, https://github.com/BerriJ/profoc

BugReports https://github.com/BerriJ/profoc/issues

RoxygenNote 7.1.1

Suggests testthat (>= 3.0.0), gamlss.dist, ggplot2

Config/testthat/edition 3

NeedsCompilation yes

Author Jonathan Berrisch [cre] (<https://orcid.org/0000-0002-4944-9074>), Florian Ziel [aut] (<https://orcid.org/0000-0002-2974-2660>)

Maintainer Jonathan Berrisch <Jonathan@Berrisch.biz>

Repository CRAN

Date/Publication 2021-09-15 14:30:07 UTC

R topics documented:

`profoc-package` *Package Info*

Description

Use multiple online-aggregation algorithms to combine probabilistic forecasts using CRPS Learning as described in Berrisch, Ziel: "CRPS Learning", 2021. The primary function of this package is called profoc.

Details

Index of help topics:

```
autoplot              Create a complete ggplot appropriate to a
                      particular data type
autoplot.batch        Autoplot method for batch models
autoplot.online       Autoplot method for online models
batch                 Probabilistic Forecast Combination - Batch
online                Probabilistic Forecast Combination - Online
oracle                Probabilistic Forecast Combination - Oracle
plot.batch            Plot method for batch models
plot.online           Plot method for online models
predict.online        Predict method for online models
print.batch           Print method for batch models
print.online          Print method for online models
profoc-package        Package Info
update.online         Update method for online models
```

Author(s)

NA

Maintainer: Jonathan Berrisch mailto:Jonathan@Berrisch.biz

Co-Author: Florian Ziel

References

Berrisch, Ziel: "CRPS Learning", 2021

See Also

Source Code: https://github.com/BerriJ/profoc

BugReports: https://github.com/BerriJ/profoc/issues

`autoplot` *Create a complete ggplot appropriate to a particular data type*

Description

'autoplot()' uses ggplot2 to draw a particular plot for an object of a particular class in a single command. This defines the S3 generic that other classes and packages can extend.

Usage

```
autoplot(object, ...)
```

Arguments

`object`	an object, whose class will determine the behaviour of autoplot
`...`	other arguments passed to specific methods

Value

a ggplot object

See Also

[autolayer()], [ggplot()] and [fortify()]

autoplot.batch *Autoplot method for batch models*

Description

Plots the most recent weights in each quantile using ggplot2.

Usage

```
## S3 method for class 'batch'
autoplot(object, ...)
```

Arguments

object	Object of class inheriting from 'batch'
...	further arguments are ignored

autoplot.online *Autoplot method for online models*

Description

Plots the most recent weights in each quantile using ggplot2.

Usage

```
## S3 method for class 'online'
autoplot(object, ...)
```

Arguments

object	Object of class inheriting from 'online'
...	further arguments are ignored

batch *Probabilistic Forecast Combination - Batch*

Description

Returns predictions and weights calculated by sequential numeric optimization. The optimization is done stepwise, always calculating a one-step-ahead forecast.

Usage

```
batch(
  y,
  experts,
  tau = 1:dim(experts)[2]/(dim(experts)[2] + 1),
  affine = FALSE,
  positive = FALSE,
  intercept = FALSE,
  debias = TRUE,
  lead_time = 0,
  initial_window = 30,
  rolling_window = initial_window,
  loss_function = "quantile",
  loss_parameter = 1,
  qw_crps = FALSE,
  basis_knot_distance = 1/(dim(experts)[2] + 1),
  basis_knot_distance_power = 1,
  basis_deg = 1,
  forget = 0,
  soft_threshold = -Inf,
  hard_threshold = -Inf,
  fixed_share = 0,
  p_smooth_lambda = -Inf,
  p_smooth_knot_distance = basis_knot_distance,
  p_smooth_knot_distance_power = basis_knot_distance_power,
  p_smooth_deg = basis_deg,
  p_smooth_ndiff = 1.5,
  parametergrid_max_combinations = 100,
  parametergrid = NULL,
  forget_past_performance = 0,
  allow_quantile_crossing = FALSE,
  trace = TRUE
)
```

Arguments

y	A numeric matrix of realizations. In probabilistic settings a matrix of dimension Tx1. In multivariate setting a TxP matrix can be used. In the latter case, each

slice of the expert's array gets evaluated using the corresponding column of the y matrix.

`experts` An array of predictions with dimension (Observations, Quantiles, Experts).

`tau` A numeric vector of probabilities.

`affine` Defines whether weights are summing to 1 or now. Defaults to FALSE.

`positive` Defines if a positivity constraint is applied to the weights. Defaults to FALSE.

`intercept` Determines if an intercept is added, defaults to FALSE. If true, a new first expert is added, always predicting 1.

`debias` Defines whether the intercepts weight is constrained or not. If TRUE (the default), the intercept weight is unconstrained. Only affects the results if affine and or positive is set to TRUE. If FALSE, the intercept is treated as an expert.

`lead_time` offset for expert forecasts. Defaults to 0, which means that experts forecast t+1 at t. Setting this to h means experts predictions refer to t+1+h at time t. The weight updates delay accordingly.

`initial_window` Defines the size of the initial estimation window.

`rolling_window` Defines the size of the rolling window. Defaults to the value of initial_window. Set it to the number of observations to receive an expanding window.

`loss_function` Either "quantile", "expectile" or "percentage".

`loss_parameter` Optional parameter scaling the power of the loss function.

`qw_crps` Decides wether the sum of quantile scores (FALSE) or the quantile weighted crps (TRUE) should be minimized. Defaults to FALSE. Which corresponds to Berrisch & Ziel (2021)

`basis_knot_distance`
determines the distance of the knots in the probability basis. Defaults to 1 / (dim(experts)[2] + 1).

`basis_knot_distance_power`
Parameter which defines the symmetry of the basis reducing the probability space. Defaults to 1, which corresponds to equidistant knots. Values less than 1 create more knots in the center, while values above 1 concentrate more knots in the tails.

`basis_deg` Degree of the basis reducing the probability space. Defaults to 1.

`forget` Adds an exponential forgetting to the optimization. Past observations will get less influence on the optimization. Defaults to 0, which corresponds to no forgetting.

`soft_threshold` If specified, the following soft threshold will be applied to the weights: w = sgn(w)*max(abs(w)-t,0) where t is the soft_threshold parameter. Defaults to -inf, which means that no threshold will be applied. If all expert weights are thresholded to 0, a weight of 1 will be assigned to the expert with the highest weights prior to thresholding. Thus soft_threshold = 1 leads to the 'follow the leader' strategy if method is set to "ewa".

`hard_threshold` If specified, the following hard thresholding will be applied to the weights: w = w*(abs(w)>t) where t is the threshold_hard parameter. Defaults to -inf, which means that no threshold will be applied. If all expert weights are thresholded to

0, a weight of 1 will be assigned to the expert with the highest weight prior to thresholding. Thus hard_threshold = 1 leads to the 'follow the leader' strategy if method is set to "ewa".

`fixed_share` Amount of fixed share to be added to the weights. Defaults to 0. 1 leads to uniform weights.

`p_smooth_lambda`
Penalization parameter used in the smoothing step. -Inf causes the smoothing step to be skipped (default).

`p_smooth_knot_distance`
determines the distance of the knots. Defaults to the value of basis_knot_distance. Corresponds to the grid steps when knot_distance_power = 1 (the default).

`p_smooth_knot_distance_power`
Parameter which defines the symmetry of the P-Spline basis. Takes the value of basis_knot_distance_power if unspecified.

`p_smooth_deg` Degree of the B-Spine basis functions. Defaults to the value of basis_deg.

`p_smooth_ndiff` Degree of the differencing operator in the smoothing equation. 1.5 (default) leads to shrinkage towards a constant. Can take values from 1 to 2. If a value in between is used, a weighted sum of the first and second differentiation matrix is calculated.

`parametergrid_max_combinations`
Integer specifying the maximum number of parameter combinations that should be considered. If the number of possible combinations exceeds this threshold, the maximum allowed number is randomly sampled. Defaults to 100.

`parametergrid` User supplied grid of parameters. Can be used if not all combinations of the input vectors should be considered. Must be a matrix with 13 columns (online) or 12 columns batch with the following order: basis_knot_distance, basis_knot_distance_power, basis_deg, forget_regret, soft_threshold, hard_threshold, fixed_share, p_smooth_lambda, p_smooth_knot_distance, p_smooth_knot_distance_power, p_smooth_deg, p_smooth_ndiff, gamma.

`forget_past_performance`
Share of past performance not to be considered, resp. to be forgotten in every iteration of the algorithm when selecting the best parameter combination. Defaults to 0.

`allow_quantile_crossing`
Shall quantile crossing be allowed? Defaults to false, which means that predictions are sorted in ascending order.

`trace` Print a progress bar to the console? Defaults to TRUE.

Value

Returns weights and corresponding predictions. It is possible to impose a convexity constraint to the weights by setting affine and positive to TRUE.

Examples

```
## Not run:
T <- 50 # Observations
```

```
N <- 2 # Experts
P <- 9 # Quantiles
prob_grid <- 1:P / (P + 1)

y <- rnorm(n = T) # Realized
experts <- array(dim = c(T, P, N)) # Predictions
for (t in 1:T) {
    experts[t, , 1] <- qnorm(prob_grid, mean = -1, sd = 1)
    experts[t, , 2] <- qnorm(prob_grid, mean = 3, sd = sqrt(4))
}

model <- batch(
    y = matrix(y),
    experts = experts,
    p_smooth_lambda = 10
)

print(model)
plot(model)
autoplot(model)

## End(Not run)
```

`online` *Probabilistic Forecast Combination - Online*

Description

Returns predictions and weights calculated by online-learning algorithms using CRPS Learning.

Usage

```
online(
  y,
  experts,
  tau = 1:dim(experts)[2]/(dim(experts)[2] + 1),
  lead_time = 0,
  loss_function = "quantile",
  loss_parameter = 1,
  loss_gradient = TRUE,
  method = "bewa",
  basis_knot_distance = 1/(dim(experts)[2] + 1),
  basis_knot_distance_power = 1,
  basis_deg = 1,
  forget_regret = 0,
  soft_threshold = -Inf,
  hard_threshold = -Inf,
  fixed_share = 0,
  p_smooth_lambda = -Inf,
```

```
  p_smooth_knot_distance = basis_knot_distance,
  p_smooth_knot_distance_power = basis_knot_distance_power,
  p_smooth_deg = basis_deg,
  p_smooth_ndiff = 1.5,
  gamma = 1,
  parametergrid_max_combinations = 100,
  parametergrid = NULL,
  forget_past_performance = 0,
  allow_quantile_crossing = FALSE,
  init_weights = NULL,
  loss = NULL,
  regret = NULL,
  trace = TRUE
)
```

Arguments

y	A numeric matrix of realizations. In probabilistic settings a matrix of dimension Tx1. In multivariate setting a TxP matrix can be used. In the latter case, each slice of the expert's array gets evaluated using the corresponding column of the y matrix.
experts	An array of predictions with dimension (Observations, Quantiles, Experts).
tau	A numeric vector of probabilities.
lead_time	offset for expert forecasts. Defaults to 0, which means that experts forecast t+1 at t. Setting this to h means experts predictions refer to t+1+h at time t. The weight updates delay accordingly.
loss_function	Either "quantile", "expectile" or "percentage".
loss_parameter	Optional parameter scaling the power of the loss function.
loss_gradient	Determines if a linearized version of the loss is used.
method	One of "boa", "bewa", "ml_poly" or "ewa". Where "bewa" refers to a mixture of boa and ewa, including the second order refinement of boa, but updating weights with the simple exponential weighting.
basis_knot_distance	determines the distance of the knots in the probability basis. Defaults to 1 / (dim(experts)[2] + 1), which means that one knot is created for every quantile. Takes a vector with values >0 where values > .5 correspond to constant weights (only one single knot).
basis_knot_distance_power	Parameter which defines the symmetry of the basis reducing the probability space. Defaults to 1, which corresponds to equidistant knots. Values less than 1 create more knots in the center, while values above 1 concentrate more knots in the tails.
basis_deg	Degree of the basis reducing the probability space. Defaults to 1.
forget_regret	Share of past regret not to be considered, resp. to be forgotten in every iteration of the algorithm. Defaults to 0.

`soft_threshold` If specified, the following soft threshold will be applied to the weights: w = sgn(w)*max(abs(w)-t,0) where t is the soft_threshold parameter. Defaults to -inf, which means that no threshold will be applied. If all expert weights are thresholded to 0, a weight of 1 will be assigned to the expert with the highest weights prior to thresholding. Thus soft_threshold = 1 leads to the 'follow the leader' strategy if method is set to "ewa".

`hard_threshold` If specified, the following hard thresholding will be applied to the weights: w = w*(abs(w)>t) where t is the threshold_hard parameter. Defaults to -inf, which means that no threshold will be applied. If all expert weights are thresholded to 0, a weight of 1 will be assigned to the expert with the highest weight prior to thresholding. Thus hard_threshold = 1 leads to the 'follow the leader' strategy if method is set to "ewa".

`fixed_share` Amount of fixed share to be added to the weights. Defaults to 0. 1 leads to uniform weights.

`p_smooth_lambda`
Penalization parameter used in the smoothing step. -Inf causes the smoothing step to be skipped (default).

`p_smooth_knot_distance`
determines the distance of the knots. Defaults to the value of basis_knot_distance. Corresponds to the grid steps when knot_distance_power = 1 (the default).

`p_smooth_knot_distance_power`
Parameter which defines the symmetry of the P-Spline basis. Takes the value of basis_knot_distance_power if unspecified.

`p_smooth_deg` Degree of the B-Spine basis functions. Defaults to the value of basis_deg.

`p_smooth_ndiff` Degree of the differencing operator in the smoothing equation. 1.5 (default) leads to shrinkage towards a constant. Can take values from 1 to 2. If a value in between is used, a weighted sum of the first and second differentiation matrix is calculated.

`gamma` Scaling parameter for the learning rate.

`parametergrid_max_combinations`
Integer specifying the maximum number of parameter combinations that should be considered. If the number of possible combinations exceeds this threshold, the maximum allowed number is randomly sampled. Defaults to 100.

`parametergrid` User supplied grid of parameters. Can be used if not all combinations of the input vectors should be considered. Must be a matrix with 13 columns (online) or 12 columns batch with the following order: basis_knot_distance, basis_knot_distance_power, basis_deg, forget_regret, soft_threshold, hard_threshold, fixed_share, p_smooth_lambda, p_smooth_knot_distance, p_smooth_knot_distance_power, p_smooth_deg, p_smooth_ndiff, gamma.

`forget_past_performance`
Share of past performance not to be considered, resp. to be forgotten in every iteration of the algorithm when selecting the best parameter combination. Defaults to 0.

`allow_quantile_crossing`
Shall quantile crossing be allowed? Defaults to false, which means that predictions are sorted in ascending order.

init_weights	Matrix of dimension 1xK or PxK used as starting weights. 1xK represents the constant solution with equal weights over all P, whereas specifying a PxK matrix allows different starting weights for each P.
loss	User specified loss array. Can also be a list with elements "loss_array" and "share", share mixes the provided loss with the loss calculated by profoc. 1 means, only the provided loss will be used. share can also be vector of shares to consider.
regret	User specified regret array. If specific, the regret will not be calculated by profoc. Can also be a list with elements "regret_array" and "share", share mixes the provided regret with the regret calculated by profoc. 1 means, only the provided regret will be used. share can also be vector of shares to consider.
trace	Print a progress bar to the console? Defaults to TRUE.

Details

online can tune various parameters automatically based on the past loss. For this, lambda, forget, fixed_share, gamma, ndiff, deg and knot_distance can be specified as numeric vectors containing parameters to consider. online will automatically try all possible combinations of values provide.

Value

Returns weights and corresponding predictions.

Examples

```
## Not run:
T <- 50 # Observations
N <- 2 # Experts
P <- 9 # Quantiles
prob_grid <- 1:P / (P + 1)

y <- rnorm(n = T) # Realized
experts <- array(dim = c(T, P, N)) # Predictions
for (t in 1:T) {
    experts[t, , 1] <- qnorm(prob_grid, mean = -1, sd = 1)
    experts[t, , 2] <- qnorm(prob_grid, mean = 3, sd = sqrt(4))
}

model <- online(
    y = matrix(y),
    experts = experts,
    p_smooth_lambda = 10
)

print(model)
plot(model)
autoplot(model)

new_y <- matrix(rnorm(1)) # Realized
new_experts <- experts[T, , , drop = FALSE]
```

```
# Update will update the model object, no need for new assignment
update(model, new_y = new_y, new_experts = new_experts)

# Use predict to combine new_experts, model$predictions will be extended
predict(model, new_experts = new_experts)

## End(Not run)
```

`oracle` *Probabilistic Forecast Combination - Oracle*

Description

Returns predictions and weights calculated by numeric optimization. The optimization is done in hindsight. This means all observations are used.

Usage

```
oracle(y, experts, tau, affine = FALSE,
positive = FALSE, intercept = FALSE, debias = TRUE,
loss_function = "quantile", loss_parameter = 1, forget = 0)
```

Arguments

`y`	A numeric matrix of realizations. In probabilistic settings a matrix of dimension Tx1. In multivariate setting a TxP matrix can be used. In the latter case, each slice of the expert's array gets evaluated using the corresponding column of the y matrix.
`experts`	An array of predictions with dimension (Observations, Quantiles, Experts).
`tau`	A numeric vector of probabilities.
`affine`	Defines whether weights are summing to 1 or now. Defaults to FALSE.
`positive`	Defines if a positivity constraint is applied to the weights. Defaults to FALSE.
`intercept`	Determines if an intercept is added, defaults to FALSE. If true, a new first expert is added, always predicting 1.
`debias`	Defines whether the intercepts weight is constrained or not. If TRUE (the default), the intercept weight is unconstrained. Only affects the results if affine and or positive is set to TRUE. If FALSE, the intercept is treated as an expert.
`loss_function`	Either "quantile", "expectile" or "percentage".
`loss_parameter`	Optional parameter scaling the power of the loss function.
`forget`	Adds an exponential forgetting to the optimization. Past observations will get less influence on the optimization. Defaults to 0, which corresponds to no forgetting.

Value

Returns weights and corresponding predictions. It is possible to calculate the best convex combination of weights by setting affine and positive to TRUE.

Examples

```
## Not run:
T <- 50 # Observations
N <- 2 # Experts
P <- 9 # Quantiles
prob_grid <- 1:P / (P + 1)

y <- rnorm(n = T) # Realized
experts <- array(dim = c(T, P, N)) # Predictions
for (t in 1:T) {
    experts[t, , 1] <- qnorm(prob_grid, mean = -1, sd = 1)
    experts[t, , 2] <- qnorm(prob_grid, mean = 3, sd = sqrt(4))
}

model <- oracle(
    y = matrix(y),
    experts = experts
)

## End(Not run)
```

---

plot.batch *Plot method for batch models*

Description

Plots the most recent weights in each quantile.

Usage

```
## S3 method for class 'batch'
plot(x, ...)
```

Arguments

x	Object of class inheriting from 'batch'
...	further arguments are ignored

plot.online *Plot method for online models*

Description

Plots the most recent weights in each quantile.

Usage

```
## S3 method for class 'online'
plot(x, ...)
```

Arguments

x	Object of class inheriting from 'online'
...	further arguments are ignored

predict.online *Predict method for online models*

Description

Calculates predictions based on new expert advice. This does not update weights. If new observations are available use update instead. The latter updates and computes predictions.

Usage

```
## S3 method for class 'online'
predict(object, new_experts, ...)
```

Arguments

object	Object of class inheriting from 'online'
new_experts	new expert predictions
...	further arguments are ignored

Value

predict.online produces an updated model object.

print.batch *Print method for batch models*

Description

Prints the average loss of all and the forecast combination.

Usage

```
## S3 method for class 'batch'
print(x, ...)
```

Arguments

x	Object of class inheriting from 'batch'
...	further arguments are ignored

print.online *Print method for online models*

Description

Prints the average loss of all experts and the forecast combination.

Usage

```
## S3 method for class 'online'
print(x, ...)
```

Arguments

x	Object of class inheriting from 'online'
...	further arguments are ignored

update.online *Update method for online models*

Description

Continues learning using new observations and new expert advice.

Usage

```
## S3 method for class 'online'
update(object, new_y, new_experts = as.numeric(c()), ...)
```

Arguments

object	Object of class inheriting from 'online'
new_y	new observations
new_experts	new expert predictions. This must be left unspecified if the model already contains the expert predictions corresponding to new_y.
...	further arguments are ignored

Value

update.online produces an updated model object.

Index