Package 'EQRN'

November 21, 2025

Type Package

Title Extreme Quantile Regression Neural Networks for Risk Forecasting

Version 0.1.2

Description This framework enables forecasting and extrapolating measures of conditional risk (e.g. of extreme or unprecedented events), including quantiles and exceedance probabilities, using extreme value statistics and flexible neural network architectures.
It allows for capturing complex multivariate dependencies, including dependencies between observations, such as sequential dependence (time-series).
The methodology was introduced in Pasche and Engelke (2024) <doi:10.1214/24-AOAS1907> (also available in preprint: Pasche and Engelke (2022) <doi:10.48550/arXiv.2208.07590>).

License GPL (>= 3)

Encoding UTF-8

Imports coro (>= 1.0.2), doFuture, evd, foreach, future, ismev, magrittr, stats, torch, utils

RoxygenNote 7.3.2

URL https://github.com/opasche/EQRN, https://opasche.github.io/EQRN/

BugReports https://github.com/opasche/EQRN/issues

NeedsCompilation no

Author Olivier C. Pasche [aut, cre, cph] (ORCID: <https://orcid.org/0000-0002-1202-9199>)

Maintainer Olivier C. Pasche <olivier_pasche@alumni.epfl.ch>

Repository CRAN

Date/Publication 2025-11-21 13:10:02 UTC

Contents

`backend_is_installed` *Check if Torch Backend Libraries are Installed*

Description

Check if Torch Backend Libraries are Installed

Usage

```
backend_is_installed(...)
```

Arguments

`...`	Optional parameters passed to `torch::torch_is_installed()`.

Value

Boolean indicating whether the LibTorch and LibLantern backend libraries are installed.

Examples

```
if(backend_is_installed()){
  cat("torch's LibTorch and LibLantern backend libraries are installed!\n")
}
```

`check_directory` *Check directory existence*

Description

Checks if the desired directory exists. If not, the desired directory is created.

Usage

```
check_directory(dir_name, recursive = TRUE, no_warning = FALSE)
```

Arguments

`dir_name`	Path to the desired directory, as a string.
`recursive`	Should elements of the path other than the last be created? If `TRUE`, behaves like the Unix command `mkdir -p`.
`no_warning`	Whether to cancel the warning issued if a directory is created (bool).

Value

No return value.

Examples

```
check_directory("./some_folder/my_new_folder")
```

`compute_EQRN_GPDLoss` *Generalized Pareto likelihood loss of a EQRN_iid predictor*

Description

Generalized Pareto likelihood loss of a EQRN_iid predictor

Usage

```
compute_EQRN_GPDLoss(
  fit_eqrn,
  X,
  y,
  intermediate_quantiles = NULL,
  interm_lvl = fit_eqrn$interm_lvl,
  device = default_device()
)
```

Arguments

`fit_eqrn`	Fitted `"EQRN_iid"` object.
`X`	Matrix of covariates.
`y`	Response variable vector.
`intermediate_quantiles`	Vector of intermediate conditional quantiles at level `fit_eqrn$interm_lvl`.
`interm_lvl`	Optional, checks that `interm_lvl == fit_eqrn$interm_lvl`.
`device`	(optional) A `torch::torch_device()`. Defaults to `default_device()`.

Value

Negative GPD log likelihood of the conditional EQRN predicted parameters over the response exceedances over the intermediate quantiles.

---

compute_EQRN_seq_GPDLoss

Generalized Pareto likelihood loss of a EQRN_seq predictor

Description

Generalized Pareto likelihood loss of a EQRN_seq predictor

Usage

```
compute_EQRN_seq_GPDLoss(
  fit_eqrn,
  X,
  Y,
  intermediate_quantiles = NULL,
  interm_lvl = fit_eqrn$interm_lvl,
  seq_len = fit_eqrn$seq_len,
  device = default_device()
)
```

Arguments

`fit_eqrn`	Fitted `"EQRN_seq"` object.
`X`	Matrix of covariates.
`Y`	Response variable vector corresponding to the rows of `X`.
`intermediate_quantiles`	Vector of intermediate conditional quantiles at level `fit_eqrn$interm_lvl`.
`interm_lvl`	Optional, checks that `interm_lvl == fit_eqrn$interm_lvl`.
`seq_len`	Data sequence length (i.e. number of past observations) used to predict each response quantile. By default, the training `fit_eqrn$seq_len` is used.
`device`	(optional) A `torch::torch_device()`. Defaults to `default_device()`.

Value

Negative GPD log likelihood of the conditional EQRN predicted parameters over the response exceedances over the intermediate quantiles.

`default_device` *Default torch device*

Description

Default torch device

Usage

```
default_device()
```

Value

Returns `torch::torch_device("cuda")` if `torch::cuda_is_available()`, or `torch::torch_device("cpu")` otherwise.

Examples

```
if(backend_is_installed()){
  device <- default_device()
  print(device)
}
```

end_doFuture_strategy *End the currently set doFuture strategy*

Description

Resets the default strategy using `future::plan("default")`.

Usage

```
end_doFuture_strategy()
```

Value

No return value.

Examples

```
`%fun%` <- set_doFuture_strategy("multisession", n_workers=3)
# perform foreach::foreach loop using the %fun% operator
end_doFuture_strategy()
```

EQRN_excess_probability *Tail excess probability prediction using an EQRN_iid object*

Description

Tail excess probability prediction using an EQRN_iid object

Usage

```
EQRN_excess_probability(
  val,
  fit_eqrn,
  X,
  intermediate_quantiles,
  interm_lvl = fit_eqrn$interm_lvl,
  body_proba = "default",
  proba_type = c("excess", "cdf"),
  device = default_device()
)
```

Arguments

`val`	Quantile value(s) used to estimate the conditional excess probability or cdf.
`fit_eqrn`	Fitted `"EQRN_iid"` object.
`X`	Matrix of covariates to predict the corresponding response's conditional excess probabilities.
`intermediate_quantiles`	Vector of intermediate conditional quantiles at level `fit_eqrn$interm_lvl`.
`interm_lvl`	Optional, checks that `interm_lvl == fit_eqrn$interm_lvl`.
`body_proba`	Value to use when the predicted conditional probability is below `interm_lvl` (in which case it cannot be precisely assessed by the model). If `"default"` is given (the default), `paste0(">",1-interm_lvl)` is used if `proba_type=="excess"`, and `paste0("<",interm_lvl)` is used if `proba_type=="cdf"`.
`proba_type`	Whether to return the `"excess"` probability over `val` (default) or the `"cdf"` at `val`.
`device`	(optional) A `torch::torch_device()`. Defaults to `default_device()`.

Value

Vector of probabilities (and possibly a few `body_proba` values if `val` is not large enough) of length `nrow(X)`.

`EQRN_excess_probability_seq` *Tail excess probability prediction using an EQRN_seq object*

Description

Tail excess probability prediction using an EQRN_seq object

Usage

```
EQRN_excess_probability_seq(
  val,
  fit_eqrn,
  X,
  Y,
  intermediate_quantiles,
  interm_lvl = fit_eqrn$interm_lvl,
  crop_predictions = FALSE,
  body_proba = "default",
  proba_type = c("excess", "cdf"),
  seq_len = fit_eqrn$seq_len,
  device = default_device()
)
```

Arguments

val	Quantile value(s) used to estimate the conditional excess probability or cdf.
fit_eqrn	Fitted "EQRN_seq" object.
X	Matrix of covariates to predict the response's conditional excess probabilities.
Y	Response variable vector corresponding to the rows of X.
intermediate_quantiles	Vector of intermediate conditional quantiles at level fit_eqrn$interm_lvl.
interm_lvl	Optional, checks that interm_lvl == fit_eqrn$interm_lvl.
crop_predictions	Whether to crop out the fist seq_len observations (which are NA) from the returned vector
body_proba	Value to use when the predicted conditional probability is below interm_lvl (in which case it cannot be precisely assessed by the model). If "default" is given (the default), paste0(">",1-interm_lvl) is used if proba_type=="excess", and paste0("<",interm_lvl) is used if proba_type=="cdf".
proba_type	Whether to return the "excess" probability over val (default) or the "cdf" at val.
seq_len	Data sequence length (i.e. number of past observations) used to predict each response quantile. By default, the training fit_eqrn$seq_len is used.
device	(optional) A torch::torch_device(). Defaults to default_device().

Value

Vector of probabilities (and possibly a few body_proba values if val is not large enough) of length nrow(X) (or nrow(X)-seq_len if crop_predictions).

EQRN_fit *EQRN fit function for independent data*

Description

Use the EQRN_fit_restart() wrapper instead, with data_type="iid", for better stability using fitting restart.

Usage

```
EQRN_fit(
  X,
  y,
  intermediate_quantiles,
  interm_lvl,
  shape_fixed = FALSE,
  net_structure = c(5, 3, 3),
  hidden_fct = torch::nnf_sigmoid,
```

```
  p_drop = 0,
  intermediate_q_feature = TRUE,
  learning_rate = 1e-04,
  L2_pen = 0,
  shape_penalty = 0,
  scale_features = TRUE,
  n_epochs = 500,
  batch_size = 256,
  X_valid = NULL,
  y_valid = NULL,
  quant_valid = NULL,
  lr_decay = 1,
  patience_decay = n_epochs,
  min_lr = 0,
  patience_stop = n_epochs,
  tol = 1e-06,
  orthogonal_gpd = TRUE,
  patience_lag = 1,
  optim_met = "adam",
  seed = NULL,
  verbose = 2,
  device = default_device()
)
```

Arguments

X	Matrix of covariates, for training.
y	Response variable vector to model the extreme conditional quantile of, for training.
intermediate_quantiles	Vector of intermediate conditional quantiles at level interm_lvl.
interm_lvl	Probability level for the intermediate quantiles intermediate_quantiles.
shape_fixed	Whether the shape estimate depends on the covariates or not (bool).
net_structure	Vector of integers whose length determines the number of layers in the neural network and entries the number of neurons in each corresponding successive layer. If hidden_fct=="SSNN", should instead be a named list with "scale" and "shape" vectors for the two respective sub-networks. Can also be a torch::nn_module network with correct input and output dimensions, which overrides the hidden_fct, shape_fixed and p_drop arguments.
hidden_fct	Activation function for the hidden layers. Can be either a callable function (preferably from the torch library), or one of the the strings "SNN", "SSNN" for self normalizing networks (with common or separated networks for the scale and shape estimates, respectively). In the latter cases, shape_fixed has no effect.
p_drop	Probability parameter for dropout before each hidden layer for regularization during training. alpha-dropout is used with SNNs.

`intermediate_q_feature`
: Whether to use the `intermediate_quantiles` as an additional covariate, by appending it to the `X` matrix (bool).

`learning_rate`
: Initial learning rate for the optimizer during training of the neural network.

`L2_pen`
: L2 weight penalty parameter for regularization during training.

`shape_penalty`
: Penalty parameter for the shape estimate, to potentially regularize its variation from the fixed prior estimate.

`scale_features`
: Whether to rescale each input covariates to zero mean and unit variance before applying the network (recommended).

`n_epochs`
: Number of training epochs.

`batch_size`
: Batch size used during training.

`X_valid`
: Covariates in a validation set, or `NULL`. Used for monitoring validation loss during training, enabling learning-rate decay and early stopping.

`y_valid`
: Response variable in a validation set, or `NULL`. Used for monitoring validation loss during training, enabling learning-rate decay and early stopping.

`quant_valid`
: Intermediate conditional quantiles at level `interm_lvl` in a validation set, or `NULL`. Used for monitoring validation loss during training, enabling learning-rate decay and early stopping.

`lr_decay`
: Learning rate decay factor.

`patience_decay`
: Number of epochs of non-improving validation loss before a learning-rate decay is performed.

`min_lr`
: Minimum learning rate, under which no more decay is performed.

`patience_stop`
: Number of epochs of non-improving validation loss before early stopping is performed.

`tol`
: Tolerance for stopping training, in case of no significant training loss improvements.

`orthogonal_gpd`
: Whether to use the orthogonal reparametrization of the estimated GPD parameters (recommended).

`patience_lag`
: The validation loss is considered to be non-improving if it is larger than on any of the previous `patience_lag` epochs.

`optim_met`
: DEPRECATED. Optimization algorithm to use during training. `"adam"` is the default.

`seed`
: Integer random seed for reproducibility in network weight initialization.

`verbose`
: Amount of information printed during training (0:nothing, 1:most important, 2:everything).

`device`
: (optional) A `torch::torch_device()`. Defaults to `default_device()`.

Value

An EQRN object of classes `c("EQRN_iid", "EQRN")`, containing the fitted network, as well as all the relevant information for its usage in other functions.

`EQRN_fit_restart` *Wrapper for fitting EQRN with restart for stability*

Description

Wrapper for fitting EQRN with restart for stability

Usage

```
EQRN_fit_restart(
  X,
  y,
  intermediate_quantiles,
  interm_lvl,
  number_fits = 3,
  ...,
  seed = NULL,
  data_type = c("iid", "seq")
)
```

Arguments

`X`	Matrix of covariates, for training.
`y`	Response variable vector to model the extreme conditional quantile of, for training.
`intermediate_quantiles`	Vector of intermediate conditional quantiles at level `interm_lvl`.
`interm_lvl`	Probability level for the intermediate quantiles `intermediate_quantiles`.
`number_fits`	Number of restarts.
`...`	Other parameters given to either `EQRN_fit()` or `EQRN_fit_seq()`, depending on the `data_type`.
`seed`	Integer random seed for reproducibility in network weight initialization.
`data_type`	Type of data dependence, must be one of `"iid"` (for iid observations) or `"seq"` (for sequentially dependent observations).

Value

An EQRN object of classes `c("EQRN_iid", "EQRN")`, if `data_type=="iid"`, or `c("EQRN_seq", "EQRN")`, if 'data_type=="seq", containing the fitted network, as well as all the relevant information for its usage in other functions.

EQRN_fit_seq *EQRN fit function for sequential and time series data*

Description

Use the EQRN_fit_restart() wrapper instead, with data_type="seq", for better stability using fitting restart.

Usage

```
EQRN_fit_seq(
  X,
  y,
  intermediate_quantiles,
  interm_lvl,
  shape_fixed = FALSE,
  hidden_size = 10,
  num_layers = 1,
  rnn_type = c("lstm", "gru"),
  p_drop = 0,
  intermediate_q_feature = TRUE,
  learning_rate = 1e-04,
  L2_pen = 0,
  seq_len = 10,
  shape_penalty = 0,
  scale_features = TRUE,
  n_epochs = 500,
  batch_size = 256,
  X_valid = NULL,
  y_valid = NULL,
  quant_valid = NULL,
  lr_decay = 1,
  patience_decay = n_epochs,
  min_lr = 0,
  patience_stop = n_epochs,
  tol = 1e-05,
  orthogonal_gpd = TRUE,
  patience_lag = 1,
  fold_separation = NULL,
  optim_met = "adam",
  seed = NULL,
  verbose = 2,
  device = default_device()
)
```

Arguments

X	Matrix of covariates, for training. Entries must be in sequential order.

y	Response variable vector to model the extreme conditional quantile of, for training. Entries must be in sequential order.
intermediate_quantiles	Vector of intermediate conditional quantiles at level interm_lvl.
interm_lvl	Probability level for the intermediate quantiles intermediate_quantiles.
shape_fixed	Whether the shape estimate depends on the covariates or not (bool).
hidden_size	Dimension of the hidden latent state variables in the recurrent network.
num_layers	Number of recurrent layers.
rnn_type	Type of recurrent architecture, can be one of "lstm" (default) or "gru".
p_drop	Probability parameter for dropout before each hidden layer for regularization during training.
intermediate_q_feature	Whether to use the intermediate_quantiles as an additional covariate, by appending it to the X matrix (bool).
learning_rate	Initial learning rate for the optimizer during training of the neural network.
L2_pen	L2 weight penalty parameter for regularization during training.
seq_len	Data sequence length (i.e. number of past observations) used during training to predict each response quantile.
shape_penalty	Penalty parameter for the shape estimate, to potentially regularize its variation from the fixed prior estimate.
scale_features	Whether to rescale each input covariates to zero mean and unit covariance before applying the network (recommended).
n_epochs	Number of training epochs.
batch_size	Batch size used during training.
X_valid	Covariates in a validation set, or NULL. Entries must be in sequential order. Used for monitoring validation loss during training, enabling learning-rate decay and early stopping.
y_valid	Response variable in a validation set, or NULL. Entries must be in sequential order. Used for monitoring validation loss during training, enabling learning-rate decay and early stopping.
quant_valid	Intermediate conditional quantiles at level interm_lvl in a validation set, or NULL. Used for monitoring validation loss during training, enabling learning-rate decay and early stopping.
lr_decay	Learning rate decay factor.
patience_decay	Number of epochs of non-improving validation loss before a learning-rate decay is performed.
min_lr	Minimum learning rate, under which no more decay is performed.
patience_stop	Number of epochs of non-improving validation loss before early stopping is performed.
tol	Tolerance for stopping training, in case of no significant training loss improvements.

orthogonal_gpd	Whether to use the orthogonal reparametrization of the estimated GPD parameters (recommended).
patience_lag	The validation loss is considered to be non-improving if it is larger than on any of the previous patience_lag epochs.
fold_separation	Index of fold separation or sequential discontinuity in the data.
optim_met	DEPRECATED. Optimization algorithm to use during training. "adam" is the default.
seed	Integer random seed for reproducibility in network weight initialization.
verbose	Amount of information printed during training (0:nothing, 1:most important, 2:everything).
device	(optional) A torch::torch_device(). Defaults to default_device().

Value

An EQRN object of classes c("EQRN_seq", "EQRN"), containing the fitted network, as well as all the relevant information for its usage in other functions.

EQRN_load *Load an EQRN object from disc*

Description

Loads in memory an "EQRN" object that has previously been saved on disc using EQRN_save().

Usage

```
EQRN_load(path, name = NULL, device = default_device(), ...)
```

Arguments

path	Path to the save location as a string.
name	String name of the save. If NULL (default), assumes the save name has been given implicitly in the path.
device	(optional) A torch::torch_device(). Defaults to default_device().
...	DEPRECATED. Used for back-compatibility.

Value

The loaded "EQRN" model.

EQRN_predict *Predict function for an EQRN_iid fitted object*

Description

Predict function for an EQRN_iid fitted object

Usage

```
EQRN_predict(
  fit_eqrn,
  X,
  prob_lvls_predict,
  intermediate_quantiles,
  interm_lvl = fit_eqrn$interm_lvl,
  device = default_device()
)
```

Arguments

`fit_eqrn`	Fitted `"EQRN_iid"` object.
`X`	Matrix of covariates to predict the corresponding response's conditional quantiles.
`prob_lvls_predict`	Vector of probability levels at which to predict the conditional quantiles.
`intermediate_quantiles`	Vector of intermediate conditional quantiles at level `fit_eqrn$interm_lvl`.
`interm_lvl`	Optional, checks that `interm_lvl == fit_eqrn$interm_lvl`.
`device`	(optional) A `torch::torch_device()`. Defaults to `default_device()`.

Value

Matrix of size `nrow(X)` times `prob_lvls_predict` containing the conditional quantile estimates of the response associated to each covariate observation at each probability level. Simplifies to a vector if `length(prob_lvls_predict)==1`.

EQRN_predict_params *GPD parameters prediction function for an EQRN_iid fitted object*

Description

GPD parameters prediction function for an EQRN_iid fitted object

Usage

```
EQRN_predict_params(
  fit_eqrn,
  X,
  intermediate_quantiles = NULL,
  return_parametrization = c("classical", "orthogonal"),
  interm_lvl = fit_eqrn$interm_lvl,
  device = default_device()
)
```

Arguments

`fit_eqrn`	Fitted `"EQRN_iid"` object.
`X`	Matrix of covariates to predict conditional GPD parameters.
`intermediate_quantiles`	Vector of intermediate conditional quantiles at level `fit_eqrn$interm_lvl`.
`return_parametrization`	Which parametrization to return the parameters in, either `"classical"` or `"orthogonal"`.
`interm_lvl`	Optional, checks that `interm_lvl == fit_eqrn$interm_lvl`.
`device`	(optional) A `torch::torch_device()`. Defaults to `default_device()`.

Value

Named list containing: `"scales"` and `"shapes"` as numerical vectors of length `nrow(X)`.

EQRN_predict_params_seq

GPD parameters prediction function for an EQRN_seq fitted object

---

Description

GPD parameters prediction function for an EQRN_seq fitted object

Usage

```
EQRN_predict_params_seq(
  fit_eqrn,
  X,
  Y,
  intermediate_quantiles = NULL,
  return_parametrization = c("classical", "orthogonal"),
  interm_lvl = fit_eqrn$interm_lvl,
  seq_len = fit_eqrn$seq_len,
  device = default_device()
)
```

Arguments

fit_eqrn	Fitted "EQRN_seq" object.
X	Matrix of covariates to predict conditional GPD parameters.
Y	Response variable vector corresponding to the rows of X.
intermediate_quantiles	Vector of intermediate conditional quantiles at level fit_eqrn$interm_lvl.
return_parametrization	Which parametrization to return the parameters in, either "classical" or "orthogonal".
interm_lvl	Optional, checks that interm_lvl == fit_eqrn$interm_lvl.
seq_len	Data sequence length (i.e. number of past observations) used to predict each response quantile. By default, the training fit_eqrn$seq_len is used.
device	(optional) A torch::torch_device(). Defaults to default_device().

Value

Named list containing: "scales" and "shapes" as numerical vectors of length nrow(X), and the seq_len used.

EQRN_predict_seq *Predict function for an EQRN_seq fitted object*

Description

Predict function for an EQRN_seq fitted object

Usage

```
EQRN_predict_seq(
  fit_eqrn,
  X,
  Y,
  prob_lvls_predict,
  intermediate_quantiles,
  interm_lvl,
  crop_predictions = FALSE,
  seq_len = fit_eqrn$seq_len,
  device = default_device()
)
```

Arguments

fit_eqrn	Fitted "EQRN_seq" object.
X	Matrix of covariates to predict the corresponding response's conditional quantiles.

Y	Response variable vector corresponding to the rows of X.
prob_lvls_predict	Vector of probability levels at which to predict the conditional quantiles.
intermediate_quantiles	Vector of intermediate conditional quantiles at level fit_eqrn$interm_lvl.
interm_lvl	Optional, checks that interm_lvl == fit_eqrn$interm_lvl.
crop_predictions	Whether to crop out the fist seq_len observations (which are NA) from the returned matrix.
seq_len	Data sequence length (i.e. number of past observations) used to predict each response quantile. By default, the training fit_eqrn$seq_len is used.
device	(optional) A torch::torch_device(). Defaults to default_device().

Value

Matrix of size nrow(X) times prob_lvls_predict (or nrow(X)-seq_len times prob_lvls_predict if crop_predictions) containing the conditional quantile estimates of the corresponding response observations at each probability level. Simplifies to a vector if length(prob_lvls_predict)==1.

EQRN_save *Save an EQRN object on disc*

Description

Creates a folder named name and located in path, containing binary save files, so that the given "EQRN" object fit_eqrn can be loaded back in memory from disc using EQRN_load().

Usage

```
EQRN_save(fit_eqrn, path, name = NULL, no_warning = TRUE)
```

Arguments

fit_eqrn	An "EQRN" object
path	Path to save folder as a string.
name	String name of the save.
no_warning	Whether to silence the warning raised if a save folder needed beeing created (bool).

Value

No return value.

`excess_probability` *Excess Probability Predictions*

Description

A generic function (method) for excess probability predictions from various fitted EQR models. The function invokes particular methods which depend on the class of the first argument.

Usage

```
excess_probability(object, ...)
```

Arguments

`object`	A model object for which excess probability prediction is desired.
`...`	additional model-specific arguments affecting the predictions produced. See the corresponding method documentation.

Value

The excess probability estimates from the given EQR model.

`excess_probability.EQRN_iid`
Tail excess probability prediction method using an EQRN_iid object

Description

Tail excess probability prediction method using an EQRN_iid object

Usage

```
## S3 method for class 'EQRN_iid'
excess_probability(object, ...)
```

Arguments

`object`	Fitted `"EQRN_iid"` object.
`...`	Arguments passed on to `EQRN_excess_probability` `val` Quantile value(s) used to estimate the conditional excess probability or cdf. `X` Matrix of covariates to predict the corresponding response's conditional excess probabilities. `intermediate_quantiles` Vector of intermediate conditional quantiles at level `fit_eqrn$interm_lvl`.

`interm_lvl` Optional, checks that `interm_lvl == fit_eqrn$interm_lvl`.

`body_proba` Value to use when the predicted conditional probability is below `interm_lvl` (in which case it cannot be precisely assessed by the model). If `"default"` is given (the default), `paste0(">",1-interm_lvl)` is used if `proba_type=="excess"`, and `paste0("<",interm_lvl)` is used if `proba_type=="cdf"`.

`proba_type` Whether to return the `"excess"` probability over `val` (default) or the `"cdf"` at `val`.

`device` (optional) A `torch::torch_device()`. Defaults to `default_device()`.

Details

See `EQRN_excess_probability()` for more details.

Value

Vector of probabilities (and possibly a few `body_proba` values if `val` is not large enough) of length `nrow(X)`.

`excess_probability.EQRN_seq`

Tail excess probability prediction method using an EQRN_iid object

---

Description

Tail excess probability prediction method using an EQRN_iid object

Usage

```
## S3 method for class 'EQRN_seq'
excess_probability(object, ...)
```

Arguments

`object` Fitted `"EQRN_seq"` object.

`...` Arguments passed on to `EQRN_excess_probability_seq`

`val` Quantile value(s) used to estimate the conditional excess probability or cdf.

`X` Matrix of covariates to predict the response's conditional excess probabilities.

`Y` Response variable vector corresponding to the rows of `X`.

`intermediate_quantiles` Vector of intermediate conditional quantiles at level `fit_eqrn$interm_lvl`.

`interm_lvl` Optional, checks that `interm_lvl == fit_eqrn$interm_lvl`.

`crop_predictions` Whether to crop out the fist `seq_len` observations (which are `NA`) from the returned vector

body_proba Value to use when the predicted conditional probability is below interm_lvl (in which case it cannot be precisely assessed by the model). If "default" is given (the default), paste0(">",1-interm_lvl) is used if proba_type=="excess", and paste0("<",interm_lvl) is used if proba_type=="cdf".

proba_type Whether to return the "excess" probability over val (default) or the "cdf" at val.

seq_len Data sequence length (i.e. number of past observations) used to predict each response quantile. By default, the training fit_eqrn$seq_len is used.

device (optional) A torch::torch_device(). Defaults to default_device().

Details

See EQRN_excess_probability_seq() for more details.

Value

Vector of probabilities (and possibly a few body_proba values if val is not large enough) of length nrow(X) (or nrow(X)-seq_len if crop_predictions).

FC_GPD_net *MLP module for GPD parameter prediction*

Description

A fully-connected network (or multi-layer perception) as a torch::nn_module, designed for generalized Pareto distribution parameter prediction.

Usage

```
FC_GPD_net(
  D_in,
  Hidden_vect = c(5, 5, 5),
  activation = torch::nnf_sigmoid,
  p_drop = 0,
  shape_fixed = FALSE,
  device = EQRN::default_device()
)
```

Arguments

D_in	the input size (i.e. the number of features),
Hidden_vect	a vector of integers whose length determines the number of layers in the neural network and entries the number of neurons in each corresponding successive layer,
activation	the activation function for the hidden layers (should be either a callable function, preferably from the torch library),

p_drop	probability parameter for dropout before each hidden layer for regularization during training,
shape_fixed	whether the shape estimate depends on the covariates or not (bool),
device	a torch::torch_device() for an internal constant vector. Defaults to default_device().

Details

The constructor allows specifying:

D_in the input size (i.e. the number of features),

Hidden_vect a vector of integers whose length determines the number of layers in the neural network and entries the number of neurons in each corresponding successive layer,

activation the activation function for the hidden layers (should be either a callable function, preferably from the torch library),

p_drop probability parameter for dropout before each hidden layer for regularization during training,

shape_fixed whether the shape estimate depends on the covariates or not (bool),

device a torch::torch_device() for an internal constant vector. Defaults to default_device().

Value

The specified MLP GPD network as a torch::nn_module.

FC_GPD_SNN *Self-normalized fully-connected network module for GPD parameter prediction*

Description

A fully-connected self-normalizing network as a torch::nn_module, designed for generalized Pareto distribution parameter prediction.

Usage

```
FC_GPD_SNN(D_in, Hidden_vect = c(64, 64, 64), p_drop = 0.01)
```

Arguments

D_in	the input size (i.e. the number of features),
Hidden_vect	a vector of integers whose length determines the number of layers in the neural network and entries the number of neurons in each corresponding successive layer,
p_drop	probability parameter for the alpha-dropout before each hidden layer for regularization during training.

Details

The constructor allows specifying:

D_in the input size (i.e. the number of features),

Hidden_vect a vector of integers whose length determines the number of layers in the neural network and entries the number of neurons in each corresponding successive layer,

p_drop probability parameter for the `alpha-dropout` before each hidden layer for regularization during training.

Value

The specified SNN MLP GPD network as a `torch::nn_module`.

References

Gunter Klambauer, Thomas Unterthiner, Andreas Mayr, Sepp Hochreiter. Self-Normalizing Neural Networks. Advances in Neural Information Processing Systems 30 (NIPS 2017), 2017.

`fit_GPD_unconditional` *Maximum likelihood estimates for the GPD distribution using peaks over threshold*

Description

Maximum likelihood estimates for the GPD distribution using peaks over threshold

Usage

```
fit_GPD_unconditional(Y, interm_lvl = NULL, thresh_quantiles = NULL)
```

Arguments

`Y`	Vector of observations
`interm_lvl`	Probability level at which the empirical quantile should be used as the threshold, if `thresh_quantiles` is not given.
`thresh_quantiles`	Numerical value or numerical vector of the same length as `Y` representing either a fixed or a varying threshold, respectively.

Value

Named list containing:

`scale`	the GPD scale MLE,
`shape`	the GPD shape MLE,
`fit`	the fitted `ismev::gpd.fit()` object.

get_doFuture_operator *Get doFuture operator*

Description

Get doFuture operator

Usage

```
get_doFuture_operator(
  strategy = c("sequential", "multisession", "multicore", "mixed")
)
```

Arguments

strategy	One of "sequential" (default), "multisession", "multicore", or "mixed".

Value

Returns the appropriate operator to use in a foreach::foreach() loop. The %do% operator is returned if strategy=="sequential". Otherwise, the %dopar% operator is returned.

Examples

```
`%fun%` <- get_doFuture_operator("sequential")
```

get_excesses *Computes rescaled excesses over the conditional quantiles*

Description

Computes rescaled excesses over the conditional quantiles

Usage

```
get_excesses(
  X = NULL,
  y,
  quantiles,
  intermediate_q_feature = FALSE,
  scale_features = FALSE,
  X_scaling = NULL
)
```

Arguments

X	A covariate matrix. Can be NULL if there are no covariates.
y	The response variable vector.
quantiles	The intermediate quantiles over which to compute the excesses of y.
intermediate_q_feature	Whether to use the intermediate quantiles as an additional covariate, by appending it to the X matrix (bool).
scale_features	Whether to rescale each input covariates to zero mean and unit variance before applying the network (recommended). If X_scaling is given, X_scaling$scaling overrides scale_features.
X_scaling	Existing "X_scaling" object containing the precomputed mean and variance for each covariate. This enables reusing the scaling choice and parameters from the train set, if computing the excesses on a validation or test set, in order to avoid overfitting. This is performed automatically in the "EQRN" objects.

Value

Named list containing:

Y_excesses	thematrix of response excesses,
X_excesses	the (possibly rescaled and q_feat transformed) covariate matrix,
X_scaling	object of class "X_scaling" to use for consistent scaling on future datasets,
excesses_ratio	and the ratio of escesses for troubleshooting.

GPD_excess_probability

Tail excess probability prediction based on conditional GPD parameters

---

Description

Tail excess probability prediction based on conditional GPD parameters

Usage

```
GPD_excess_probability(
  val,
  sigma,
  xi,
  interm_threshold,
  threshold_p,
  body_proba = "default",
  proba_type = c("excess", "cdf")
)
```

Arguments

val	Quantile value(s) used to estimate the conditional excess probability or cdf.
sigma	Value(s) for the GPD scale parameter.
xi	Value(s) for the GPD shape parameter.
interm_threshold	Intermediate (conditional) quantile(s) at level threshold_p used as a (varying) threshold.
threshold_p	Probability level of the intermediate conditional quantiles interm_threshold.
body_proba	Value to use when the predicted conditional probability is below threshold_p (in which case it cannot be precisely assessed by the model). If "default" is given (the default), paste0(">",1-threshold_p) is used if proba_type=="excess", and paste0("<",threshold_p) is used if proba_type=="cdf".
proba_type	Whether to return the "excess" probability over val (default) or the "cdf" at val.

Value

Vector of probabilities (and possibly a few body_proba values if val is not large enough) of the same length as the longest vector between val, sigma, xi and interm_threshold.

GPD_quantiles *Compute extreme quantile from GPD parameters*

Description

Compute extreme quantile from GPD parameters

Usage

```
GPD_quantiles(p, p0, t_x0, sigma, xi)
```

Arguments

p	Probability level of the desired extreme quantile.
p0	Probability level of the (possibly varying) intermediate threshold/quantile.
t_x0	Value(s) of the (possibly varying) intermediate threshold/quantile.
sigma	Value(s) for the GPD scale parameter.
xi	Value(s) for the GPD shape parameter.

Value

The quantile value at probability level p.

install_backend *Install Torch Backend Libraries*

Description

This function can be called just after installing the EQRN package. Calling `EQRN::install_backend()` installs the necessary LibTorch and LibLantern backend libraries of the `torch` dependency by calling `torch::install_torch()`. See `https://torch.mlverse.org/docs/articles/installation` for more details and troubleshooting. Calling this function shouldn't be necessary in interactive environments, as loading EQRN (e.g. with `library(EQRN)` or with any `EQRN::fct()`) should prompt to do it automatically (via `.onLoad()`). This behaviour is inherited from the `torch` package.

Usage

```
install_backend(...)
```

Arguments

`...`	Arguments passed to `torch::install_torch()`.

Value

No return value.

lagged_features *Covariate lagged replication for temporal dependence*

Description

Covariate lagged replication for temporal dependence

Usage

```
lagged_features(X, max_lag, drop_present = TRUE)
```

Arguments

`X`	Covariate matrix.
`max_lag`	Integer giving the maximum lag (i.e. the number of temporal dependence steps).
`drop_present`	Whether to drop the "present" features (bool).

Value

Matrix with the original columns replicated, and shifted by `1:max_lag` if `drop_present==TRUE` (default) or by `0:max_lag` if `drop_present==FALSE`.

Examples

```
lagged_features(matrix(seq(20), ncol=2), max_lag=3, drop_present=TRUE)
```

---

last_elem *Last element of a vector*

Description

Returns the last element of the given vector in the most efficient way.

Usage

```
last_elem(x)
```

Arguments

x	Vector.

Details

The last element is obtained using `x[length(x)]`, which is done in `O(1)` and faster than, for example, any of `Rcpp::mylast(x)`, `tail(x, n=1)`, `dplyr::last(x)`, `x[end(x)[1]]]`, and `rev(x)[1]`.

Value

The last element in the vector `x`.

Examples

```
last_elem(c(2, 6, 1, 4))
```

---

loss_GPD *Generalized Pareto likelihood loss*

Description

Generalized Pareto likelihood loss

Usage

```
loss_GPD(
  sigma,
  xi,
  y,
  rescaled = TRUE,
  interm_lvl = NULL,
  return_vector = FALSE
)
```

Arguments

sigma	Value(s) for the GPD scale parameter.
xi	Value(s) for the GPD shape parameter.
y	Vector of observations
rescaled	Whether y already is a vector of excesses (TRUE) or needs rescaling (FALSE).
interm_lvl	Probability level at which the empirical quantile should be used as the intermediate threshold to compute the excesses, if rescaled==FALSE.
return_vector	Whether to return the the vector of GPD losses for each observation instead of the negative log-likelihood (average loss).

Value

GPD negative log-likelihood of the GPD parameters over the sample of observations.

---

loss_GPD_tensor *GPD tensor loss function for training a EQRN network*

Description

GPD tensor loss function for training a EQRN network

Usage

```
loss_GPD_tensor(
  out,
  y,
  orthogonal_gpd = TRUE,
  shape_penalty = 0,
  prior_shape = NULL,
  return_agg = c("mean", "sum", "vector", "nanmean", "nansum")
)
```

Arguments

out	Batch tensor of GPD parameters output by the network.
y	Batch tensor of corresponding response variable.
orthogonal_gpd	Whether the network is supposed to regress in the orthogonal reparametrization of the GPD parameters (recommended).
shape_penalty	Penalty parameter for the shape estimate, to potentially regularize its variation from the fixed prior estimate.
prior_shape	Prior estimate for the shape, used only if shape_penalty>0.
return_agg	The return aggregation of the computed loss over the batch. Must be one of "mean", "sum", "vector", "nanmean", "nansum".

Value

The GPD loss over the batch between the network output and the observed responses as a `torch::Tensor`, whose dimensions depend on `return_agg`.

---

make_folds *Create cross-validation folds*

Description

Utility function to create folds of data, used in cross-validation proceidures. The implementation is originally from the `gbex R` package

Usage

```
make_folds(y, num_folds, stratified = FALSE)
```

Arguments

`y`	Numerical vector of observations
`num_folds`	Number of folds to create.
`stratified`	Logical value. If `TRUE`, the folds are stratified along `rank(y)`.

Value

Vector of indices of the assigned folds for each observation.

Examples

```
make_folds(rnorm(30), 5)
```

---

mean_absolute_error *Mean absolute error*

Description

Mean absolute error

Usage

```
mean_absolute_error(
  y,
  y_hat,
  return_agg = c("mean", "sum", "vector"),
  na.rm = FALSE
)
```

Arguments

y	Vector of observations or ground-truths.
y_hat	Vector of predictions.
return_agg	Whether to return the "mean" (default), "sum", or "vector" of errors.
na.rm	A logical value indicating whether NA values should be stripped before the computation proceeds.

Value

The mean (or total or vectorial) absolute error between y and y_hat.

Examples

```
mean_absolute_error(c(2.3, 4.2, 1.8), c(2.2, 4.6, 1.7))
```

---

mean_squared_error *Mean squared error*

Description

Mean squared error

Usage

```
mean_squared_error(
  y,
  y_hat,
  return_agg = c("mean", "sum", "vector"),
  na.rm = FALSE
)
```

Arguments

y	Vector of observations or ground-truths.
y_hat	Vector of predictions.
return_agg	Whether to return the "mean" (default), "sum", or "vector" of errors.
na.rm	A logical value indicating whether NA values should be stripped before the computation proceeds.

Value

The mean (or total or vectorial) squared error between y and y_hat.

Examples

```
mean_squared_error(c(2.3, 4.2, 1.8), c(2.2, 4.6, 1.7))
```

mts_dataset *Dataset creator for sequential data*

Description

A `torch::dataset` object that can be initialized with sequential data, used to feed a recurrent network during training or prediction. It is used in `EQRN_fit_seq()` and corresponding predict functions, as well as in other recurrent methods such as `QRN_seq_fit()` and its predict functions. It can perform scaling of the response's past as a covariate, and compute excesses as a response when used in `EQRN_fit_seq()`. It also allows for fold separation or sequential discontinuity in the data.

Usage

```
mts_dataset(
  Y,
  X,
  seq_len,
  intermediate_quantiles = NULL,
  scale_Y = TRUE,
  fold_separation = NULL,
  sample_frac = 1,
  device = EQRN::default_device()
)
```

Arguments

`Y`	Response variable vector to model the extreme conditional quantile of, for training. Entries must be in sequential order.
`X`	Matrix of covariates, for training. Entries must be in sequential order.
`seq_len`	Data sequence length (i.e. number of past observations) used during training to predict each response quantile.
`intermediate_quantiles`	Vector of intermediate conditional quantiles at level `interm_lvl`.
`scale_Y`	Whether to rescale the response past, when considered as an input covariate, to zero mean and unit covariance before applying the network (recommended).
`fold_separation`	Fold separation index, when using concatenated folds as data.
`sample_frac`	Value between `0` and `1`. If `sample_frac` < `1`, a subsample of the data is used. Defaults to `1`.
`device`	(optional) A `torch::torch_device()`. Defaults to `default_device()`.

Value

The `torch::dataset` containing the given data, to be used with a recurrent neural network.

multilevel_exceedance_proba_error *Multilevel 'quantile_exceedance_proba_error'*

Description

Multilevel version of `quantile_exceedance_proba_error()`.

Usage

```
multilevel_exceedance_proba_error(
  Probs,
  proba_levels = NULL,
  return_years = NULL,
  type_probs = c("cdf", "exceedance"),
  prefix = "",
  na.rm = FALSE,
  give_names = TRUE
)
```

Arguments

`Probs`	Matrix, whose columns give, for each `proba_levels`, the predicted probabilities to exceed or be smaller than a fixed quantile.
`proba_levels`	Vector of probability levels of the quantiles.
`return_years`	The probability levels can be given in term or return years instead. Only used if `proba_levels` is not given.
`type_probs`	Whether the predictions are the `"cdf"` (default) or `"exceedance"` probabilities.
`prefix`	A string prefix to add to the output's names (if `give_names` is `TRUE`).
`na.rm`	A logical value indicating whether `NA` values should be stripped before the computation proceeds.
`give_names`	Whether to name the output errors (bool).

Value

A vector of length `length(proba_levels)` giving the `quantile_exceedance_proba_error()` calibration metric of each column of `Probs` at the corresponding `proba_levels`. If `give_names` is `TRUE`, the output vector is named `paste0(prefix, "exPrErr_q", proba_levels)` (or `paste0(prefix, "exPrErr_", return_years, "y")` if `return_years` are given instead of `proba_levels`).

multilevel_MAE *Multilevel quantile MAEs*

Description

Multilevel version of `mean_absolute_error()`.

Usage

```
multilevel_MAE(
  True_Q,
  Pred_Q,
  proba_levels,
  prefix = "",
  na.rm = FALSE,
  give_names = TRUE,
  sd = FALSE
)
```

Arguments

`True_Q`	Matrix of size `n_obs` times `proba_levels`, whose columns are the vectors of ground-truths at each `proba_levels` and each row corresponds to an observation or realisation.
`Pred_Q`	Matrix of the same size as `True_Q`, whose columns are the predictions at each `proba_levels` and each row corresponds to an observation or realisation.
`proba_levels`	Vector of probability levels at which the predictions were made. Must be of length `ncol(Pred_Q)`.
`prefix`	A string prefix to add to the output's names (if `give_names` is `TRUE`).
`na.rm`	A logical value indicating whether `NA` values should be stripped before the computation proceeds.
`give_names`	Whether to name the output MAEs (bool).
`sd`	Whether to return the absolute error standard deviation (bool).

Value

A vector of length `length(proba_levels)` giving the mean absolute errors between each respective columns of `True_Q` and `Pred_Q`. If `give_names` is `TRUE`, the output vector is named `paste0(prefix, "MAE_q", proba_levels)`. If `sd==TRUE` a named list is instead returned, containing the `"MAEs"` described above and `"SDs"`, their standard deviations.

multilevel_MSE *Multilevel quantile MSEs*

Description

Multilevel version of mean_squared_error().

Usage

```
multilevel_MSE(
  True_Q,
  Pred_Q,
  proba_levels,
  prefix = "",
  na.rm = FALSE,
  give_names = TRUE,
  sd = FALSE
)
```

Arguments

True_Q	Matrix of size n_obs times proba_levels, whose columns are the vectors of ground-truths at each proba_levels and each row corresponds to an observation or realisation.
Pred_Q	Matrix of the same size as True_Q, whose columns are the predictions at each proba_levels and each row corresponds to an observation or realisation.
proba_levels	Vector of probability levels at which the predictions were made. Must be of length ncol(Pred_Q).
prefix	A string prefix to add to the output's names (if give_names is TRUE).
na.rm	A logical value indicating whether NA values should be stripped before the computation proceeds.
give_names	Whether to name the output MSEs (bool).
sd	Whether to return the squared error standard deviation (bool).

Value

A vector of length length(proba_levels) giving the mean square errors between each respective columns of True_Q and Pred_Q. If give_names is TRUE, the output vector is named paste0(prefix, "MSE_q", proba_levels). If sd==TRUE a named list is instead returned, containing the "MSEs" described above and "SDs", their standard deviations.

`multilevel_pred_bias` *Multilevel prediction bias*

Description

Multilevel version of `prediction_bias()`.

Usage

```
multilevel_pred_bias(
  True_Q,
  Pred_Q,
  proba_levels,
  square_bias = FALSE,
  prefix = "",
  na.rm = FALSE,
  give_names = TRUE
)
```

Arguments

`True_Q`	Matrix of size `n_obs` times `proba_levels`, whose columns are the vectors of ground-truths at each `proba_levels` and each row corresponds to an observation or realisation.
`Pred_Q`	Matrix of the same size as `True_Q`, whose columns are the predictions at each `proba_levels` and each row corresponds to an observation or realisation.
`proba_levels`	Vector of probability levels at which the predictions were made. Must be of length `ncol(Pred_Q)`.
`square_bias`	Whether to return the square bias (bool); defaults to `FALSE`.
`prefix`	A string prefix to add to the output's names (if `give_names` is `TRUE`).
`na.rm`	A logical value indicating whether `NA` values should be stripped before the computation proceeds.
`give_names`	Whether to name the output MSEs (bool).

Value

A vector of length `length(proba_levels)` giving the (square) bias of each columns of predictions in `Pred_Q` for the respective `True_Q`. If `give_names` is `TRUE`, the output vector is named `paste0(prefix, "MSE_q", proba_levels)`.

`multilevel_prop_below` *Multilevel 'proportion_below'*

Description

Multilevel version of `proportion_below()`.

Usage

```
multilevel_prop_below(
  y,
  Pred_Q,
  proba_levels,
  prefix = "",
  na.rm = FALSE,
  give_names = TRUE
)
```

Arguments

`y`	Vector of observations.
`Pred_Q`	Matrix of of size `length(y)` times `proba_levels`, whose columns are the quantile predictions at each `proba_levels` and each row corresponds to an observation or realisation.
`proba_levels`	Vector of probability levels at which the predictions were made. Must be of length `ncol(Pred_Q)`.
`prefix`	A string prefix to add to the output's names (if `give_names` is `TRUE`).
`na.rm`	A logical value indicating whether `NA` values should be stripped before the computation proceeds.
`give_names`	Whether to name the output proportions (bool).

Value

A vector of length `length(proba_levels)` giving the proportion of observations below the predictions (`Pred_Q`) at each probability level. If `give_names` is `TRUE`, the output vector is named `paste0(prefix, "propBelow_q", proba_levels)`.

`multilevel_q_loss` *Multilevel quantile losses*

Description

Multilevel version of `quantile_loss()`.

Usage

```
multilevel_q_loss(
  y,
  Pred_Q,
  proba_levels,
  prefix = "",
  na.rm = FALSE,
  give_names = TRUE
)
```

Arguments

`y`	Vector of observations.
`Pred_Q`	Matrix of of size `length(y)` times `proba_levels`, whose columns are the quantile predictions at each `proba_levels` and each row corresponds to an observation or realisation.
`proba_levels`	Vector of probability levels at which the predictions were made. Must be of length `ncol(Pred_Q)`.
`prefix`	A string prefix to add to the output's names (if `give_names` is `TRUE`).
`na.rm`	A logical value indicating whether `NA` values should be stripped before the computation proceeds.
`give_names`	Whether to name the output quantile errors (bool).

Value

A vector of length `length(proba_levels)` giving the average quantile losses between each column of `Pred_Q` and the observations. If `give_names` is `TRUE`, the output vector is named `paste0(prefix, "qloss_q", proba_levels)`.

multilevel_q_pred_error *Multilevel 'quantile_prediction_error'*

Description

Multilevel version of quantile_prediction_error().

Usage

```
multilevel_q_pred_error(
  y,
  Pred_Q,
  proba_levels,
  prefix = "",
  na.rm = FALSE,
  give_names = TRUE
)
```

Arguments

y	Vector of observations.
Pred_Q	Matrix of of size length(y) times proba_levels, whose columns are the quantile predictions at each proba_levels and each row corresponds to an observation or realisation.
proba_levels	Vector of probability levels at which the predictions were made. Must be of length ncol(Pred_Q).
prefix	A string prefix to add to the output's names (if give_names is TRUE).
na.rm	A logical value indicating whether NA values should be stripped before the computation proceeds.
give_names	Whether to name the output errors (bool).

Value

A vector of length length(proba_levels) giving the quantile prediction error calibration metrics between each column of Pred_Q and the observations. If give_names is TRUE, the output vector is named paste0(prefix, "qPredErr_q", proba_levels).

`multilevel_resid_var` *Multilevel residual variance*

Description

Multilevel version of `prediction_residual_variance()`.

Usage

```
multilevel_resid_var(
  True_Q,
  Pred_Q,
  proba_levels,
  prefix = "",
  na.rm = FALSE,
  give_names = TRUE
)
```

Arguments

`True_Q`	Matrix of size `n_obs` times `proba_levels`, whose columns are the vectors of ground-truths at each `proba_levels` and each row corresponds to an observation or realisation.
`Pred_Q`	Matrix of the same size as `True_Q`, whose columns are the predictions at each `proba_levels` and each row corresponds to an observation or realisation.
`proba_levels`	Vector of probability levels at which the predictions were made. Must be of length `ncol(Pred_Q)`.
`prefix`	A string prefix to add to the output's names (if `give_names` is `TRUE`).
`na.rm`	A logical value indicating whether `NA` values should be stripped before the computation proceeds.
`give_names`	Whether to name the output MSEs (bool).

Value

A vector of length `length(proba_levels)` giving the residual variances of each columns of predictions in `Pred_Q` for the respective `True_Q`. If `give_names` is `TRUE`, the output vector is named `paste0(prefix, "MSE_q", proba_levels)`.

`multilevel_R_squared` *Multilevel R squared*

Description

Multilevel version of `R_squared()`.

Usage

```
multilevel_R_squared(
  True_Q,
  Pred_Q,
  proba_levels,
  prefix = "",
  na.rm = FALSE,
  give_names = TRUE
)
```

Arguments

`True_Q`	Matrix of size `n_obs` times `proba_levels`, whose columns are the vectors of ground-truths at each `proba_levels` and each row corresponds to an observation or realisation.
`Pred_Q`	Matrix of the same size as `True_Q`, whose columns are the predictions at each `proba_levels` and each row corresponds to an observation or realisation.
`proba_levels`	Vector of probability levels at which the predictions were made. Must be of length `ncol(Pred_Q)`.
`prefix`	A string prefix to add to the output's names (if `give_names` is `TRUE`).
`na.rm`	A logical value indicating whether `NA` values should be stripped before the computation proceeds.
`give_names`	Whether to name the output MSEs (bool).

Value

A vector of length `length(proba_levels)` giving the R squared coefficient of determination of each columns of predictions in `Pred_Q` for the respective `True_Q`. If `give_names` is `TRUE`, the output vector is named `paste0(prefix, "MSE_q", proba_levels)`.

perform_scaling *Performs feature scaling without overfitting*

Description

Performs feature scaling without overfitting

Usage

```
perform_scaling(X, X_scaling = NULL, scale_features = TRUE, stat_attr = FALSE)
```

Arguments

X	A covariate matrix.
X_scaling	Existing "X_scaling" object containing the precomputed mean and variance for each covariate. This enables reusing the scaling choice and parameters from the train set, if computing the excesses on a validation or test set, in order to avoid overfitting. This is performed automatically in the "EQRN" objects.
scale_features	Whether to rescale each input covariates to zero mean and unit variance before applying the model (recommended). If X_scaling is given, X_scaling$scaling overrides scale_features.
stat_attr	DEPRECATED. Whether to keep attributes in the returned covariate matrix itself.

Value

Named list containing:

X_excesses	the (possibly rescaled and q_feat transformed) covariate matrix,
X_scaling	object of class "X_scaling" to use for consistent scaling on future datasets.

predict.EQRN_iid *Predict method for an EQRN_iid fitted object*

Description

Predict method for an EQRN_iid fitted object

Usage

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

Arguments

object	Fitted "EQRN_iid" object.
...	Arguments passed on to EQRN_predict
	X Matrix of covariates to predict the corresponding response's conditional quantiles.
	prob_lvls_predict Vector of probability levels at which to predict the conditional quantiles.
	intermediate_quantiles Vector of intermediate conditional quantiles at level fit_eqrn$interm_lvl.
	interm_lvl Optional, checks that interm_lvl == fit_eqrn$interm_lvl.
	device (optional) A torch::torch_device(). Defaults to default_device().

Details

See EQRN_predict() for more details.

Value

Matrix of size nrow(X) times prob_lvls_predict containing the conditional quantile estimates of the response associated to each covariate observation at each probability level. Simplifies to a vector if length(prob_lvls_predict)==1.

---

predict.EQRN_seq *Predict method for an EQRN_seq fitted object*

Description

Predict method for an EQRN_seq fitted object

Usage

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

Arguments

object	Fitted "EQRN_seq" object.
...	Arguments passed on to EQRN_predict_seq
	X Matrix of covariates to predict the corresponding response's conditional quantiles.
	Y Response variable vector corresponding to the rows of X.
	prob_lvls_predict Vector of probability levels at which to predict the conditional quantiles.
	intermediate_quantiles Vector of intermediate conditional quantiles at level fit_eqrn$interm_lvl.

`interm_lvl` Optional, checks that `interm_lvl == fit_eqrn$interm_lvl`.

`crop_predictions` Whether to crop out the fist `seq_len` observations (which are `NA`) from the returned matrix.

`seq_len` Data sequence length (i.e. number of past observations) used to predict each response quantile. By default, the training `fit_eqrn$seq_len` is used.

`device` (optional) A `torch::torch_device()`. Defaults to `default_device()`.

Details

See `EQRN_predict_seq()` for more details.

Value

Matrix of size `nrow(X)` times `prob_lvls_predict` (or `nrow(X)-seq_len` times `prob_lvls_predict` if `crop_predictions`) containing the conditional quantile estimates of the corresponding response observations at each probability level. Simplifies to a vector if `length(prob_lvls_predict)==1`.

---

predict.QRN_seq *Predict method for a QRN_seq fitted object*

Description

Predict method for a QRN_seq fitted object

Usage

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

Arguments

`object` Fitted "`QRN_seq`" object.

`...` Arguments passed on to `QRN_seq_predict`

`X` Matrix of covariates to predict the corresponding response's conditional quantiles.

`Y` Response variable vector corresponding to the rows of `X`.

`q_level` Optional, checks that `q_level == fit_qrn_ts$interm_lvl`.

`crop_predictions` Whether to crop out the fist `seq_len` observations (which are `NA`) from the returned matrix.

`device` (optional) A `torch::torch_device()`. Defaults to `default_device()`.

Details

See `QRN_seq_predict()` for more details.

Value

Matrix of size `nrow(X)` times 1 (or `nrow(X)-seq_len` times 1 if `crop_predictions`) containing the conditional quantile estimates of the corresponding response observations.

`prediction_bias` *Prediction bias*

Description

Prediction bias

Usage

```
prediction_bias(y, y_hat, square_bias = FALSE, na.rm = FALSE)
```

Arguments

`y`	Vector of observations or ground-truths.
`y_hat`	Vector of predictions.
`square_bias`	Whether to return the square bias (bool); defaults to `FALSE`.
`na.rm`	A logical value indicating whether `NA` values should be stripped before the computation proceeds.

Value

The (square) bias of the predictions `y_hat` for `y`.

Examples

```
prediction_bias(c(2.3, 4.2, 1.8), c(2.2, 4.6, 1.7))
```

`prediction_residual_variance` *Prediction residual variance*

Description

Prediction residual variance

Usage

```
prediction_residual_variance(y, y_hat, na.rm = FALSE)
```

Arguments

y	Vector of observations or ground-truths.
y_hat	Vector of predictions.
na.rm	A logical value indicating whether NA values should be stripped before the computation proceeds.

Value

The residual variance of the predictions y_hat for y.

Examples

```
prediction_residual_variance(c(2.3, 4.2, 1.8), c(2.2, 4.6, 1.7))
```

---

predict_GPD_semiconditional

Predict semi-conditional extreme quantiles using peaks over threshold

---

Description

Predict semi-conditional extreme quantiles using peaks over threshold

Usage

```
predict_GPD_semiconditional(
  Y,
  interm_lvl,
  thresh_quantiles,
  interm_quantiles_test = thresh_quantiles,
  prob_lvls_predict = c(0.99)
)
```

Arguments

Y	Vector of ("training") observations.
interm_lvl	Probability level at which the empirical quantile should be used as the intermediate threshold.
thresh_quantiles	Numerical vector of the same length as Y representing the varying intermediate threshold on the train set.
interm_quantiles_test	Numerical vector of the same length as Y representing the varying intermediate threshold used for prediction on the test set.
prob_lvls_predict	Probability levels at which to predict the extreme semi-conditional quantiles.

Value

Named list containing:

predictions	matrix of dimension length(interm_quantiles_test) times length(prob_lvls_predict) containing the estimated extreme quantile at levels quantile, for each interm_quantiles_test,
pars	matrix of dimension ntest times 2 containing the two GPD parameter MLEs, repeated length(interm_quantiles_test) times.

`predict_unconditional_quantiles`

Predict unconditional extreme quantiles using peaks over threshold

---

Description

Predict unconditional extreme quantiles using peaks over threshold

Usage

```
predict_unconditional_quantiles(interm_lvl, quantiles = c(0.99), Y, ntest = 1)
```

Arguments

interm_lvl	Probability level at which the empirical quantile should be used as the intermediate threshold.
quantiles	Probability levels at which to predict the extreme quantiles.
Y	Vector of ("training") observations.
ntest	Number of "test" observations.

Value

Named list containing:

predictions	matrix of dimension ntest times length(quantiles) containing the estimated extreme quantile at levels quantile, repeated ntest times,
pars	matrix of dimension ntest times 2 containing the two GPD parameter MLEs, repeated ntest times.
threshold	The threshold for the peaks-over-threshold GPD model. It is the empirical quantile of Y at level interm_lvl, i.e. stats::quantile(Y, interm_lvl).

process_features *Feature processor for EQRN*

Description

Feature processor for EQRN

Usage

```
process_features(
  X,
  intermediate_q_feature,
  intermediate_quantiles = NULL,
  X_scaling = NULL,
  scale_features = TRUE
)
```

Arguments

X	A covariate matrix.
intermediate_q_feature	Whether to use the intermediate quantiles as an additional covariate, by appending it to the X matrix (bool).
intermediate_quantiles	The intermediate conditional quantiles.
X_scaling	Existing "X_scaling" object containing the precomputed mean and variance for each covariate. This enables reusing the scaling choice and parameters from the train set, if computing the excesses on a validation or test set, in order to avoid overfitting. This is performed automatically in the "EQRN" objects.
scale_features	Whether to rescale each input covariates to zero mean and unit variance before applying the network (recommended). If X_scaling is given, X_scaling$scaling overrides scale_features.

Value

Named list containing:

X_excesses	the (possibly rescaled and q_feat transformed) covariate matrix,
X_scaling	object of class "X_scaling" to use for consistent scaling on future datasets.

proportion_below *Proportion of observations below conditional quantile vector*

Description

Proportion of observations below conditional quantile vector

Usage

```
proportion_below(y, Q_hat, na.rm = FALSE)
```

Arguments

y	Vector of observations.
Q_hat	Vector of predicted quantiles.
na.rm	A logical value indicating whether NA values should be stripped before the computation proceeds.

Value

The proportion of observation below the predictions.

Examples

```
proportion_below(c(2.3, 4.2, 1.8), c(2.9, 5.6, 1.7))
```

QRNN_RNN_net *Recurrent quantile regression neural network module*

Description

A recurrent neural network as a torch::nn_module, designed for quantile regression.

Usage

```
QRNN_RNN_net(
  type = c("lstm", "gru"),
  nb_input_features,
  hidden_size,
  num_layers = 1,
  dropout = 0
)
```

Arguments

type	the type of recurrent architecture, can be one of "lstm" (default) or "gru",
nb_input_features	the input size (i.e. the number of features),
hidden_size	the dimension of the hidden latent state variables in the recurrent network,
num_layers	the number of recurrent layers,
dropout	probability parameter for dropout before each hidden layer for regularization during training.

Details

The constructor allows specifying:

type the type of recurrent architecture, can be one of "lstm" (default) or "gru",

nb_input_features the input size (i.e. the number of features),

hidden_size the dimension of the hidden latent state variables in the recurrent network,

num_layers the number of recurrent layers,

dropout probability parameter for dropout before each hidden layer for regularization during training.

Value

The specified recurrent QRN as a torch::nn_module.

QRN_fit_multiple *Wrapper for fitting a recurrent QRN with restart for stability*

Description

Wrapper for fitting a recurrent QRN with restart for stability

Usage

```
QRN_fit_multiple(
  X,
  y,
  q_level,
  number_fits = 3,
  ...,
  seed = NULL,
  data_type = c("seq", "iid")
)
```

Arguments

X	Matrix of covariates, for training.
y	Response variable vector to model the conditional quantile of, for training.
q_level	Probability level of the desired conditional quantiles to predict.
number_fits	Number of restarts.
...	Other parameters given to QRN_seq_fit().
seed	Integer random seed for reproducibility in network weight initialization.
data_type	Type of data dependence, must be one of "iid" (for iid observations) or "seq" (for sequentially dependent observations). For the moment, only "seq" is accepted.

Value

An QRN object of classes c("QRN_seq", "QRN"), containing the fitted network, as well as all the relevant information for its usage in other functions.

QRN_seq_fit *Recurrent QRN fitting function*

Description

Used to fit a recurrent quantile regression neural network on a data sample. Use the QRN_fit_multiple() wrapper instead, with data_type="seq", for better stability using fitting restart.

Usage

```
QRN_seq_fit(
  X,
  Y,
  q_level,
  hidden_size = 10,
  num_layers = 1,
  rnn_type = c("lstm", "gru"),
  p_drop = 0,
  learning_rate = 1e-04,
  L2_pen = 0,
  seq_len = 10,
  scale_features = TRUE,
  n_epochs = 10000,
  batch_size = 256,
  X_valid = NULL,
  Y_valid = NULL,
  lr_decay = 1,
  patience_decay = n_epochs,
  min_lr = 0,
```

```
  patience_stop = n_epochs,
  tol = 1e-04,
  fold_separation = NULL,
  warm_start_path = NULL,
  patience_lag = 5,
  optim_met = "adam",
  seed = NULL,
  verbose = 2,
  device = default_device()
)
```

Arguments

X	Matrix of covariates, for training. Entries must be in sequential order.
Y	Response variable vector to model the conditional quantile of, for training. Entries must be in sequential order.
q_level	Probability level of the desired conditional quantiles to predict.
hidden_size	Dimension of the hidden latent state variables in the recurrent network.
num_layers	Number of recurrent layers.
rnn_type	Type of recurrent architecture, can be one of "lstm" (default) or "gru".
p_drop	Probability parameter for dropout before each hidden layer for regularization during training.
learning_rate	Initial learning rate for the optimizer during training of the neural network.
L2_pen	L2 weight penalty parameter for regularization during training.
seq_len	Data sequence length (i.e. number of past observations) used during training to predict each response quantile.
scale_features	Whether to rescale each input covariates to zero mean and unit covariance before applying the network (recommended).
n_epochs	Number of training epochs.
batch_size	Batch size used during training.
X_valid	Covariates in a validation set, or NULL. Entries must be in sequential order. Used for monitoring validation loss during training, enabling learning-rate decay and early stopping.
Y_valid	Response variable in a validation set, or NULL. Entries must be in sequential order. Used for monitoring validation loss during training, enabling learning-rate decay and early stopping.
lr_decay	Learning rate decay factor.
patience_decay	Number of epochs of non-improving validation loss before a learning-rate decay is performed.
min_lr	Minimum learning rate, under which no more decay is performed.
patience_stop	Number of epochs of non-improving validation loss before early stopping is performed.

tol	Tolerance for stopping training, in case of no significant training loss improvements.
fold_separation	Index of fold separation or sequential discontinuity in the data.
warm_start_path	Path of a saved network using torch::torch_save(), to load back for a warm start.
patience_lag	The validation loss is considered to be non-improving if it is larger than on any of the previous patience_lag epochs.
optim_met	DEPRECATED. Optimization algorithm to use during training. "adam" is the default.
seed	Integer random seed for reproducibility in network weight initialization.
verbose	Amount of information printed during training (0:nothing, 1:most important, 2:everything).
device	(optional) A torch::torch_device(). Defaults to default_device().

Value

An QRN object of classes c("QRN_seq", "QRN"), containing the fitted network, as well as all the relevant information for its usage in other functions.

QRN_seq_predict *Predict function for a QRN_seq fitted object*

Description

Predict function for a QRN_seq fitted object

Usage

```
QRN_seq_predict(
  fit_qrn_ts,
  X,
  Y,
  q_level = fit_qrn_ts$interm_lvl,
  crop_predictions = FALSE,
  device = default_device()
)
```

Arguments

fit_qrn_ts	Fitted "QRN_seq" object.
X	Matrix of covariates to predict the corresponding response's conditional quantiles.
Y	Response variable vector corresponding to the rows of X.

q_level	Optional, checks that q_level == fit_qrn_ts$interm_lvl.
crop_predictions	Whether to crop out the fist seq_len observations (which are NA) from the returned matrix.
device	(optional) A torch::torch_device(). Defaults to default_device().

Value

Matrix of size nrow(X) times 1 (or nrow(X)-seq_len times 1 if crop_predictions) containing the conditional quantile estimates of the corresponding response observations.

QRN_seq_predict_foldwise

Foldwise fit-predict function using a recurrent QRN

Description

Foldwise fit-predict function using a recurrent QRN

Usage

```
QRN_seq_predict_foldwise(
  X,
  y,
  q_level,
  n_folds = 3,
  number_fits = 3,
  seq_len = 10,
  seed = NULL,
  ...
)
```

Arguments

X	Matrix of covariates, for training. Entries must be in sequential order.
y	Response variable vector to model the conditional quantile of, for training. Entries must be in sequential order.
q_level	Probability level of the desired conditional quantiles to predict.
n_folds	Number of folds.
number_fits	Number of restarts, for stability.
seq_len	Data sequence length (i.e. number of past observations) used during training to predict each response quantile.
seed	Integer random seed for reproducibility in network weight initialization.
...	Other parameters given to QRN_seq_fit().

Value

A named list containing the foldwise predictions and fits. It namely contains:

`predictions`	the numerical vector of quantile predictions for each observation entry in y,
`fits`	a list containing the `"QRN_seq"` fitted networks for each fold,
`cuts`	the fold cuts indices,
`folds`	a list of lists containing the train indices, validation indices and fold separations as a list for each fold setup,
`n_folds`	number of folds,
`q_level`	probability level of the predicted quantiles,
`train_losses`	the vector of train losses on each fold,
`valid_losses`	the vector of validation losses on each fold,
`min_valid_losses`	the minimal validation losses obtained on each fold,
`min_valid_e`	the epoch index of the minimal validation losses obtained on each fold.

QRN_seq_predict_foldwise_sep

Sigle-fold foldwise fit-predict function using a recurrent QRN

Description

Separated single-fold version of `QRN_seq_predict_foldwise()`, for computation purposes.

Usage

```
QRN_seq_predict_foldwise_sep(
  X,
  y,
  q_level,
  n_folds = 3,
  fold_todo = 1,
  number_fits = 3,
  seq_len = 10,
  seed = NULL,
  ...
)
```

Arguments

`X`	Matrix of covariates, for training. Entries must be in sequential order.
`y`	Response variable vector to model the conditional quantile of, for training. Entries must be in sequential order.
`q_level`	Probability level of the desired conditional quantiles to predict.

n_folds	Number of folds.
fold_todo	Index of the fold to do (integer in 1:n_folds).
number_fits	Number of restarts, for stability.
seq_len	Data sequence length (i.e. number of past observations) used during training to predict each response quantile.
seed	Integer random seed for reproducibility in network weight initialization.
...	Other parameters given to QRN_seq_fit().

Value

A named list containing the foldwise predictions and fits. It namely contains:

predictions	the numerical vector of quantile predictions for each observation entry in y,
fits	a list containing the "QRN_seq" fitted networks for each fold,
cuts	the fold cuts indices,
folds	a list of lists containing the train indices, validation indices and fold separations as a list for each fold setup,
n_folds	number of folds,
q_level	probability level of the predicted quantiles,
train_losses	the vector of train losses on each fold,
valid_losses	the vector of validation losses on each fold,
min_valid_losses	the minimal validation losses obtained on each fold,
min_valid_e	the epoch index of the minimal validation losses obtained on each fold.

---

quantile_exceedance_proba_error

Quantile exceedance probability prediction calibration error

Description

Quantile exceedance probability prediction calibration error

Usage

```
quantile_exceedance_proba_error(
  Probs,
  prob_level = NULL,
  return_years = NULL,
  type_probs = c("cdf", "exceedance"),
  na.rm = FALSE
)
```

Arguments

Probs	Predicted probabilities to exceed or be smaller than a fixed quantile.
prob_level	Probability level of the quantile.
return_years	The probability level can be given in term or return years instead. Only used if prob_level is not given.
type_probs	Whether the predictions are the "cdf" (default) or "exceedance" probabilities.
na.rm	A logical value indicating whether NA values should be stripped before the computation proceeds.

Value

The calibration metric for the predicted probabilities.

Examples

```
quantile_exceedance_proba_error(c(0.1, 0.3, 0.2), prob_level=0.8)
```

---

quantile_loss *Quantile loss*

Description

Quantile loss

Usage

```
quantile_loss(
  y,
  y_hat,
  q,
  return_agg = c("mean", "sum", "vector"),
  na.rm = FALSE
)
```

Arguments

y	Vector of observations.
y_hat	Vector of predicted quantiles at probability level q.
q	Probability level of the predicted quantile.
return_agg	Whether to return the "mean" (default), "sum", or "vector" of losses.
na.rm	A logical value indicating whether NA values should be stripped before the computation proceeds.

Value

The mean (or total or vectorial) quantile loss between y and y_hat at level q.

Examples

```
quantile_loss(c(2.3, 4.2, 1.8), c(2.9, 5.6, 2.7), q=0.8)
```

---

quantile_loss_tensor *Tensor quantile loss function for training a QRN network*

Description

Tensor quantile loss function for training a QRN network

Usage

```
quantile_loss_tensor(
  out,
  y,
  q = 0.5,
  return_agg = c("mean", "sum", "vector", "nanmean", "nansum")
)
```

Arguments

out	Batch tensor of the quantile output by the network.
y	Batch tensor of corresponding response variable.
q	Probability level of the predicted quantile
return_agg	The return aggregation of the computed loss over the batch. Must be one of "mean", "sum", "vector", "nanmean", "nansum".

Value

The quantile loss over the batch between the network output ans the observed responses as a torch::Tensor, whose dimensions depend on return_agg.

---

quantile_prediction_error *Quantile prediction calibration error*

Description

Quantile prediction calibration error

Usage

```
quantile_prediction_error(y, Q_hat, prob_level, na.rm = FALSE)
```

Arguments

y	Vector of observations.
Q_hat	Vector of predicted quantiles at probability level prob_level.
prob_level	Probability level of the predicted quantile.
na.rm	A logical value indicating whether NA values should be stripped before the computation proceeds.

Value

The quantile prediction error calibration metric.

Examples

```
quantile_prediction_error(c(2.3, 4.2, 1.8), c(2.9, 5.6, 2.7), prob_level=0.8)
```

Recurrent_GPD_net *Recurrent network module for GPD parameter prediction*

Description

A recurrent neural network as a torch::nn_module, designed for generalized Pareto distribution parameter prediction, with sequential dependence.

Usage

```
Recurrent_GPD_net(
  type = c("lstm", "gru"),
  nb_input_features,
  hidden_size,
  num_layers = 1,
  dropout = 0,
  shape_fixed = FALSE,
  device = EQRN::default_device()
)
```

Arguments

type	the type of recurrent architecture, can be one of "lstm" (default) or "gru",
nb_input_features	the input size (i.e. the number of features),
hidden_size	the dimension of the hidden latent state variables in the recurrent network,
num_layers	the number of recurrent layers,
dropout	probability parameter for dropout before each hidden layer for regularization during training,
shape_fixed	whether the shape estimate depends on the covariates or not (bool),
device	a torch::torch_device() for an internal constant vector. Defaults to default_device().

Details

The constructor allows specifying:

type the type of recurrent architecture, can be one of `"lstm"` (default) or `"gru"`,

nb_input_features the input size (i.e. the number of features),

hidden_size the dimension of the hidden latent state variables in the recurrent network,

num_layers the number of recurrent layers,

dropout probability parameter for dropout before each hidden layer for regularization during training,

shape_fixed whether the shape estimate depends on the covariates or not (bool),

device a `torch::torch_device()` for an internal constant vector. Defaults to `default_device()`.

Value

The specified recurrent GPD network as a `torch::nn_module`.

`roundm` *Mathematical number rounding*

Description

This function rounds numbers in the mathematical sense, as opposed to the base `R` function `round()` that rounds 'to the even digit'.

Usage

```
roundm(x, decimals = 0)
```

Arguments

`x`	Vector of numerical values to round.
`decimals`	Integer indicating the number of decimal places to be used.

Value

A vector containing the entries of `x`, rounded to `decimals` decimals.

Examples

```
roundm(2.25, 1)
```

R_squared *R squared*

Description

The coefficient of determination, often called R squared, is the proportion of data variance explained by the predictions.

Usage

```
R_squared(y, y_hat, na.rm = FALSE)
```

Arguments

y	Vector of observations or ground-truths.
y_hat	Vector of predictions.
na.rm	A logical value indicating whether NA values should be stripped before the computation proceeds.

Value

The R squared of the predictions y_hat for y.

Examples

```
R_squared(c(2.3, 4.2, 1.8), c(2.2, 4.6, 1.7))
```

safe_save_rds *Safe RDS save*

Description

Safe version of saveRDS(). If the given save path (i.e. dirname(file_path)) does not exist, it is created instead of raising an error.

Usage

```
safe_save_rds(object, file_path, recursive = TRUE, no_warning = FALSE)
```

Arguments

object	R variable or object to save on disk.
file_path	Path and name of the save file, as a string.
recursive	Should elements of the path other than the last be created? If TRUE, behaves like the Unix command mkdir -p.
no_warning	Whether to cancel the warning issued if a directory is created (bool).

Value

No return value.

Examples

```
safe_save_rds(c(1, 2, 8), "./some_folder/my_new_folder/my_vector.rds")
```

`semiconditional_train_valid_GPD_loss` *Semi-conditional GPD MLEs and their train-validation likelihoods*

Description

Semi-conditional GPD MLEs and their train-validation likelihoods

Usage

```
semiconditional_train_valid_GPD_loss(
  Y_train,
  Y_valid,
  interm_quant_train,
  interm_quant_valid
)
```

Arguments

`Y_train`	Vector of "training" observations on which to estimate the MLEs.
`Y_valid`	Vector of "validation" observations, on which to estimate the out of training sample GPD loss.
`interm_quant_train`	Vector of intermediate quantiles serving as a varying threshold for each training observation.
`interm_quant_valid`	Vector of intermediate quantiles serving as a varying threshold for each validation observation.

Value

Named list containing:

`scale`	GPD scale MLE inferred from the train set,
`shape`	GPD shape MLE inferred from the train set,
`train_loss`	the negative log-likelihoods of the MLEs over the training samples,
`valid_loss`	the negative log-likelihoods of the MLEs over the validation samples.

`Separated_GPD_SNN` *Self-normalized separated network module for GPD parameter prediction*

Description

A parameter-separated self-normalizing network as a `torch::nn_module`, designed for generalized Pareto distribution parameter prediction.

Usage

```
Separated_GPD_SNN(
  D_in,
  Hidden_vect_scale = c(64, 64, 64),
  Hidden_vect_shape = c(5, 3),
  p_drop = 0.01
)
```

Arguments

`D_in`	the input size (i.e. the number of features),
`Hidden_vect_scale`	a vector of integers whose length determines the number of layers in the subnetwork for the scale parameter and entries the number of neurons in each corresponding successive layer,
`Hidden_vect_shape`	a vector of integers whose length determines the number of layers in the subnetwork for the shape parameter and entries the number of neurons in each corresponding successive layer,
`p_drop`	probability parameter for the `alpha-dropout` before each hidden layer for regularization during training.

Details

The constructor allows specifying:

D_in the input size (i.e. the number of features),

Hidden_vect_scale a vector of integers whose length determines the number of layers in the subnetwork for the scale parameter and entries the number of neurons in each corresponding successive layer,

Hidden_vect_shape a vector of integers whose length determines the number of layers in the subnetwork for the shape parameter and entries the number of neurons in each corresponding successive layer,

p_drop probability parameter for the `alpha-dropout` before each hidden layer for regularization during training.

Value

The specified parameter-separated SNN MLP GPD network as a torch::nn_module.

References

Gunter Klambauer, Thomas Unterthiner, Andreas Mayr, Sepp Hochreiter. Self-Normalizing Neural Networks. Advances in Neural Information Processing Systems 30 (NIPS 2017), 2017.

`set_doFuture_strategy` *Set a doFuture execution strategy*

Description

Set a doFuture execution strategy

Usage

```
set_doFuture_strategy(
  strategy = c("sequential", "multisession", "multicore", "mixed"),
  n_workers = NULL
)
```

Arguments

`strategy`	One of `"sequential"` (default), `"multisession"`, `"multicore"`, or `"mixed"`.
`n_workers`	A positive numeric scalar or a function specifying the maximum number of parallel futures that can be active at the same time before blocking. If a function, it is called without arguments when the future is created and its value is used to configure the workers. The function should return a numeric scalar. Defaults to `future::availableCores()-1` if `NULL` (default), with `"multicore"` constraint in the relevant case. Ignored if `strategy=="sequential"`.

Value

The appropriate `get_doFuture_operator()` operator to use in a `foreach::foreach()` loop. The `%do%` operator is returned if `strategy=="sequential"`. Otherwise, the `%dopar%` operator is returned.

Examples

```
`%fun%` <- set_doFuture_strategy("multisession", n_workers=3)
# perform foreach::foreach loop using the %fun% operator
end_doFuture_strategy()
```

square_loss *Square loss*

Description

Square loss

Usage

```
square_loss(y, y_hat)
```

Arguments

y	Vector of observations or ground-truths.
y_hat	Vector of predictions.

Value

The vector of square errors between y and y_hat.

Examples

```
square_loss(c(2.3, 4.2, 1.8), c(2.2, 4.6, 1.7))
```

unconditional_train_valid_GPD_loss *Unconditional GPD MLEs and their train-validation likelihoods*

Description

Unconditional GPD MLEs and their train-validation likelihoods

Usage

```
unconditional_train_valid_GPD_loss(Y_train, interm_lvl, Y_valid)
```

Arguments

Y_train	Vector of "training" observations on which to estimate the MLEs.
interm_lvl	Probability level at which the empirical quantile should be used as the threshold.
Y_valid	Vector of "validation" observations, on which to estimate the out of training sample GPD loss.

Value

Named list containing:

`scale`	GPD scale MLE inferred from the train set,
`shape`	GPD shape MLE inferred from the train set,
`train_loss`	the negative log-likelihoods of the MLEs over the training samples,
`valid_loss`	the negative log-likelihoods of the MLEs over the validation samples.

`vec2mat` *Convert a vector to a matrix*

Description

Convert a vector to a matrix

Usage

```
vec2mat(v, axis = c("col", "row"))
```

Arguments

`v`	Vector.
`axis`	One of `"col"` (default) or `"row"`.

Value

The vector `v` as a matrix. If `axis=="col"` (default) the column vector `v` is returned as a `length(v)` times `1` matrix. If `axis=="row"`, the vector `v` is returned as a transposed `1` times `length(v)` matrix.

Examples

```
vec2mat(c(2, 7, 3, 8), "col")
```

`vector_insert` *Insert value in vector*

Description

Insert value in vector

Usage

```
vector_insert(vect, val, ind)
```

Arguments

`vect`	A 1-D vector.
`val`	A value to insert in the vector.
`ind`	The index at which to insert the value in the vector, must be an integer between `1` and `length(vect) + 1`.

Value

A 1-D vector of length `length(vect) + 1`, with `val` inserted at position `ind` in the original `vect`.

Examples

```
vector_insert(c(2, 7, 3, 8), val=5, ind=3)
```

Index