Package ‘opera’

March 18, 2020

Type Package

Title Online Prediction by Expert Aggregation

Version 1.1

Date 2020-03-11

Author Pierre Gaillard [cre, aut],
Yannig Goude [aut]

Maintainer Pierre Gaillard `<pierre@gaillard.me>`

Copyright EDF R&D 2012-2015

Description Misc methods to form online predictions, for regression-oriented time-series, by combining a finite set of forecasts provided by the user. See Cesa-Bianchi and Lugosi (2006) <doi:10.1017/CBO9780511546921> for an overview.

License LGPL

URL `http://pierre.gaillard.me/opera.html`

BugReports `https://github.com/dralliag/opera/issues`

Depends R (>= 3.1.0)

Imports

Suggests quantreg, quadprog, RColorBrewer, testthat, splines, caret, mgcv, survival, knitr, gbm, htmltools

LazyData true

VignetteBuilder knitr

RoxygenNote 7.1.0

NeedsCompilation no

Repository CRAN

Date/Publication 2020-03-18 22:40:03 UTC

R topics documented:

`opera-package` *Online Prediction by ExpeRt Aggregation*

Description

The package `opera` performs, for regression-oriented time-series, predictions by combining a finite set of forecasts provided by the user. More formally, it considers a sequence of observations `Y` (such as electricity consumption, or any bounded time series) to be predicted step by step. At each time instance `t`, a finite set of experts (basicly some based forecasters) provide predictions `x` of the next observation in `y`. This package proposes several adaptive and robust methods to combine the expert forecasts based on their past performance.

Author(s)

Pierre Gaillard <pierre@gaillard.me>

References

Prediction, Learning, and Games. N. Cesa-Bianchi and G. Lugosi.

Forecasting the electricity consumption by aggregating specialized experts; a review of sequential aggregation of specialized experts, with an application to Slovakian an French contry-wide one-day-ahead (half-)hourly predictions, Machine Learning, in press, 2012. Marie Devaine, Pierre Gaillard, Yannig Goude, and Gilles Stoltz

Contributions to online robust aggregation: work on the approximation error and on probabilistic forecasting. Pierre Gaillard. PhD Thesis, University Paris-Sud, 2015.

Examples

```
#'
library('opera')  # load the package
set.seed(1)
```

```
# Example: find the best one week ahead forecasting strategy (weekly data)
# packages
library(mgcv)

# import data
data(electric_load)
idx_data_test <- 620:nrow(electric_load)
data_train <- electric_load[-idx_data_test, ]
data_test <- electric_load[idx_data_test, ]

# Build the expert forecasts
# ##########################

# 1) A generalized additive model
gam.fit <- gam(Load ~ s(IPI) + s(Temp) + s(Time, k=3) +
                s(Load1) + as.factor(NumWeek), data = data_train)
gam.forecast <- predict(gam.fit, newdata = data_test)

# 2) An online autoregressive model on the residuals of a medium term model

# Medium term model to remove trend and seasonality (using generalized additive model)
detrend.fit <- gam(Load ~ s(Time,k=3) + s(NumWeek) + s(Temp) + s(IPI), data = data_train)
electric_load$Trend <- c(predict(detrend.fit), predict(detrend.fit,newdata = data_test))
electric_load$Load.detrend <- electric_load$Load - electric_load$Trend

# Residual analysis
ar.forecast <- numeric(length(idx_data_test))
for (i in seq(idx_data_test)) {
 ar.fit <- ar(electric_load$Load.detrend[1:(idx_data_test[i] - 1)])
 ar.forecast[i] <- as.numeric(predict(ar.fit)$pred) + electric_load$Trend[idx_data_test[i]]
}

# Aggregation of experts
###########################

X <- cbind(gam.forecast, ar.forecast)
colnames(X) <- c('gam', 'ar')
Y <- data_test$Load

matplot(cbind(Y, X), type = 'l', col = 1:6, ylab = 'Weekly load', xlab = 'Week')


# How good are the expert? Look at the oracles
oracle.convex <- oracle(Y = Y, experts = X, loss.type = 'square', model = 'convex')
plot(oracle.convex)
oracle.convex

# Is a single expert the best over time ? Are there breaks ?
oracle.shift <- oracle(Y = Y, experts = X, loss.type = 'percentage', model = 'shifting')
plot(oracle.shift)
oracle.shift

# Online aggregation of the experts with BOA
```

```
##############################################

# Initialize the aggregation rule
m0.BOA <- mixture(model = 'BOA', loss.type = 'square')

# Perform online prediction using BOA There are 3 equivalent possibilities 1)
# start with an empty model and update the model sequentially
m1.BOA <- m0.BOA
for (i in 1:length(Y)) {
 m1.BOA <- predict(m1.BOA, newexperts = X[i, ], newY = Y[i])
}

# 2) perform online prediction directly from the empty model
m2.BOA <- predict(m0.BOA, newexpert = X, newY = Y, online = TRUE)

# 3) perform the online aggregation directly
m3.BOA <- mixture(Y = Y, experts = X, model = 'BOA', loss.type = 'square')

# These predictions are equivalent:
identical(m1.BOA, m2.BOA)  # TRUE
identical(m1.BOA, m3.BOA)  # TRUE

# Display the results
summary(m3.BOA)
plot(m1.BOA)
```

electric_load *Electricity forecasting data set*

Description

Electricity forecasting data set provided by EDF R&D. It contains weekly measurements of the total electricity consumption in France from 1996 to 2009, together with several covariates, including temperature, industrial production indices (source: INSEE) and calendar information.

Usage

```
data(electric_load)
```

Format

An object of class `data.frame` with 731 rows and 11 columns.

Examples

```
 data(electric_load)
 # a few graphs to display the data
 attach(electric_load)
 plot(Load, type = 'l')
 plot(Temp, Load, pch = 16, cex = 0.5)
```

```
plot(NumWeek, Load, pch = 16, cex = 0.5)
plot(Load, Load1, pch = 16, cex = 0.5)
acf(Load, lag.max = 20)
detach(electric_load)
```

`loss` *Errors suffered by a sequence of predictions*

Description

The function `loss` computes the sequence of instantaneous losses suffered by the predictions in `x` to predict the observation in `y`.

Usage

```
loss(x, y, loss.type = "square")
```

Arguments

`x`	A vector of length `T` containing the sequence of prediction to be evaluated.
`y`	A vector of length `T` that contains the observations to be predicted.
`loss.type`	A string or a list with a component 'name' specifying the loss function considered to evaluate the performance. It can be 'square', 'absolute', 'percentage', or 'pinball'. In the case of the pinball loss, the quantile can be provided by assigning to loss.type a list of two elements: **name** A string defining the name of the loss function (i.e., 'pinball') **tau** A number in `[0,1]` defining the quantile to be predicted. The default value is 0.5 to predict the median.

Value

A vector of length `T` containing the sequence of instantaneous losses suffered by the prediction `x`.

Author(s)

Pierre Gaillard <pierre@gaillard.me>

`mixture` *Compute an aggregation rule*

Description

The function `mixture` builds an aggregation rule chosen by the user. It can then be used to predict new observations Y sequentially. If observations `Y` and expert advice `experts` are provided, `mixture` is trained by predicting the observations in `Y` sequentially with the help of the expert advice in `experts`. At each time instance $t = 1, 2, \ldots, T$, the mixture forms a prediction of `Y[t,]` by assigning a weight to each expert and by combining the expert advice.

Usage

```
mixture(
  Y = NULL,
  experts = NULL,
  model = "MLpol",
  loss.type = "square",
  loss.gradient = TRUE,
  coefficients = "Uniform",
  awake = NULL,
  parameters = list()
)

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

## S3 method for class 'mixture'
summary(object, ...)
```

Arguments

`Y` A matrix with T rows and d columns. Each row `Y[t,]` contains a d-dimensional observation to be predicted sequentially.

`experts` An array of dimension `c(T,d,K)`, where `T` is the length of the data-set, `d` the dimension of the observations, and `K` is the number of experts. It contains the expert forecasts. Each vector `experts[t,,k]` corresponds to the d-dimensional prediction of `Y[t,]` proposed by expert k at time $t = 1, \ldots, T$. In the case of real prediction (i.e., $d = 1$), `experts` is a matrix with `T` rows and `K` columns.

`model` A character string specifying the aggregation rule to use. Currently available aggregation rules are:

'EWA' Exponentially weighted average aggregation rule. A positive learning rate **eta** can be chosen by the user. The bigger it is the faster the aggregation rule will learn from observations and experts performances. However, too high values lead to unstable weight vectors and thus unstable predictions. If it is not specified, the learning rate is calibrated online. A finite grid of potential learning rates to be optimized online can be specified with **grid.eta**.

'FS' Fixed-share aggregation rule. As for `ewa`, a learning rate **eta** can be chosen by the user or calibrated online. The main difference with `ewa` aggregation rule rely in the mixing rate **alpha**$\in[0,1]$ wich considers at each instance a small probability `alpha` to have a rupture in the sequence and that the best expert may change. Fixed-share aggregation rule can thus compete with the best sequence of experts that can change a few times (see `oracle`), while `ewa` can only compete with the best fixed expert. The mixing rate **alpha** is either chosen by the user either calibrated online. Finite grids of learning rates and mixing rates to be optimized can be specified with parameters **grid.eta** and **grid.alpha**.

'Ridge' Ridge regression. It minimizes at each instance a penalized criterion. It forms at each instance linear combination of the experts' forecasts and can assign negative weights that not necessarily sum to one. It is useful if the experts are biased or correlated. It cannot be used with specialized experts. A positive regularization coefficient **lambda** can either be chosen by the user or calibrated online. A finite grid of coefficient to be optimized can be specified with a parameter **grid.lambda**.

'MLpol' Polynomial Potential aggregation rule with different learning rates for each expert. The learning rates are calibrated using theoretical values. There are similar aggregation rules like 'BOA' (Bernstein online Aggregation see [Wintenberger, 2014], 'MLewa', and 'MLprod' (see [Gaillard, Erven, and Stoltz, 2014])

'OGD' Online Gradient descent (see Zinkevich, 2003). The optimization is performed with a time-varying learning rate. At time step $t \geq 1$, the learning rate is chosen to be $t^{-\alpha}$, where α is provided by alpha in the parameters argument. The algorithm may or not perform a projection step into the simplex space (non-negative weights that sum to one) according to the value of the parameter 'simplex' provided by the user.

`loss.type` A string or a list with a component 'name' specifying the loss function considered to evaluate the performance. It can be 'square', 'absolute', 'percentage', or 'pinball'. In the case of the pinball loss, the quantile can be provided by assigning to loss.type a list of two elements:

name A string defining the name of the loss function (i.e., 'pinball')

tau A number in `[0,1]` defining the quantile to be predicted. The default value is 0.5 to predict the median.

'Ridge' aggregation rule is restricted to square loss.

`loss.gradient` A boolean. If TRUE (default) the aggregation rule will not be directly applied to the loss function at hand but to a gradient version of it. The aggregation rule is then similar to gradient descent aggregation rule.

`coefficients` A probability vector of length K containing the prior weights of the experts (not possible for 'MLpol'). The weights must be non-negative and sum to 1.

`awake` A matrix specifying the activation coefficients of the experts. Its entries lie in `[0,1]`. Possible if some experts are specialists and do not always form and

	suggest prediction. If the expert number `k` at instance `t` does not form any prediction of observation `Y_t`, we can put `awake[t,k]=0` so that the mixture does not consider expert `k` in the mixture to predict `Y_t`.
`parameters`	A list that contains optional parameters for the aggregation rule. If no parameters are provided, the aggregation rule is fully calibrated online. Possible parameters are:

- **eta** A positive number defining the learning rate. Possible if model is either 'EWA' or 'FS'
- **grid.eta** A vector of positive numbers defining potential learning rates for 'EWA' of 'FS'. The learning rate is then calibrated by sequentially optimizing the parameter in the grid. The grid may be extended online if needed by the aggregation rule.
- **gamma** A positive number defining the exponential step of extension of grid.eta when it is needed. The default value is 2.
- **alpha** A number in [0,1]. If the model is 'FS', it defines the mixing rate. If the model is 'OGD', it defines the order of the learning rate: $\eta_t = t^{-\alpha}$.
- **grid.alpha** A vector of numbers in [0,1] defining potential mixing rates for 'FS' to be optimized online. The grid is fixed over time. The default value is `[0.0001,0.001,0.01,0.1]`.
- **lambda** A positive number defining the smoothing parameter of 'Ridge' aggregation rule.
- **grid.lambda** Similar to `grid.eta` for the parameter `lambda`.
- **simplex** A boolean that specifies if 'OGD' does a project on the simplex. In other words, if TRUE (default) the online gradient descent will be under the constraint that the weights sum to 1 and are non-negative. If FALSE, 'OGD' performs an online gradient descent on K dimensional real space. without any projection step.
- **averaged** A boolean (default is FALSE). If TRUE the coefficients and the weights returned (and used to form the predictions) are averaged over the past. It leads to more stability on the time evolution of the weights but needs more regularity assumption on the underlying process genearting the data (i.i.d. for instance).

`x`	An object of class mixture
`...`	Additional parameters
`object`	An object of class mixture

Value

An object of class mixture that can be used to perform new predictions. It contains the parameters `model`, `loss.type`, `loss.gradient`, `experts`, `Y`, `awake`, and the fields

`coefficients`	A vector of coefficients assigned to each expert to perform the next prediction.
`weights`	A matrix of dimension `c(T,K)`, with `T` the number of instances to be predicted and `K` the number of experts. Each row contains the convex combination to form the predictions

`prediction`	A matrix with `T` rows and `d` columns that contains the predictions outputted by the aggregation rule.
`loss`	The average loss (as stated by parameter `loss.type`) suffered by the aggregation rule.
`parameters`	The learning parameters chosen by the aggregation rule or by the user.
`training`	A list that contains useful temporary information of the aggregation rule to be updated and to perform predictions.

Methods (by class)

- `mixture`: `print`
- `mixture`: `summary`

Author(s)

Pierre Gaillard <pierre@gaillard.me>

See Also

See `opera-package` and opera-vignette for a brief example about how to use the package.

Examples

```
#'
library('opera')  # load the package
set.seed(1)

# Example: find the best one week ahead forecasting strategy (weekly data)
# packages
library(mgcv)

# import data
data(electric_load)
idx_data_test <- 620:nrow(electric_load)
data_train <- electric_load[-idx_data_test, ]
data_test <- electric_load[idx_data_test, ]

# Build the expert forecasts
# ############################

# 1) A generalized additive model
gam.fit <- gam(Load ~ s(IPI) + s(Temp) + s(Time, k=3) +
                s(Load1) + as.factor(NumWeek), data = data_train)
gam.forecast <- predict(gam.fit, newdata = data_test)

# 2) An online autoregressive model on the residuals of a medium term model

# Medium term model to remove trend and seasonality (using generalized additive model)
detrend.fit <- gam(Load ~ s(Time,k=3) + s(NumWeek) + s(Temp) + s(IPI), data = data_train)
electric_load$Trend <- c(predict(detrend.fit), predict(detrend.fit,newdata = data_test))
```

```
electric_load$Load.detrend <- electric_load$Load - electric_load$Trend

# Residual analysis
ar.forecast <- numeric(length(idx_data_test))
for (i in seq(idx_data_test)) {
 ar.fit <- ar(electric_load$Load.detrend[1:(idx_data_test[i] - 1)])
 ar.forecast[i] <- as.numeric(predict(ar.fit)$pred) + electric_load$Trend[idx_data_test[i]]
}

# Aggregation of experts
###########################

X <- cbind(gam.forecast, ar.forecast)
colnames(X) <- c('gam', 'ar')
Y <- data_test$Load

matplot(cbind(Y, X), type = 'l', col = 1:6, ylab = 'Weekly load', xlab = 'Week')


# How good are the expert? Look at the oracles
oracle.convex <- oracle(Y = Y, experts = X, loss.type = 'square', model = 'convex')
plot(oracle.convex)
oracle.convex

# Is a single expert the best over time ? Are there breaks ?
oracle.shift <- oracle(Y = Y, experts = X, loss.type = 'percentage', model = 'shifting')
plot(oracle.shift)
oracle.shift

# Online aggregation of the experts with BOA
#############################################

# Initialize the aggregation rule
m0.BOA <- mixture(model = 'BOA', loss.type = 'square')

# Perform online prediction using BOA There are 3 equivalent possibilities 1)
# start with an empty model and update the model sequentially
m1.BOA <- m0.BOA
for (i in 1:length(Y)) {
 m1.BOA <- predict(m1.BOA, newexperts = X[i, ], newY = Y[i])
}

# 2) perform online prediction directly from the empty model
m2.BOA <- predict(m0.BOA, newexpert = X, newY = Y, online = TRUE)

# 3) perform the online aggregation directly
m3.BOA <- mixture(Y = Y, experts = X, model = 'BOA', loss.type = 'square')

# These predictions are equivalent:
identical(m1.BOA, m2.BOA)  # TRUE
identical(m1.BOA, m3.BOA)  # TRUE

# Display the results
```

```
summary(m3.BOA)
plot(m1.BOA)
```

`oracle` *Compute oracle predictions*

Description

The function `oracle` performs a strategie that cannot be defined online (in contrast to mixture). It requires in advance the knowledge of the whole data set `Y` and the expert advice to be well defined. Examples of oracles are the best fixed expert, the best fixed convex combination rule, the best linear combination rule, or the best expert that can shift a few times.

Usage

```
oracle(
  Y,
  experts,
  model = "convex",
  loss.type = "square",
  awake = NULL,
  lambda = NULL,
  niter = NULL,
  ...
)

## S3 method for class 'oracle'
plot(x, sort = TRUE, col = NULL, ...)
```

Arguments

`Y`	A vector containing the observations to be predicted.
`experts`	A matrix containing the experts forecasts. Each column corresponds to the predictions proposed by an expert to predict `Y`. It has as many columns as there are experts.
`model`	A character string specifying the oracle to use or a list with a component `name` specifying the oracle and any additional parameter needed. Currently available oracles are: **'expert'** The best fixed (constant over time) expert oracle. **'convex'** The best fixed convex combination (vector of non-negative weights that sum to 1) **'linear'** The best fixed linear combination of expert **'shifting'** It computes for all number \$m\$ of stwitches the sequence of experts with at most \$m\$ shifts that would have performed the best to predict the sequence of observations in `Y`.

`loss.type`	A string or a list with a component 'name' specifying the loss function considered to evaluate the performance. It can be 'square', 'absolute', 'percentage', or 'pinball'. In the case of the pinball loss, the quantile can be provided by assigning to loss.type a list of two elements: **name** A string defining the name of the loss function (i.e., 'pinball') **tau** A number in `[0,1]` defining the quantile to be predicted. The default value is 0.5 to predict the median.
`awake`	A matrix specifying the activation coefficients of the experts. Its entries lie in `[0,1]`. Possible if some experts are specialists and do not always form and suggest prediction. If the expert number `k` at instance `t` does not form any prediction of observation `Y_t`, we can put `awake[t,k]=0` so that the mixture does not consider expert `k` in the mixture to predict `Y_t`. Remark that to compute the best expert oracle, the performance of unactive (or partially active) experts is computed by using the prediction of the uniform average of active experts.
`lambda`	A positive number used by the 'linear' oracle only. A possible \$L_2\$ regularization parameter for computing the linear oracle (if the design matrix is not identifiable)
`niter`	A positive integer for 'convex' and 'linear' oracles if direct computation of the oracle is not implemented. It defines the number of optimization steps to perform in order to approximate the oracle (default value is 3).
`...`	Additional parameters that are passed to `optim` function is order to perform convex optimization (see parameter `niter`).
`x`	An object of class `oracle`.
`sort`	if set to TRUE (default), it sorts the experts by performance before the plots.
`col`	colors

Value

An object of class 'oracle' that contains:

`loss`	The average loss suffered by the oracle. For the 'shifting' oracle, it is a vector of length `T` where `T` is the number of instance to be predicted (i.e., the length of the sequence `Y`). The value of \$loss(m)\$ is the loss (determined by the parameter `loss.type`) suffered by the best sequence of expert with at most \$m-1\$ shifts.
`coefficients`	Not for the 'shifting' oracle. A vector containing the best weight vector corresponding to the oracle.
`prediction`	Not for the 'shifting' oracle. A vector containing the predictions of the oracle.
`rmse`	If loss.type is the square loss (default) only. The root mean square error (i.e., it is the square root of `loss`.

Methods (by class)

- `oracle`: `plot`. It has one optional arguments.

Author(s)

Pierre Gaillard <pierre@gaillard.me>

`plot.mixture` *Plot an object of class mixture*

Description

provides different diagnostic plots for an aggregation procedure.

Usage

```
## S3 method for class 'mixture'
plot(x, pause = FALSE, col = NULL, ...)
```

Arguments

`x`	an object of class mixture. If awake is provided (i.e., some experts are unactive), their residuals and cumulative losses are computed by using the predictions of the mixture.
`pause`	if set to TRUE (default) displays the plots separately, otherwise on a single page
`col`	the color to use to represent each experts, if set to NULL (default) use `RRColorBrewer::brewer.pal(...`
`...`	additional plotting parameters

Value

plots representing: plot of weights of each expert in function of time, boxplots of these weights, cumulative loss $L_T = \sum_{t=1}^T l_{i,t}$ of each expert in function of time, cumulative residuals $\sum_{t=1}^T (y_t - f_{i,t})$ of each expert's forecast in function of time, average loss suffered by the experts and the contribution of each expert to the aggregation $p_{i,t}f_{i,t}$ in function of time.

Author(s)

Pierre Gaillard <pierre@gaillard.me>

Yannig Goude <yannig.goude@edf.fr>

See Also

See `opera-package` and opera-vignette for a brief example about how to use the package.

predict.mixture *Predict method for Mixture models*

Description

Performs sequential predictions and updates of a mixture object based on new observations and expert advice.

Usage

```
## S3 method for class 'mixture'
predict(
  object,
  newexperts = NULL,
  newY = NULL,
  awake = NULL,
  online = TRUE,
  type = c("model", "response", "weights", "all"),
  ...
)
```

Arguments

`object`	Object of class inheriting from 'mixture'
`newexperts`	An optional matrix in which to look for expert advice with which predict. If omitted, the past predictions of the object are returned and the object is not updated.
`newY`	An optional matrix with d columns (or vector if $d = 1$) of observations to be predicted. If provided, it should have the same number of rows as the number of rows of `newexperts`. If omitted, the object (i.e, the aggregation rule) is not updated.
`awake`	An optional array specifying the activation coefficients of the experts. It must have the same dimension as experts. Its entries lie in `[0,1]`. Possible if some experts are specialists and do not always form and suggest prediction. If the expert number `k` at instance `t` does not form any prediction of observation `Y_t`, we can put `awake[t,k]=0` so that the mixture does not consider expert `k` in the mixture to predict `Y_t`.
`online`	A boolean determining if the observations in newY are predicted sequentially (by updating the object step by step) or not. If FALSE, the observations are predicting using the object (without using any past information in newY). If TRUE, newY and newexperts should not be null.
`type`	Type of prediction. It can be **model** return the updated version of object (using newY and newexperts). **response** return the forecasts. If type is 'model', forecasts can also be obtained from the last values of object$prediction.

	weights return the weights assigned to the expert advice to produce the forecasts. If type is 'model', forecasts can also be obtained from the last rows of object$weights.
	all return a list containing 'model', 'response', and 'weights'.
`...`	further arguments are ignored

Value

`predict.mixture` produces a matrix of predictions (type = 'response'), an updated object (type = 'model'), or a matrix of weights (type = 'weights').

`seriesToBlock` *Convert a 1-dimensional series to blocks*

Description

The functions `seriesToBlock` and `blockToSeries` convert 1-dimensional series into series of higher dimension. For instance, suppose you have a time-series that consists of $T = 100$ days of $d = 24$ hours. The function seriesToBlock converts the time-series X of $Td = 2400$ observations into a matrix of size `c(T=100,d =24)`, where each line corresponds to a specific day. This function is usefull if you need to perform the prediction day by day, instead of hour by hour. The function can also be used to convert a matrix of expert prediction of dimension `c(dT,K)` where K is the number of experts, into an array of dimension `c(T,d,K)`. The new arrays of observations and of expert predictions can be given to the aggregation rule procedure to perform `d`-dimensional predictions (i.e., day predictions).

Usage

```
seriesToBlock(X, d)

blockToSeries(X)
```

Arguments

`X`	An array or a vector to be converted.
`d`	A positive integer defining the block size.

Details

The function blockToSeries performs the inverse operation.

Index