Package: mixvlmc (via r-universe)

September 4, 2024

Type Package

Title Variable Length Markov Chains with Covariates

Version 0.2.1.9000

Description Estimates Variable Length Markov Chains (VLMC) models and VLMC with covariates models from discrete sequences. Supports model selection via information criteria and simulation of new sequences from an estimated model. See Bühlmann, P. and Wyner, A. J. (1999) <doi:10.1214/aos/1018031204> for VLMC and Zanin Zambom, A., Kim, S. and Lopes Garcia, N. (2022) <doi:10.1111/jtsa.12615> for VLMC with covariates.

License GPL (>= 3)

URL https://github.com/fabrice-rossi/mixvlmc, https://fabrice-rossi.github.io/mixvlmc/

BugReports https://github.com/fabrice-rossi/mixvlmc/issues

Encoding UTF-8

LazyData true

Imports assertthat, butcher, cli, methods, nnet, pROC, Rcpp (>= 1.0.8.3), rlang, stats, stringr, VGAM, withr, xtable

LinkingTo Rcpp

RoxygenNote 7.3.1

Suggests data.table, foreach, geodist, ggplot2, knitr, rmarkdown, testthat (>= 3.0.0), tibble, vdiffr, waldo

Config/testthat/edition 3

Config/testthat/parallel true

Config/testthat/start-first covlmc*

Depends R (>= 2.10)

Roxygen list(markdown = TRUE)

VignetteBuilder knitr

Repository https://fabrice-rossi.r-universe.dev

RemoteUrl https://github.com/fabrice-rossi/mixvlmc

RemoteRef HEAD

RemoteSha 0b4e012f17b731c6a861300b91e6e0353226af56

Contents

mixvlmc-package *mixvlmc: Variable Length Markov Chains with Covariates*

Description

Estimates Variable Length Markov Chains (VLMC) models and VLMC with covariates models from discrete sequences. Supports model selection via information criteria and simulation of new sequences from an estimated model. See Bühlmann, P. and Wyner, A. J. (1999) doi:10.1214/aos/1018031204 for VLMC and Zanin Zambom, A., Kim, S. and Lopes Garcia, N. (2022) doi:10.1111/jtsa.12615 for VLMC with covariates.

Package options

Mixvlmc uses the following `options()`:

- `mixvlmc.maxit`: maximum number of iterations in model fitting for `covlmc()`
- `mixvlmc.predictive`: specifies the computing engine used for model fitting for `covlmc()`. Two values are supported:
  - `"glm"` (default value): `covlmc()` uses `stats::glm()` with a binomial link (`stats::binomial()`) for a two values state space, and `VGAM::vglm()` with a multinomial link (`VGAM::multinomial()`) for a state space with three or more values;
  - `"multinom"`: `covlmc()` uses `nnet::multinom()` in all cases.

  The first option `"glm"` is recommended as both `stats::glm()` and `VGAM::vglm()` are able to detect and deal with degeneracy in the data set.
- `mixvlmc.backend`: specifies the implementation used for the context tree construction in `ctx_tree()`, `vlmc()` and `tune_vlmc()`. Two values are supported:
  - `"R"` (default value): this corresponds to the original almost pure R implementation.
  - `"C++"`: this corresponds to the experimental C++ implementation. This version is significantly faster than the R version, but is still considered experimental.
- `mixvlmc.charset`: specifies the collection of characters used to display context trees in "ascii art" when using the `"text"` format for `draw()` and related functions. Two values are supported:
  - `"ascii"`: the collection uses only standard ASCII characters and should be compatible with all environments;
  - `"utf8"`: the collection uses UTF-8 symbols and needs a compatible display. At loading the option is set based on a call to `cli::is_utf8_output()`. It defaults to `"utf8"` is this encoding is supported.

Author(s)

Maintainer: Fabrice Rossi `<Fabrice.Rossi@apiacoa.org>` (ORCID) [copyright holder]

Other contributors:

- Hugo Le Picard `<lepicardhugo@gmail.com>` (ORCID) [contributor]
- Guénolé Joubioux `<guenole.joubioux@gmail.com>` [contributor]

See Also

Useful links:

- https://github.com/fabrice-rossi/mixvlmc
- https://fabrice-rossi.github.io/mixvlmc/
- Report bugs at https://github.com/fabrice-rossi/mixvlmc/issues

---

as_covlmc — *Convert an object to a Variable Length Markov Chain with covariates (coVLMC)*

Description

This generic function converts an object into a covlmc.

Usage

```
as_covlmc(x, ...)

## S3 method for class 'tune_covlmc'
as_covlmc(x, ...)
```

Arguments

`x`	an object to convert into a covlmc.
`...`	additional arguments for conversion functions.

Value

a covlmc

See Also

`tune_covlmc()`

Examples

```
## conversion from the results of tune_covlmc
pc <- powerconsumption[powerconsumption$week == 5, ]
rdts <- cut(pc$active_power, breaks = c(0, quantile(pc$active_power, probs = c(0.5, 1))))
rdts_cov <- data.frame(day_night = (pc$hour >= 7 & pc$hour <= 17))
rdts_best_model_tune <- tune_covlmc(rdts, rdts_cov)
rdts_best_model <- as_covlmc(rdts_best_model_tune)
draw(rdts_best_model)
```

as_sequence — *Extract the sequence encoded by a node*

Description

This function returns the sequence represented by the `node` object.

Usage

```
as_sequence(node, reverse)
```

Arguments

`node`	a `ctx_node` object as returned by `find_sequence()`
`reverse`	specifies whether the sequence should be reported in reverse temporal order (`TRUE`) or in the temporal order (`FALSE`). Defaults to the order associated to the `ctx_node` which is determined by the parameters of the call to `contexts()` or `find_sequence()`.

Value

the sequence represented by the `node` object, a vector

Examples

```
rdts <- c("A", "B", "C", "A", "A", "B", "B", "C", "C", "A")
rdts_tree <- ctx_tree(rdts, max_depth = 3)
res <- find_sequence(rdts_tree, "A")
as_sequence(res)
```

as_vlmc — *Convert an object to a Variable Length Markov Chain (VLMC)*

Description

This generic function converts an object into a vlmc.

Usage

```
as_vlmc(x, ...)

## S3 method for class 'ctx_tree'
as_vlmc(x, alpha, cutoff, ...)

## S3 method for class 'tune_vlmc'
as_vlmc(x, ...)
```

Arguments

`x`	an object to convert into a vlmc.
`...`	additional arguments for conversion functions.
`alpha`	cut off parameter applied during the conversion, quantile scale (if specified)
`cutoff`	cut off parameter applied during the conversion, native scale (if specified)

Details

This function converts a context tree into a VLMC. If `alpha` or `cutoff` is specified, it is used to reduce the complexity of the tree as in a direct call to `vlmc()` (`prune()`).

Value

a vlmc

See Also

`ctx_tree()`

`tune_vlmc()`

Examples

```
## conversion from a context tree
rdts <- c(0, 1, 1, 1, 0, 0, 1, 0, 1, 0)
rdts_ctree <- ctx_tree(rdts, min_size = 1, max_depth = 3)
draw(rdts_ctree)
rdts_vlmc <- as_vlmc(rdts_ctree)
class(rdts_vlmc)
draw(rdts_vlmc)
## conversion from the result of tune_vlmc
rdts <- sample(as.factor(c("A", "B", "C")), 100, replace = TRUE)
tune_result <- tune_vlmc(rdts)
tune_result
rdts_best_vlmc <- as_vlmc(tune_result)
draw(rdts_best_vlmc)
```

`as_vlmc.ctx_tree_cpp` *Convert an object to a Variable Length Markov Chain (VLMC)*

Description

This generic function converts an object into a vlmc.

Usage

```
## S3 method for class 'ctx_tree_cpp'
as_vlmc(x, alpha, cutoff, ...)
```

Arguments

`x`	an object to convert into a vlmc.
`alpha`	cut off parameter applied during the conversion, quantile scale (if specified)
`cutoff`	cut off parameter applied during the conversion, native scale (if specified)
`...`	additional arguments for conversion functions.

Details

This function converts a context tree into a VLMC. If `alpha` or `cutoff` is specified, it is used to reduce the complexity of the tree as in a direct call to `vlmc()` (`prune()`).

Value

a vlmc

See Also

`ctx_tree()`

`tune_vlmc()`

Examples

```
## conversion from a context tree
rdts <- c(0, 1, 1, 1, 0, 0, 1, 0, 1, 0)
rdts_ctree <- ctx_tree(rdts, min_size = 1, max_depth = 3, backend = "C++")
draw(rdts_ctree)
rdts_vlmc <- as_vlmc(rdts_ctree)
class(rdts_vlmc)
draw(rdts_vlmc)
```

---

autoplot.tune_covlmc *Create a complete ggplot for the results of automatic COVLMC complexity selection*

Description

This function prepares a plot of the results of `tune_covlmc()` using ggplot2. The result can be passed to `print()` to display the result.

Usage

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

Arguments

`object`	a `tune_covlmc` object
`...`	additional parameters (not used currently)

Details

The graphical representation proposed by this function is complete, while the one produced by `plot.tune_covlmc()` is minimalistic. We use here the faceting capabilities of ggplot2 to combine on a single graphical representation the evolution of multiple characteristics of the VLMC during the pruning process, while `plot.tune_covlmc()` shows only the selection criterion or the log likelihood. Each facet of the resulting plot shows a quantity as a function of the cut off expressed in quantile or native scale.

Value

a ggplot object

Examples

```
pc <- powerconsumption[powerconsumption$week %in% 10:12, ]
rdts <- cut(pc$active_power, breaks = c(0, quantile(pc$active_power, probs = c(0.5, 1))))
rdts_cov <- data.frame(day_night = (pc$hour >= 7 & pc$hour <= 17))
rdts_best_model_tune <- tune_covlmc(rdts, rdts_cov, criterion = "AIC")
covlmc_plot <- ggplot2::autoplot(rdts_best_model_tune)
print(covlmc_plot)
```

`autoplot.tune_vlmc` *Create a complete ggplot for the results of automatic VLMC complexity selection*

Description

This function prepares a plot of the results of `tune_vlmc()` using ggplot2. The result can be passed to `print()` to display the result.

Usage

```
## S3 method for class 'tune_vlmc'
autoplot(object, cutoff = c("quantile", "native"), ...)
```

Arguments

`object`	a `tune_vlmc` object
`cutoff`	the scale used for the cut off criterion (default "quantile")
`...`	additional parameters (not used currently)

Details

The graphical representation proposed by this function is complete, while the one produced by `plot.tune_vlmc()` is minimalistic. We use here the faceting capabilities of ggplot2 to combine on a single graphical representation the evolution of multiple characteristics of the VLMC during the pruning process, while `plot.tune_vlmc()` shows only the selection criterion or the log likelihood. Each facet of the resulting plot shows a quantity as a function of the cut off expressed in quantile or native scale.

Value

a ggplot object

Examples

```
pc <- powerconsumption[powerconsumption$week %in% 10:11, ]
rdts <- cut(pc$active_power, breaks = c(0, quantile(pc$active_power, probs = c(0.5, 1))))
rdts_best_model_tune <- tune_vlmc(rdts, criterion = "BIC")
vlmc_plot <- ggplot2::autoplot(rdts_best_model_tune)
print(vlmc_plot)
## simple post customisation
print(vlmc_plot + ggplot2::geom_point())
```

`charset_ascii` *ASCII character set for context tree text representation*

Description

This function returns a list of ASCII characters used to fine tune the `draw()` function behaviour when it is used with `format="text"`. It can be used as is or customised using its parameters.

Usage

```
charset_ascii(
  root = "*",
  first_node = "+",
  next_node = "'",
  final_node = "'",
  vbranch = "|",
  hbranch = "--",
  open_ct = "(",
  close_ct = ")",
  level_sep = " ~ ",
  time_sep = " | ",
  intercept = "(I)",
  intercept_sep = " & ",
  open_p_value = "<",
  close_p_value = ">",
```

```
  open_model = "[",
  close_model = "]"
)
```

Arguments

root	character used for the root node.
first_node	characters used for the first child of a node.
next_node	characters used for intermediate children of a node.
final_node	characters used for the last child of a node.
vbranch	characters used to represent a branch in a vertical way.
hbranch	characters used to represent a branch in a horizontal was.
open_ct	characters used to start each node specific text representation.
close_ct	characters used to end each node specific text representation.
level_sep	characters used to separate levels from models in draw.covlmc().
time_sep	characters used to separate temporal blocks in draw.covlmc().
intercept	characters used to represent the intercept in draw.covlmc().
intercept_sep	characters used to the intercept from the other parameters in draw.covlmc().
open_p_value	characters used as opening delimiters for the p value of a node in draw.covlmc().
close_p_value	characters used as closing delimiters for the p value of a node in draw.covlmc().
open_model	characters used as opening delimiters for the representation of a model in draw.covlmc().
close_model	characters used as closing delimiters for the representation of a model in draw.covlmc().

Value

a list

See Also

draw(), charset_utf8().

Examples

```
charset_ascii(root = "x")
```

charset_utf8 *UTF-8 character set for context tree text representation*

Description

This function returns a list of UTF-8 characters and symbols used to fine tune the `draw()` function behaviour when it is used with `format="text"`. It can be used as is or customised using its parameters.

Usage

```
charset_utf8(
  root = " ",
  first_node = " ",
  next_node = " ",
  final_node = " ",
  vbranch = " ",
  hbranch = " ",
  open_ct = "(",
  close_ct = ")",
  level_sep = " ~ ",
  time_sep = "   ",
  intercept = "(I)",
  intercept_sep = " • ",
  open_p_value = "<",
  close_p_value = ">",
  open_model = "[",
  close_model = "]"
)
```

Arguments

`root`	character used for the root node.
`first_node`	characters used for the first child of a node.
`next_node`	characters used for intermediate children of a node.
`final_node`	characters used for the last child of a node.
`vbranch`	characters used to represent a branch in a vertical way.
`hbranch`	characters used to represent a branch in a horizontal was.
`open_ct`	characters used to start each node specific text representation.
`close_ct`	characters used to end each node specific text representation.
`level_sep`	characters used to separate levels from models in `draw.covlmc()`.
`time_sep`	characters used to separate temporal blocks in `draw.covlmc()`.
`intercept`	characters used to represent the intercept in `draw.covlmc()`.
`intercept_sep`	characters used to the intercept from the other parameters in `draw.covlmc()`.

`open_p_value`	characters used as opening delimiters for the p value of a node in `draw.covlmc()`.
`close_p_value`	characters used as closing delimiters for the p value of a node in `draw.covlmc()`.
`open_model`	characters used as opening delimiters for the representation of a model in `draw.covlmc()`.
`close_model`	characters used as closing delimiters for the representation of a model in `draw.covlmc()`.

Value

a list

See Also

`draw()`, `charset_ascii()`.

Examples

```
charset_utf8(root = "\u27E1")
```

---

children — *Find the children nodes of a node in a context tree*

Description

This function returns a list (possibly empty) of `ctx_node` objects. Each object represents one of the children of the node represented by the `node` parameter.

Usage

```
children(node)

## S3 method for class 'ctx_node'
children(node)

## S3 method for class 'ctx_node_cpp'
children(node)
```

Arguments

`node`	a `ctx_node` object as returned by `find_sequence()`

Details

Each node of a context tree represents a sequence. When find_sequence() is called with success, the returned object represents the corresponding node in the context tree. If this node has no child, the present function returns an empty list. When the node has at least one child, the function returns a list with one value for each element in the state space (see states()). The value is NULL if the corresponding child is empty, while it is a ctx_node object when the child is present. Each ctx_node object is associated to the sequence obtained by adding to the past of the sequence represented by node an observation of the associated state (this corresponds to an extension to the left of the sequence in temporal order).

Value

a list of ctx_node objects, see details.

Examples

```
rdts <- c(0, 1, 1, 1, 0, 0, 1, 0, 1, 0)
rdts_ctree <- ctx_tree(rdts, min_size = 1, max_depth = 3)
ctx_00 <- find_sequence(rdts_ctree, c(0, 0))
## this context can only be extended in the past by 1:
children(ctx_00)
ctx_10 <- find_sequence(rdts_ctree, c(1, 0))
## this context can be extended by both states
children(ctx_10)
## C++ backend
rdts <- c(0, 1, 1, 1, 0, 0, 1, 0, 1, 0)
rdts_ctree <- ctx_tree(rdts, min_size = 1, max_depth = 3, backend = "C++")
ctx_00 <- find_sequence(rdts_ctree, c(0, 0))
## this context can only be extended in the past by 1:
children(ctx_00)
ctx_10 <- find_sequence(rdts_ctree, c(1, 0))
## this context can be extended by both states
children(ctx_10)
```

---

contexts *Contexts of a context tree*

Description

This function extracts from a context tree a description of all of its contexts.

Usage

```
contexts(ct, sequence = FALSE, reverse = FALSE, ...)
```

Arguments

`ct`	a context tree.
`sequence`	if `TRUE` the function returns its results as a `data.frame`, if `FALSE` (default) as a list of `ctx_node` objects. (see details)
`reverse`	logical (defaults to `FALSE`). See details.
`...`	additional arguments for the contexts function.

Details

The default behaviour consists in returning a list of all the contexts contained in the tree using `ctx_node` objects (as returned by e.g. `find_sequence()`) (with `type="list"`). The properties of the contexts can then be explored using adapted functions such as `counts()` and `positions()`. The result list is of class `contexts`. When `sequence=TRUE`, the method returns a data.frame whose first column, named `context`, contains the contexts as vectors (i.e. the value returned by `as_sequence()` applied to a `ctx_node` object). Other columns contain context specific values which depend on the actual class of the tree and on additional parameters. In all implementations of `contexts()`, setting the additional parameters to any no default value leads to a `data.frame` result.

Value

A list of class `contexts` containing the contexts represented in this tree (as `ctx_node`) or a data.frame.

State order in a context

Notice that contexts are given by default in the temporal order and not in the "reverse" order used by many VLMC research papers: older values are on the left. For instance, the context `c(1, 0)` is reported if the sequence 0, then 1 appeared in the time series used to build the context tree. Set reverse to `TRUE` for the reverse convention which is somewhat easier to relate to the way the context trees are represented by `draw()` (i.e. recent values at the top the tree).

See Also

`find_sequence()` and `find_sequence.covlmc()` for direct access to a specific context, and `contexts.ctx_tree()`, `contexts.vlmc()` and `contexts.covlmc()` for concrete implementations of `contexts()`.

Examples

```
rdts <- sample(as.factor(c("A", "B", "C")), 100, replace = TRUE)
rdts_tree <- ctx_tree(rdts, max_depth = 3, min_size = 5)
contexts(rdts_tree)
contexts(rdts_tree, TRUE, TRUE)
```

contexts.covlmc *Contexts of a VLMC with covariates*

Description

This function returns the different contexts present in a VLMC with covariates, possibly with some associated data.

Usage

```
## S3 method for class 'covlmc'
contexts(
  ct,
  sequence = FALSE,
  reverse = FALSE,
  frequency = NULL,
  positions = FALSE,
  local = FALSE,
  metrics = FALSE,
  model = NULL,
  hsize = FALSE,
  merging = FALSE,
  ...
)
```

Arguments

`ct` a fitted covlmc model.

`sequence` if `TRUE` the function returns its results as a `data.frame`, if `FALSE` (default) as a list of `ctx_node` objects. (see details)

`reverse` logical (defaults to `FALSE`). See details.

`frequency` specifies the counts to be included in the result data.frame. The default value of `NULL` does not include anything. `"total"` gives the number of occurrences of each context in the original sequence. `"detailed"` includes in addition the break down of these occurrences into all the possible states.

`positions` logical (defaults to FALSE). Specify whether the positions of each context in the time series used to build the context tree should be reported in a `positions` column of the result data frame. The availability of the positions depends on the way the context tree was built. See details for the definition of a position.

`local` specifies how the counts reported by `frequency` are computed. When `local` is `FALSE` (default value) the counts include both counts that are specific to the context (if any) and counts from the descendants of the context in the tree. When `local` is `TRUE` the counts include only the number of times the context appears without being the last part of a longer context.

`metrics` if TRUE, adds predictive metrics for each context (see `metrics()` for the definition of predictive metrics).

`model` specifies whether to include the model associated to a each context. The default result with `model=NULL` does not include any model. Setting `model` to `"coef"` adds the coefficients of the models in a `coef` column, while `"full"` include the models themselves (as R objects) in a `model` column.

`hsize` if TRUE, adds a `hsize` column to the result data frame that gives for each context the size of the history of covariates used by the model.

`merging` if TRUE, adds a `merged` column to the result data frame. For a normal context, the value of `merged` is FALSE. Contexts that share the same model have a TRUE `merged` value.

`...` additional arguments for the contexts function.

Details

The default behaviour of the function is to return a list of all the contexts using `ctx_node_covlmc` objects (as returned by `find_sequence.covlmc()`). The properties of the contexts can then be explored using adapted functions such as `counts()`, `covariate_memory()`, `cutoff.ctx_node()`, `metrics.ctx_node()`, `model()`, `merged_with()` and `positions()`.

When `sequence=TRUE` the method returns a data.frame whose first column, named `context`, contains the contexts as vectors (i.e. the value returned by `as_sequence()` applied to a `ctx_node` object). Other columns contain context specific values specified by the additional parameters. Setting any of those parameters to a value that ask for reporting information will toggle the result type of the function to `data.frame`.

See `contexts.ctx_tree()` for details about the `frequency` parameter. When `model` is non `NULL`, the resulting `data.frame` contains the models associated to each context (either the full R model or its coefficients). Other columns are added is the corresponding parameters are set to `TRUE`.

Value

A list of class `contexts` containing the contexts represented in this tree (as `ctx_node_covlmc`) or a data.frame.

Positions

A position of a context `ctx` in the time series `x` is an index value `t` such that the context ends with `x[t]`. Thus `x[t+1]` is after the context. For instance if `x=c(0, 0, 1, 1)` and `ctx=c(0, 1)` (in standard state order), then the position of `ctx` in `x` is 3.

State order in a context

Notice that contexts are given by default in the temporal order and not in the "reverse" order used by many VLMC research papers: older values are on the left. For instance, the context `c(1, 0)` is reported if the sequence 0, then 1 appeared in the time series used to build the context tree. Set reverse to `TRUE` for the reverse convention which is somewhat easier to relate to the way the context trees are represented by `draw()` (i.e. recent values at the top the tree).

See Also

find_sequence() and find_sequence.covlmc() for direct access to a specific context, and contexts.ctx_tree(), contexts.vlmc() and contexts.covlmc() for concrete implementations of contexts().

Examples

```
pc <- powerconsumption[powerconsumption$week == 5, ]
breaks <- c(0, median(pc$active_power), max(pc$active_power))
dts <- cut(pc$active_power, breaks = breaks)
dts_cov <- data.frame(day_night = (pc$hour >= 7 & pc$hour <= 17))
m_cov <- covlmc(dts, dts_cov, min_size = 5)
## direct representation with ctx_node_covlmc objects
m_cov_ctxs <- contexts(m_cov)
m_cov_ctxs
sapply(m_cov_ctxs, covariate_memory)
sapply(m_cov_ctxs, is_merged)
sapply(m_cov_ctxs, model)
## data.frame interface
contexts(m_cov, model = "coef")
contexts(m_cov, model = "full", hsize = TRUE)
```

---

contexts.ctx_tree *Contexts of a context tree*

Description

This function extracts from a context tree a description of all of its contexts.

Usage

```
## S3 method for class 'ctx_tree'
contexts(
  ct,
  sequence = FALSE,
  reverse = FALSE,
  frequency = NULL,
  positions = FALSE,
  ...
)

## S3 method for class 'ctx_tree_cpp'
contexts(
  ct,
  sequence = FALSE,
  reverse = FALSE,
  frequency = NULL,
  positions = FALSE,
```

```
  ...
)
```

Arguments

`ct`	a context tree.
`sequence`	if `TRUE` the function returns its results as a `data.frame`, if `FALSE` (default) as a list of `ctx_node` objects. (see details)
`reverse`	logical (defaults to `FALSE`). See details.
`frequency`	specifies the counts to be included in the result data.frame. The default value of `NULL` does not include anything. `"total"` gives the number of occurrences of each context in the original sequence. `"detailed"` includes in addition the break down of these occurrences into all the possible states.
`positions`	logical (defaults to FALSE). Specify whether the positions of each context in the time series used to build the context tree should be reported in a `positions` column of the result data frame. The availability of the positions depends on the way the context tree was built. See details for the definition of a position.
`...`	additional arguments for the contexts function.

Details

The default behaviour of the function is to return a list of all the contexts using `ctx_node` objects (as returned by `find_sequence()`). The properties of the contexts can then be explored using adapted functions such as `counts()` and `positions()`.

When `sequence=TRUE` the method returns a data.frame whose first column, named `context`, contains the contexts as vectors (i.e. the value returned by `as_sequence()` applied to a `ctx_node` object). Other columns contain context specific values specified by the additional parameters. Setting any of those parameters to a value that ask for reporting information will toggle the result type of the function to `data.frame`.

If `frequency="total"`, an additional column named `freq` gives the number of occurrences of each context in the series used to build the tree. If `frequency="detailed"`, one additional column is added per state in the context space. Each column records the number of times a given context is followed by the corresponding value in the original series.

Value

A list of class `contexts` containing the contexts represented in this tree (as `ctx_node`) or a data.frame.

Positions

A position of a context `ctx` in the time series `x` is an index value `t` such that the context ends with `x[t]`. Thus `x[t+1]` is after the context. For instance if `x=c(0, 0, 1, 1)` and `ctx=c(0, 1)` (in standard state order), then the position of `ctx` in `x` is 3.

State order in a context

Notice that contexts are given by default in the temporal order and not in the "reverse" order used by many VLMC research papers: older values are on the left. For instance, the context `c(1, 0)` is reported if the sequence 0, then 1 appeared in the time series used to build the context tree. Set reverse to `TRUE` for the reverse convention which is somewhat easier to relate to the way the context trees are represented by `draw()` (i.e. recent values at the top the tree).

See Also

`find_sequence()` and `find_sequence.covlmc()` for direct access to a specific context, and `contexts.ctx_tree()`, `contexts.vlmc()` and `contexts.covlmc()` for concrete implementations of `contexts()`.

Examples

```
rdts <- sample(as.factor(c("A", "B", "C")), 100, replace = TRUE)
rdts_tree <- ctx_tree(rdts, max_depth = 3, min_size = 5)
## direct representation with ctx_node objects
contexts(rdts_tree)
## data.frame format
contexts(rdts_tree, sequence = TRUE)
contexts(rdts_tree, frequency = "total")
contexts(rdts_tree, frequency = "detailed")
```

---

contexts.vlmc *Contexts of a VLMC*

Description

This function extracts all the contexts from a fitted VLMC, possibly with some associated data.

Usage

```
## S3 method for class 'vlmc'
contexts(
  ct,
  sequence = FALSE,
  reverse = FALSE,
  frequency = NULL,
  positions = FALSE,
  local = FALSE,
  cutoff = NULL,
  metrics = FALSE,
  ...
)
```

```
## S3 method for class 'vlmc_cpp'
contexts(
  ct,
  sequence = FALSE,
  reverse = FALSE,
  frequency = NULL,
  positions = FALSE,
  local = FALSE,
  cutoff = NULL,
  metrics = FALSE,
  ...
)
```

Arguments

`ct`	a context tree.
`sequence`	if `TRUE` the function returns its results as a `data.frame`, if `FALSE` (default) as a list of `ctx_node` objects. (see details)
`reverse`	logical (defaults to `FALSE`). See details.
`frequency`	specifies the counts to be included in the result data.frame. The default value of `NULL` does not include anything. `"total"` gives the number of occurrences of each context in the original sequence. `"detailed"` includes in addition the break down of these occurrences into all the possible states.
`positions`	logical (defaults to FALSE). Specify whether the positions of each context in the time series used to build the context tree should be reported in a `positions` column of the result data frame. The availability of the positions depends on the way the context tree was built. See details for the definition of a position.
`local`	specifies how the counts reported by `frequency` are computed. When `local` is `FALSE` (default value) the counts include both counts that are specific to the context (if any) and counts from the descendants of the context in the tree. When `local` is `TRUE` the counts include only the number of times the context appears without being the last part of a longer context.
`cutoff`	specifies whether to include the cut off value associated to each context (see `cutoff()` and `prune()`). The default result with `cutoff=NULL` does not include those values. Setting `cutoff` to `quantile` adds the cut off values in quantile scale, while `cutoff="native"` adds them in the native scale. The returned values are directly based on the log likelihood ratio computed in the context tree and are not modified to ensure pruning (as when `cutoff()` is called by `raw=TRUE`).
`metrics`	if TRUE, adds predictive metrics for each context (see `metrics()` for the definition of predictive metrics).
`...`	additional arguments for the contexts function.

Details

The default behaviour of the function is to return a list of all the contexts using `ctx_node` objects (as returned by `find_sequence()`). The properties of the contexts can then be explored using adapted functions such as `counts()`, `cutoff.ctx_node()`, `metrics.ctx_node()` and `positions()`.

When `sequence=TRUE` the method returns a data.frame whose first column, named `context`, contains the contexts as vectors (i.e. the value returned by `as_sequence()` applied to a `ctx_node` object). Other columns contain context specific values specified by the additional parameters. Setting any of those parameters to a value that ask for reporting information will toggle the result type of the function to `data.frame`.

The `frequency` parameter is described in details in the documentation of `contexts.ctx_tree()`. When `cutoff` is non `NULL`, the resulting `data.frame` contains a `cutoff` column with the cut off values, either in quantile or in native scale. See `cutoff.vlmc()` and `prune.vlmc()` for the definitions of cut off values and of the two scales.

Value

A list of class `contexts` containing the contexts represented in this tree (as `ctx_node`) or a data.frame.

Cut off values

The cut off values reported by `contexts.vlmc` can be different from the ones reported by `cutoff.vlmc()` for three reasons:

1. `cutoff.vlmc()` reports only useful cut off values, i.e., cut off values that should induce a simplification of the VLMC when used in `prune()`. This exclude cut off values associated to simple contexts that are smaller than the ones of their descendants in the context tree. Those values are reported by `context.vlmc`.
2. `context.vlmc` reports only cut off values of actual contexts, while `cutoff.vlmc()` reports cut off values for all nodes of the context tree.
3. values are not modified to induce pruning, contrarily to the default behaviour of `cutoff.vlmc()`

Positions

A position of a context `ctx` in the time series `x` is an index value `t` such that the context ends with `x[t]`. Thus `x[t+1]` is after the context. For instance if `x=c(0, 0, 1, 1)` and `ctx=c(0, 1)` (in standard state order), then the position of `ctx` in `x` is 3.

State order in a context

Notice that contexts are given by default in the temporal order and not in the "reverse" order used by many VLMC research papers: older values are on the left. For instance, the context `c(1, 0)` is reported if the sequence 0, then 1 appeared in the time series used to build the context tree. Set reverse to `TRUE` for the reverse convention which is somewhat easier to relate to the way the context trees are represented by `draw()` (i.e. recent values at the top the tree).

See Also

find_sequence() and find_sequence.covlmc() for direct access to a specific context, and contexts.ctx_tree(), contexts.vlmc() and contexts.covlmc() for concrete implementations of contexts().

Examples

```
rdts <- sample(as.factor(c("A", "B", "C")), 100, replace = TRUE)
model <- vlmc(rdts, alpha = 0.5)
## direct representation with ctx_node objects
model_ctxs <- contexts(model)
model_ctxs
sapply(model_ctxs, cutoff, scale = "quantile")
sapply(model_ctxs, cutoff, scale = "native")
sapply(model_ctxs, function(x) metrics(x)$accuracy)
## data.frame format
contexts(model, frequency = "total")
contexts(model, cutoff = "quantile")
contexts(model, cutoff = "native", metrics = TRUE)
```

---

context_number   *Number of contexts of a context tree*

Description

This function returns the number of distinct contexts in a context tree.

Usage

```
context_number(ct)
```

Arguments

ct	a context tree.

Value

the number of contexts of the tree.

Examples

```
rdts <- c(0, 1, 1, 1, 0, 0, 1, 0, 1, 0)
rdts_ctree <- ctx_tree(rdts, min_size = 1, max_depth = 3)
# should be 8
context_number(rdts_ctree)
```

`context_number.covlmc` *Contexts number of a VLMC with covariates*

Description

This function returns the total number of contexts of a VLMC with covariates.

Usage

```
## S3 method for class 'covlmc'
context_number(ct)
```

Arguments

`ct`	a fitted covlmc model.

Value

the number of contexts present in the VLMC with covariates.

Examples

```
pc <- powerconsumption[powerconsumption$week == 5, ]
dts <- cut(pc$active_power, breaks = c(0, quantile(pc$active_power, probs = c(0.5, 1))))
dts_cov <- data.frame(day_night = (pc$hour >= 7 & pc$hour <= 17))
m_cov <- covlmc(dts, dts_cov, min_size = 10)
# should be 3
context_number(m_cov)
```

`counts` *Report the distribution of values that follow occurrences of a sequence*

Description

This function reports the number of occurrences of the sequence represented by `node` in the original time series used to build the associated context tree (not including a possible final occurrence not followed by any value at the end of the original time series). In addition if `frequency=="detailed"`, the function reports the frequencies of each of the possible value of the time series when they appear just after the sequence.

Usage

```
counts(node, frequency = c("detailed", "total"), local = FALSE)

## S3 method for class 'ctx_node'
counts(node, frequency = c("detailed", "total"), local = FALSE)

## S3 method for class 'ctx_node_cpp'
counts(node, frequency = c("detailed", "total"), local = FALSE)
```

Arguments

`node`	a `ctx_node` object as returned by `find_sequence()`
`frequency`	specifies the counts to be included in the result. `"total"` gives the number of occurrences of the sequence in the original sequence. `"detailed"` includes in addition the break down of these occurrences into all the possible states.
`local`	specifies how the counts are computed. When `local` is `FALSE` (default value) the counts include both counts that are specific to the context (if any) and counts from the descendants of the context in the tree. When `local` is `TRUE` the counts include only the number of times the context appears without being the last part of a longer context.

Value

either an integer when `frequency="total"` which gives the total number of occurrences of the sequence represented by `node` or a `data.frame` with a `total` column with the same value and a column for each of the possible value of the original time series, reporting counts in each column (see the description above).

See Also

`contexts()` and `contexts.ctx_tree()`

Examples

```
rdts <- sample(as.factor(c("A", "B", "C")), 100, replace = TRUE)
rdts_tree <- ctx_tree(rdts, max_depth = 3, min_size = 5)
subseq <- find_sequence(rdts_tree, factor(c("A", "A"), levels = c("A", "B", "C")))
if (!is.null(subseq)) {
  counts(subseq)
}
```

covariate_depth *Maximal covariate memory of a VLMC with covariates*

Description

This function return the longest covariate memory used by a VLMC with covariates.

Usage

```
covariate_depth(model)
```

Arguments

`model`	a covlmc object

Value

the longest covariate memory of this model

Examples

```
pc <- powerconsumption[powerconsumption$week == 5, ]
rdts <- cut(pc$active_power, breaks = c(0, quantile(pc$active_power, probs = c(0.5, 1))))
m_nocovariate <- vlmc(rdts)
rdts_cov <- data.frame(day_night = (pc$hour >= 7 & pc$hour <= 17))
m_cov <- covlmc(rdts, rdts_cov, min_size = 10)
covariate_depth(m_cov)
```

covariate_memory *Covariate memory length for a COVLMC context*

Description

This function returns the length of the memory of a COVLMC context represented by a `ctx_node_covlmc` object.

Usage

```
covariate_memory(node)
```

Arguments

`node`	A `ctx_node_covlmc` object as returned by `find_sequence()` or `contexts.covlmc()`

Value

the memory length, an integer

Examples

```
pc <- powerconsumption[powerconsumption$week == 5, ]
rdts <- cut(pc$active_power, breaks = c(0, quantile(pc$active_power, probs = c(0.5, 1))))
rdts_cov <- data.frame(day_night = (pc$hour >= 7 & pc$hour <= 17))
m_cov <- covlmc(rdts, rdts_cov, min_size = 10)
ctxs <- contexts(m_cov)
## get all the memory lengths
sapply(ctxs, covariate_memory)
```

---

covlmc — *Fit a Variable Length Markov Chain with Covariates (coVLMC)*

Description

This function fits a Variable Length Markov Chain with covariates (coVLMC) to a discrete time series coupled with a time series of covariates.

Usage

```
covlmc(
  x,
  covariate,
  alpha = 0.05,
  min_size = 5L,
  max_depth = 100L,
  keep_data = TRUE,
  control = covlmc_control(...),
  ...
)
```

Arguments

`x`	an object that can be interpreted as a discrete time series, such as an integer vector or a `dts` object (see `dts()`)
`covariate`	a data frame of covariates.
`alpha`	number in (0,1) (default: 0.05) cut off value in the pruning phase (in quantile scale).
`min_size`	number >= 1 (default: 5). Tune the minimum number of observations for a context in the growing phase of the context tree (see below for details).
`max_depth`	integer >= 1 (default: 100). Longest context considered in growing phase of the context tree.
`keep_data`	logical (defaults to `TRUE`). If `TRUE`, the original data are stored in the resulting object to enable post pruning (see `prune.covlmc()`).
`control`	a list with control parameters, see `covlmc_control()`.
`...`	arguments passed to `covlmc_control()`.

Details

The model is built using the algorithm described in Zanin Zambom et al. As for the `vlmc()` approach, the algorithm builds first a context tree (see `ctx_tree()`). The `min_size` parameter is used to compute the actual number of observations per context in the growing phase of the tree. It is computed as `min_size*(1+ncol(covariate)*d)*(s-1)` where `d` is the length of the context (a.k.a. the depth in the tree) and `s` is the number of states. This corresponds to ensuring min_size observations per parameter of the logistic regression during the estimation phase.

Then logistic models are adjusted in the leaves at the tree: the goal of each logistic model is to estimate the conditional distribution of the next state of the times series given the context (the recent past of the time series) and delayed versions of the covariates. A pruning strategy is used to simplified the models (mainly to reduce the time window associated to the covariates) and the tree itself.

Parameters specified by `control` are used to fine tune the behaviour of the algorithm.

Value

a fitted covlmc model.

Logistic models

By default, `covlmc` uses two different computing *engines* for logistic models:

- when the time series has only two states, `covlmc` uses `stats::glm()` with a binomial link (`stats::binomial()`);
- when the time series has at least three states, `covlmc` use `VGAM::vglm()` with a multinomial link (`VGAM::multinomial()`).

Both engines are able to detect degenerate cases and lead to more robust results that using `nnet::multinom()`. It is nevertheless possible to replace `stats::glm()` and `VGAM::vglm()` with `nnet::multinom()` by setting the global option `mixvlmc.predictive` to `"multinom"` (the default value is `"glm"`). Notice that while results should be comparable, there is no guarantee that they will be identical.

References

- Bühlmann, P. and Wyner, A. J. (1999), "Variable length Markov chains." Ann. Statist. 27 (2) 480-513 doi:10.1214/aos/1018031204
- Zanin Zambom, A., Kim, S. and Lopes Garcia, N. (2022), "Variable length Markov chain with exogenous covariates." J. Time Ser. Anal., 43 (2) 312-328 doi:10.1111/jtsa.12615

See Also

`cutoff.covlmc()` and `prune.covlmc()` for post-pruning.

Examples

```
pc <- powerconsumption[powerconsumption$week == 5, ]
rdts <- cut(pc$active_power, breaks = c(0, quantile(pc$active_power,
  probs = c(1 / 3, 2 / 3, 1)
)))
rdts_cov <- data.frame(day_night = (pc$hour >= 7 & pc$hour <= 17))
m_cov <- covlmc(rdts, rdts_cov, min_size = 15)
draw(m_cov)
withr::with_options(
  list(mixvlmc.predictive = "multinom"),
  m_cov_nnet <- covlmc(rdts, rdts_cov, min_size = 15)
)
draw(m_cov_nnet)
```

`covlmc.default` *Fit a Variable Length Markov Chain with Covariates (coVLMC)*

Description

This function fits a Variable Length Markov Chain with covariates (coVLMC) to a discrete time series coupled with a time series of covariates.

Usage

```
## Default S3 method:
covlmc(
  x,
  covariate,
  alpha = 0.05,
  min_size = 5L,
  max_depth = 100L,
  keep_data = TRUE,
  control = covlmc_control(...),
  ...
)
```

Arguments

`x`	a numeric, character, factor or logical vector
`covariate`	a data frame of covariates.
`alpha`	number in (0,1) (default: 0.05) cut off value in the pruning phase (in quantile scale).
`min_size`	number >= 1 (default: 5). Tune the minimum number of observations for a context in the growing phase of the context tree (see below for details).
`max_depth`	integer >= 1 (default: 100). Longest context considered in growing phase of the context tree.

keep_data	logical (defaults to TRUE). If TRUE, the original data are stored in the resulting object to enable post pruning (see prune.covlmc()).
control	a list with control parameters, see covlmc_control().
...	arguments passed to covlmc_control().

Details

The model is built using the algorithm described in Zanin Zambom et al. As for the vlmc() approach, the algorithm builds first a context tree (see ctx_tree()). The min_size parameter is used to compute the actual number of observations per context in the growing phase of the tree. It is computed as min_size*(1+ncol(covariate)*d)*(s-1) where d is the length of the context (a.k.a. the depth in the tree) and s is the number of states. This corresponds to ensuring min_size observations per parameter of the logistic regression during the estimation phase.

Then logistic models are adjusted in the leaves at the tree: the goal of each logistic model is to estimate the conditional distribution of the next state of the times series given the context (the recent past of the time series) and delayed versions of the covariates. A pruning strategy is used to simplified the models (mainly to reduce the time window associated to the covariates) and the tree itself.

Parameters specified by control are used to fine tune the behaviour of the algorithm.

Value

a fitted covlmc model.

Logistic models

By default, covlmc uses two different computing *engines* for logistic models:

- when the time series has only two states, covlmc uses stats::glm() with a binomial link (stats::binomial());
- when the time series has at least three states, covlmc use VGAM::vglm() with a multinomial link (VGAM::multinomial()).

Both engines are able to detect degenerate cases and lead to more robust results that using nnet::multinom(). It is nevertheless possible to replace stats::glm() and VGAM::vglm() with nnet::multinom() by setting the global option mixvlmc.predictive to "multinom" (the default value is "glm"). Notice that while results should be comparable, there is no guarantee that they will be identical.

References

- Bühlmann, P. and Wyner, A. J. (1999), "Variable length Markov chains." Ann. Statist. 27 (2) 480-513 doi:10.1214/aos/1018031204
- Zanin Zambom, A., Kim, S. and Lopes Garcia, N. (2022), "Variable length Markov chain with exogenous covariates." J. Time Ser. Anal., 43 (2) 312-328 doi:10.1111/jtsa.12615

See Also

cutoff.covlmc() and prune.covlmc() for post-pruning.

Examples

```
pc <- powerconsumption[powerconsumption$week == 5, ]
rdts <- cut(pc$active_power, breaks = c(0, quantile(pc$active_power,
  probs = c(1 / 3, 2 / 3, 1)
)))
rdts_cov <- data.frame(day_night = (pc$hour >= 7 & pc$hour <= 17))
m_cov <- covlmc(rdts, rdts_cov, min_size = 15)
draw(m_cov)
withr::with_options(
  list(mixvlmc.predictive = "multinom"),
  m_cov_nnet <- covlmc(rdts, rdts_cov, min_size = 15)
)
draw(m_cov_nnet)
```

---

covlmc.dts — *Fit a Variable Length Markov Chain with Covariates (coVLMC)*

Description

This function fits a Variable Length Markov Chain with covariates (coVLMC) to a discrete time series coupled with a time series of covariates.

Usage

```
## S3 method for class 'dts'
covlmc(
  x,
  covariate,
  alpha = 0.05,
  min_size = 5L,
  max_depth = 100L,
  keep_data = TRUE,
  control = covlmc_control(...),
  ...
)
```

Arguments

`x`	a discrete time series represented by a `dts` object as created by `dts()`
`covariate`	a data frame of covariates.
`alpha`	number in (0,1) (default: 0.05) cut off value in the pruning phase (in quantile scale).

`min_size`	number >= 1 (default: 5). Tune the minimum number of observations for a context in the growing phase of the context tree (see below for details).
`max_depth`	integer >= 1 (default: 100). Longest context considered in growing phase of the context tree.
`keep_data`	logical (defaults to `TRUE`). If `TRUE`, the original data are stored in the resulting object to enable post pruning (see `prune.covlmc()`).
`control`	a list with control parameters, see `covlmc_control()`.
`...`	arguments passed to `covlmc_control()`.

Details

The model is built using the algorithm described in Zanin Zambom et al. As for the `vlmc()` approach, the algorithm builds first a context tree (see `ctx_tree()`). The `min_size` parameter is used to compute the actual number of observations per context in the growing phase of the tree. It is computed as `min_size*(1+ncol(covariate)*d)*(s-1)` where `d` is the length of the context (a.k.a. the depth in the tree) and `s` is the number of states. This corresponds to ensuring min_size observations per parameter of the logistic regression during the estimation phase.

Then logistic models are adjusted in the leaves at the tree: the goal of each logistic model is to estimate the conditional distribution of the next state of the times series given the context (the recent past of the time series) and delayed versions of the covariates. A pruning strategy is used to simplified the models (mainly to reduce the time window associated to the covariates) and the tree itself.

Parameters specified by `control` are used to fine tune the behaviour of the algorithm.

Value

a fitted covlmc model.

Logistic models

By default, `covlmc` uses two different computing *engines* for logistic models:

- when the time series has only two states, `covlmc` uses `stats::glm()` with a binomial link (`stats::binomial()`);
- when the time series has at least three states, `covlmc` use `VGAM::vglm()` with a multinomial link (`VGAM::multinomial()`).

Both engines are able to detect degenerate cases and lead to more robust results that using `nnet::multinom()`. It is nevertheless possible to replace `stats::glm()` and `VGAM::vglm()` with `nnet::multinom()` by setting the global option `mixvlmc.predictive` to `"multinom"` (the default value is `"glm"`). Notice that while results should be comparable, there is no guarantee that they will be identical.

References

- Bühlmann, P. and Wyner, A. J. (1999), "Variable length Markov chains." Ann. Statist. 27 (2) 480-513 doi:10.1214/aos/1018031204

- Zanin Zambom, A., Kim, S. and Lopes Garcia, N. (2022), "Variable length Markov chain with exogenous covariates." J. Time Ser. Anal., 43 (2) 312-328 doi:10.1111/jtsa.12615

See Also

cutoff.covlmc() and prune.covlmc() for post-pruning.

Examples

```
pc <- powerconsumption[powerconsumption$week == 5, ]
power_dts <- dts(cut(pc$active_power, breaks = c(0, quantile(pc$active_power,
  probs = c(1 / 3, 2 / 3, 1)
))))
power_cov <- data.frame(day_night = (pc$hour >= 7 & pc$hour <= 17))
m_cov <- covlmc(power_dts, power_cov, min_size = 15)
draw(m_cov)
```

---

covlmc_control *Control for coVLMC fitting*

Description

This function creates a list with parameters used to fine tune the coVLMC fitting algorithm.

Usage

```
covlmc_control(pseudo_obs = 1)
```

Arguments

pseudo_obs	number of fake observations of each state to add to the observed ones.

Details

`pseudo_obs` is used to regularize the probability estimations when a context is only observed followed by always the same state. Transition probabilities are computed after adding `pseudo_obs` pseudo observations of each of the states (including the observed one). This corresponds to a Bayesian posterior mean estimation with a Dirichlet prior.

Value

a list.

Examples

```
rdts <- rep(c(0, 1), 100)
rdts_cov <- data.frame(y = rep(0, length(rdts)))
default_model <- covlmc(rdts, rdts_cov)
contexts(default_model, type = "data.frame", model = "coef")$coef
control <- covlmc_control(pseudo_obs = 10)
model <- covlmc(rdts, rdts_cov, control = control)
contexts(model, type = "data.frame", model = "coef")$coef
```

---

ctx_tree *Build a context tree for a discrete time series*

Description

This function builds a context tree for a time series.

Usage

```
ctx_tree(
  x,
  min_size = 2L,
  max_depth = 100L,
  keep_position = TRUE,
  backend = getOption("mixvlmc.backend", "R"),
  ...
)
```

Arguments

`x`	an object that can be interpreted as a discrete time series, such as an integer vector or a `dts` object (see `dts()`)
`min_size`	integer >= 1 (default: 2). Minimum number of observations for a context to be included in the tree.
`max_depth`	integer >= 1 (default: 100). Maximum length of a context to be included in the tree.
`keep_position`	logical (default: TRUE). Should the context tree keep the position of the contexts.
`backend`	"R" or "C++" (default: as specified by the "mixvlmc.backend" option). Specifies the implementation used to represent the context tree and to built it. See details.
`...`	additional parameters

Details

The tree represents all the sequences of symbols/states of length smaller than `max_depth` that appear at least `min_size` times in the time series and stores the frequencies of the states that follow each context. Optionally, the positions of the contexts in the time series can be stored in the tree.

Value

a context tree (of class that inherits from `ctx_tree`).

Back ends

Two back ends are available to compute context trees:

- the "R" back end represents the tree in pure R data structures (nested lists) that be easily processed further in pure R (C++ helper functions are used to speed up the construction).
- the "C++" back end represents the tree with C++ classes. This back end is considered experimental. The tree is built with an optimised suffix tree algorithm which speeds up the construction by at least a factor 10 in standard settings. As the tree is kept outside of R direct reach, context trees built with the C++ back end must be restored after a `saveRDS()`/`readRDS()` sequence. This is done automatically by recomputing completely the context tree.

Examples

```
rdts <- c(0, 1, 1, 1, 0, 0, 1, 0, 1, 0)
## get all contexts of length 2
rdts_ctree <- ctx_tree(rdts, min_size = 1, max_depth = 2)
draw(rdts_ctree)
```

`ctx_tree.default` *Build a context tree for a discrete time series*

Description

This function builds a context tree for a time series.

Usage

```
## Default S3 method:
ctx_tree(
  x,
  min_size = 2L,
  max_depth = 100L,
  keep_position = TRUE,
  backend = getOption("mixvlmc.backend", "R"),
  ...
)
```

Arguments

`x`	a numeric, character, factor or logical vector
`min_size`	integer >= 1 (default: 2). Minimum number of observations for a context to be included in the tree.
`max_depth`	integer >= 1 (default: 100). Maximum length of a context to be included in the tree.
`keep_position`	logical (default: TRUE). Should the context tree keep the position of the contexts.
`backend`	"R" or "C++" (default: as specified by the "mixvlmc.backend" option). Specifies the implementation used to represent the context tree and to built it. See details.
`...`	additional parameters

Details

The tree represents all the sequences of symbols/states of length smaller than `max_depth` that appear at least `min_size` times in the time series and stores the frequencies of the states that follow each context. Optionally, the positions of the contexts in the time series can be stored in the tree.

Value

a context tree (of class that inherits from `ctx_tree`).

Back ends

Two back ends are available to compute context trees:

- the "R" back end represents the tree in pure R data structures (nested lists) that be easily processed further in pure R (C++ helper functions are used to speed up the construction).
- the "C++" back end represents the tree with C++ classes. This back end is considered experimental. The tree is built with an optimised suffix tree algorithm which speeds up the construction by at least a factor 10 in standard settings. As the tree is kept outside of R direct reach, context trees built with the C++ back end must be restored after a `saveRDS()/readRDS()` sequence. This is done automatically by recomputing completely the context tree.

Examples

```
rdts <- c(0, 1, 1, 1, 0, 0, 1, 0, 1, 0)
## get all contexts of length 2
rdts_ctree <- ctx_tree(rdts, min_size = 1, max_depth = 2)
draw(rdts_ctree)
```

`ctx_tree.dts` *Build a context tree for a discrete time series*

Description

This function builds a context tree for a time series.

Usage

```
## S3 method for class 'dts'
ctx_tree(
  x,
  min_size = 2L,
  max_depth = 100L,
  keep_position = TRUE,
  backend = getOption("mixvlmc.backend", "R"),
  ...
)
```

Arguments

`x`	a discrete time series represented by a `dts` object as created by `dts()`
`min_size`	integer >= 1 (default: 2). Minimum number of observations for a context to be included in the tree.
`max_depth`	integer >= 1 (default: 100). Maximum length of a context to be included in the tree.
`keep_position`	logical (default: TRUE). Should the context tree keep the position of the contexts.
`backend`	"R" or "C++" (default: as specified by the "mixvlmc.backend" option). Specifies the implementation used to represent the context tree and to built it. See details.
`...`	additional parameters

Details

The tree represents all the sequences of symbols/states of length smaller than `max_depth` that appear at least `min_size` times in the time series and stores the frequencies of the states that follow each context. Optionally, the positions of the contexts in the time series can be stored in the tree.

Value

a context tree (of class that inherits from `ctx_tree`).

Back ends

Two back ends are available to compute context trees:

- the "R" back end represents the tree in pure R data structures (nested lists) that be easily processed further in pure R (C++ helper functions are used to speed up the construction).
- the "C++" back end represents the tree with C++ classes. This back end is considered experimental. The tree is built with an optimised suffix tree algorithm which speeds up the construction by at least a factor 10 in standard settings. As the tree is kept outside of R direct reach, context trees built with the C++ back end must be restored after a `saveRDS()`/`readRDS()` sequence. This is done automatically by recomputing completely the context tree.

Examples

```
x_dts <- dts(c(0, 1, 1, 1, 0, 0, 1, 0, 1, 0))
## get all contexts of length 2
ctree <- ctx_tree(x_dts, min_size = 1, max_depth = 2)
draw(ctree)
```

`cutoff` *Cut off values for VLMC like model*

Description

This generic function returns one or more cut off values that are guaranteed to have an effect on the `model` passed to the function when a simplification procedure is applied (in general a tree pruning operation as provided by `prune()`).

Usage

```
cutoff(model, ...)
```

Arguments

`model`	a model.
`...`	additional arguments for the cutoff function implementations

Details

The exact definition of what is a cut off value depends on the model type and is documented in concrete implementation of the function.

Value

a cut off value or a vector of cut off values.

See Also

prune()

Examples

```
pc <- powerconsumption[powerconsumption$week == 5, ]
rdts <- cut(pc$active_power,
  breaks = c(0, quantile(pc$active_power,
    probs = c(0.25, 0.5, 0.75, 1)
  ))
)
model <- vlmc(rdts)
draw(model)
model_cuts <- cutoff(model)
model_2 <- prune(model, model_cuts[2])
draw(model_2)
```

---

cutoff.covlmc — *Cut off values for pruning the context tree of a VLMC with covariates*

Description

This function returns all the cut off values that should induce a pruning of the context tree of a VLMC with covariates.

Usage

```
## S3 method for class 'covlmc'
cutoff(model, raw = FALSE, tolerance = .Machine$double.eps^0.5, ...)
```

Arguments

`model`	a fitted COVLMC model.
`raw`	specify whether the returned values should be limit values computed in the model or modified values that guarantee pruning (see details)
`tolerance`	specify the minimum separation between two consecutive values of the cut off in native mode (before any transformation). See details.
`...`	additional arguments for the `cutoff` function.

Details

Notice that the list of cut off values returned by the function is not as complete as the one computed for a VLMC without covariates. Indeed, pruning the COVLMC tree creates new pruning opportunities that are not evaluated during the construction of the initial model, while all pruning opportunities are computed during the construction of a VLMC context tree. Nevertheless, the largest value returned by the function is guaranteed to produce the least pruned tree consistent with the reference one.

For large COVLMC, some cut off values can be almost identical, with a difference of the order of the machine epsilon value. The `tolerance` parameter is used to keep only values that are different enough. This is done in the quantile scale, before transformations implemented when `raw` is `FALSE`.

Notice that the loglikelihood scale is not directly useful in COVLMC as the differences in model sizes are not constant through the pruning process. As a consequence, this function does not provide `mode` parameter, contrarily to `cutoff.vlmc()`.

Setting `raw` to `TRUE` removes the small perturbation that are subtracted from the log-likelihood ratio values computed from the COVLMC (in quantile scale).

As automated model selection is provided by `tune_covlmc()`, the direct use of `cutoff` should be reserved to advanced exploration of the set of trees that can be obtained from a complex one, e.g. to implement model selection techniques that are not provided by `tune_covlmc()`.

Value

a vector of cut off values, `NULL` if none can be computed

Examples

```
pc <- powerconsumption[powerconsumption$week == 5, ]
rdts <- cut(pc$active_power, breaks = c(0, quantile(pc$active_power, probs = c(0.5, 1))))
m_nocovariate <- vlmc(rdts)
draw(m_nocovariate)
rdts_cov <- data.frame(day_night = (pc$hour >= 7 & pc$hour <= 17))
m_cov <- covlmc(rdts, rdts_cov, min_size = 5)
draw(m_cov)
cutoff(m_cov)
```

`cutoff.ctx_node` *Cut off value for pruning a node in the context tree of a VLMC*

Description

This function returns the cut off value associated to a specific node in the context tree interpreted as a VLMC. The node is represented by a `ctx_node` object as returned by `find_sequence()` or `contexts()`. For details, see `cutoff.vlmc()`.

Usage

```
## S3 method for class 'ctx_node'
cutoff(model, scale = c("quantile", "native"), raw = FALSE, ...)
```

Arguments

- `model` — a `ctx_node` object as returned by `find_sequence()`
- `scale` — specify whether the results should be "native" log likelihood ratio values or expressed in a "quantile" scale of a chi-squared distribution (defaults to "quantile").
- `raw` — specify whether the returned values should be limit values computed in the model or modified values that guarantee pruning (see details in `cutoff.vlmc()`)
- `...` — additional arguments for the `cutoff` function.

Value

a cut off value

See Also

`cutoff()`

Examples

```
pc <- powerconsumption[powerconsumption$week == 5, ]
rdts <- cut(pc$active_power,
  breaks = c(0, quantile(pc$active_power,
    probs = c(0.25, 0.5, 0.75, 1)
  ))
)
model <- vlmc(rdts)
model_ctxs <- contexts(model)
cutoff(model_ctxs[[1]])
cutoff(model_ctxs[[2]], scale = "native", raw = TRUE)
```

`cutoff.vlmc` *Cut off values for pruning the context tree of a VLMC*

Description

This function returns a collection of cut off values that are guaranteed to induce all valid pruned trees of the context tree of a VLMC. Pruning is implemented by the `prune()` function.

Usage

```
## S3 method for class 'vlmc'
cutoff(
  model,
  scale = c("quantile", "native"),
  raw = FALSE,
```

```
  tolerance = .Machine$double.eps^0.5,
  ...
)

## S3 method for class 'vlmc_cpp'
cutoff(
  model,
  scale = c("quantile", "native"),
  raw = FALSE,
  tolerance = .Machine$double.eps^0.5,
  ...
)
```

Arguments

`model`	a fitted VLMC model.
`scale`	specify whether the results should be "native" log likelihood ratio values or expressed in a "quantile" scale of a chi-squared distribution (defaults to "quantile").
`raw`	specify whether the returned values should be limit values computed in the model or modified values that guarantee pruning (see details)
`tolerance`	specify the minimum separation between two consecutive values of the cut off in native mode (before any transformation). See details.
`...`	additional arguments for the cutoff function.

Details

By default, the function returns values that can be used directly to induce pruning in the context tree. This is done by computing the log likelihood ratios used by the context algorithm on the reference VLMC and by keeping the relevant ones. From them the function selects intermediate values that are guaranteed to generate via pruning all the VLMC models that could be generated by using larger values of the `cutoff` parameter that was used to build the reference model (or smaller values of the `alpha` parameter in "quantile" scale).

Setting the `raw` parameter to `TRUE` removes this operation on the values and asks the function to return the relevant log likelihood ratios.

For large VLMC, some log likelihood ratios can be almost identical, with a difference of the order of the machine epsilon value. The `tolerance` parameter is used to keep only values that are different enough. This is done in the native scale, before transformations implemented when `raw` is `FALSE`.

As automated model selection is provided by `tune_vlmc()`, the direct use of `cutoff` should be reserved to advanced exploration of the set of trees that can be obtained from a complex one, e.g. to implement model selection techniques that are not provided by `tune_vlmc()`.

Value

a vector of cut off values.

See Also

prune() and tune_vlmc()

Examples

```
pc <- powerconsumption[powerconsumption$week == 5, ]
rdts <- cut(pc$active_power,
  breaks = c(0, quantile(pc$active_power,
    probs = c(0.25, 0.5, 0.75, 1)
  ))
)
model <- vlmc(rdts)
draw(model)
model_cuts <- cutoff(model)
model_2 <- prune(model, model_cuts[2])
draw(model_2)
```

---

depth   *Depth of a context tree*

Description

This function returns the depth of a context tree, i.e. the length of the longest context represented in the tree.

Usage

```
depth(ct)
```

Arguments

ct	a context tree.

Value

the depth of the tree.

Examples

```
rdts <- c(0, 1, 1, 1, 0, 0, 1, 0, 1, 0)
rdts_ctree <- ctx_tree(rdts, min_size = 1, max_depth = 3)
## should be 3
depth(rdts_ctree)
```

draw *Text based representation of a context tree*

Description

This function 'draws' a context tree as a text.

Usage

```
draw(ct, format, control = draw_control(), ...)
```

Arguments

`ct`	a context tree.
`format`	a character string that specifies the output format of the function. Possible values are `"text"` (default) and `"latex"`. See details.
`control`	a list of low level control parameters of the text representation. See details and `draw_control()`.
`...`	additional arguments for draw.

Details

The function uses different text based formats (plain "ascii art" and LaTeX) to represent the context tree. Fine tuning of the representation can be done via the `draw_control()` function.

In addition to the structure of the context tree, `draw()` can represent information attached to the nodes (contexts and partial contexts). This is controlled by additional parameters depending on the type of the context tree. In general, parameters given directly to `draw()` specify *what* information is represented while details on *how* this representation is made can be controlled via the `control` parameter and the associated `draw_control()` function.

Value

the context tree (invisibly).

Format

The `format` parameter specifies the format used for the textual output. With the default value `"text"` the output is produced in "ascii art" using by default only ascii characters (notice that `draw_control()` can be used to specified non ascii characters, but this is discouraged).

With the `latex` value, the output is produced in LaTeX, leveraging the forest Latex package (see `https://ctan.org/pkg/forest`). Each call to `draw()` produces a full `forest` LaTeX environment. This can be included as is in a LaTeX document, provided the `forest` package is loaded in the preamble of the document. The LaTeX output is sanitized to avoid potential problems induced by special characters in the names of the states of the context tree.

Examples

```
rdts <- sample(c(0, 1), 100, replace = TRUE)
ctree <- ctx_tree(rdts, min_size = 10, max_depth = 2)
draw(ctree)
rdts_c <- sample(c("A", "B", "CD"), 100, replace = TRUE)
ctree_c <- ctx_tree(rdts_c, min_size = 10, max_depth = 2)
draw(ctree_c, control = draw_control(digits = 2))
## LaTeX output
draw(ctree_c, "latex")
```

---

draw.covlmc — *Text based representation of a covlmc model*

Description

This function 'draws' a covlmc as a text.

Usage

```
## S3 method for class 'covlmc'
draw(
  ct,
  format,
  control = draw_control(),
  model = c("coef", "full"),
  p_value = FALSE,
  with_state = FALSE,
  constant_as_prob = TRUE,
  ...
)
```

Arguments

`ct`	a fitted covlmc model.
`format`	a character string that specifies the output format of the function. Possible values are `"text"` (default) and `"latex"`. See details.
`control`	a list of low level control parameters of the text representation. See details and `draw_control()`.
`model`	this parameter controls the display of logistic models associated to nodes (accepted values: `"coeff"`, `"full"` and `NULL`). The interpretation of the parameter depends on the format, see below for details.
`p_value`	specifies whether the p-values of the likelihood ratio tests conducted during the covlmc construction must be included in the representation (defaults to `FALSE`).
`with_state`	specifies whether to display the state associated to each dimension of the logistic model (see details).

`constant_as_prob`
: specifies how to represent constant logistic models for `format="text"` (defaults to `TRUE`, see details). Disregarded when `format="latex"`.

`...`
: additional arguments for draw.

Details

The function uses different text based formats (plain "ascii art" and LaTeX) to represent the context tree. Fine tuning of the representation can be done via the `draw_control()` function.

Contrarily to `draw()` functions adapted to context trees `draw.ctx_tree()` and VLMC `draw.vlmc()`, the present function does not try to produce similar results for the `"text"` format and the `"latex"` format as the `"text"` format is intrinsically more limited in terms of model representations. This is detailed below.

Format

The `format` parameter specifies the format used for the textual output. With the default value `"text"` the output is produced in "ascii art" using the charset specified by the global option `mixvlmc.charset`.

With the `latex` value, the output is produced in LaTeX, leveraging the forest Latex package (see `https://ctan.org/pkg/forest`). Each call to `draw.covlmc()` produces a full `forest` LaTeX environment. This can be included as is in a LaTeX document, provided the `forest` package is loaded in the preamble of the document. The LaTeX output is sanitized to avoid potential problems induced by special characters in the names of the states of the context tree.

`"text"` format

Parameters:

When `format="text"` the parameters are interpreted as follows:

- `model`: the default `model="coef"` represents only the *coefficients* of the logistic models associated to each context. `model="full"` includes the name of the variables in the representation. Setting `model=NULL` removes the model representations. Additional parameters can be used to tweak model representations (see below).
- `constant_as_prob`: specifies whether to represent logistic models that do not use covariates (a.k.a. constant models) using the probability distributions they induce on the state space (default behaviour with `constant_as_prob=TRUE`) or as normal models (when set to `FALSE`). This is not taken into account when `model` is not set to `"coef"`.
- fields of the `control` list (including the charset):
  - `intercept` : character(s) used to represent the intercept when `model="full"`
  - `intercept_sep`: character(s) used to separate the intercept from the other coefficients in model representation.
  - `time_sep`: character(s) used to split the coefficients list by blocks associated to time delays in the covariate inclusion into the logistic model. The first block contains the intercept(s), the second block the covariate values a time t-1, the third block at time t-2, etc.

  - level_sep: character(s) used separate levels from model, see below.
  - open_p_value and close_p_value: delimiters used around the p-values when p_value=TRUE
  - open_model and close_model: delimiters around the model when model is not NULL

State representation:

When model is not NULL, the coefficients of the logistic models are presented, organized in rows associated to states. One state is used as the reference state and the logistic model aims at predicting the ratio of probability between another state and the reference one (in log scale). When with_state is TRUE, the display includes for each row of coefficients the target state. This is useful when using e.g. VGAM::vglm() as unused levels of the target variable will be automatically dropped from the model, leading to a reduce number of rows. The reference state is either shown on the first row if model is "full" or after the state on each row if model is "coef". States are separated from the model representation by the character(s) specified in level_sep in the control list.

"latex" format

Parameters:

When format="latex" the parameters are interpreted as follows:

- model: the models are always represented completely in the LaTeX export unless model is set to NULL.
- constant_as_prob: in the LaTeX export, constant logistic models are always represented by the corresponding probability distribution on the state space, regardless of the value of constant_as_prob.
- fields of the control list:
  - orientation: specifies the orientation of the tree, either the default "vertical" (expanding from top to bottom) or "horizontal" (expanding from right to left);
  - tab_orientation: specifies the orientation of the tables used to represent model coefficients in the tree, either the default "vertical" (covariates are listed on one column) or "horizontal" (covariates are listed on one row);
  - fontsize and prob_fontsize handle the size of the fonts used for the states and for the models, see draw_control() for details;
  - decoration can be used to add borders around states, see draw_control() for details;

State representation:

When model is not NULL, the coefficients of the logistic models are presented, organized in rows or in columns (depending tab_orientation) on associated to states. One state is used as the reference state and the logistic model aims at predicting the ratio of probability between another state and the reference one (in log scale). When with_state is TRUE, the display includes for each row/column of coefficients the target state. The reference state is shown on the first row/column.

Variable representation

When the representation includes the names of the variables used by the logistic models, they are the one generated by the underlying logistic model, e.g. `stats::glm()`. Numerical variable names are used as is, while factors have levels appended. The intercept is denoted by the `intercept` member of the `control` list when`format="text"` (as part of the charset). It is always represented by `(I)` when `format="latex"`.

When `format="text"`, the time delays are represented by an underscore followed by the time delay. For instance if the model uses the numerical covariate `y` with two delays, it will appear with two variables `y_1` and `y_2`.

When `format="latex"`, the representation uses a temporal subscript of the form `t-1`, `t-2`, etc.

Examples

```
pc <- powerconsumption[powerconsumption$week == 5, ]
rdts <- cut(pc$active_power,
  breaks = c(0, quantile(pc$active_power, probs = c(0.5, 1)))
)
rdts_cov <- data.frame(day_night = (pc$hour >= 7 & pc$hour <= 17))
m_cov <- covlmc(rdts, rdts_cov, min_size = 5)
draw(m_cov, control = draw_control(digits = 3))
draw(m_cov, model = NULL)
draw(m_cov, p_value = TRUE)
draw(m_cov, p_value = FALSE, control = draw_control(digits = 2))
draw(m_cov, model = "full", control = draw_control(digits = 3))
draw(m_cov, format = "latex", control = draw_control(orientation = "h"))
```

`draw.ctx_tree_cpp` *Text based representation of a context tree*

Description

This function 'draws' a context tree as a text.

Usage

```
## S3 method for class 'ctx_tree_cpp'
draw(ct, format, control = draw_control(), frequency = NULL, ...)

## S3 method for class 'ctx_tree'
draw(ct, format, control = draw_control(), frequency = NULL, ...)
```

Arguments

`ct`	a context tree.
`format`	a character string that specifies the output format of the function. Possible values are `"text"` (default) and `"latex"`. See details.

`control` a list of low level control parameters of the text representation. See details and `draw_control()`.

`frequency` this parameter controls the display of node level information in the tree. The default `NULL` value does not include anything. Setting `frequency` to `"total"` includes the frequency of the (partial) context of the node, while `"detailed"` includes the frequency of the states that follow the context (as in `contexts.ctx_tree()`).

`...` additional arguments for draw.

Details

The function uses different text based formats (plain "ascii art" and LaTeX) to represent the context tree. Fine tuning of the representation can be done via the `draw_control()` function.

In addition to the structure of the context tree, `draw()` can represent information attached to the nodes (contexts and partial contexts). This is controlled by additional parameters depending on the type of the context tree. In general, parameters given directly to `draw()` specify *what* information is represented while details on *how* this representation is made can be controlled via the `control` parameter and the associated `draw_control()` function.

Value

the context tree (invisibly).

Format

The `format` parameter specifies the format used for the textual output. With the default value `"text"` the output is produced in "ascii art" using by default only ascii characters (notice that `draw_control()` can be used to specified non ascii characters, but this is discouraged).

With the `latex` value, the output is produced in LaTeX, leveraging the forest Latex package (see `https://ctan.org/pkg/forest`). Each call to `draw()` produces a full `forest` LaTeX environment. This can be included as is in a LaTeX document, provided the `forest` package is loaded in the preamble of the document. The LaTeX output is sanitized to avoid potential problems induced by special characters in the names of the states of the context tree.

Examples

```
rdts_c <- sample(c("A", "B", "CD"), 100, replace = TRUE)
ctree_c <- ctx_tree(rdts_c, min_size = 10, max_depth = 2)
draw(ctree_c, frequency = "total")
draw(ctree_c, frequency = "detailed")
## LaTeX output
draw(ctree_c, "latex", frequency = "detailed")
rdts_c <- sample(c("A$", "_{B", "{C}_{D}"), 100, replace = TRUE)
ctree_c <- ctx_tree(rdts_c, min_size = 10, max_depth = 2)
## the LaTeX output is sanitized
draw(ctree_c, "latex", frequency = "detailed")
```

`draw.vlmc` *Text based representation of a vlmc*

Description

This function 'draws' a context tree as a text.

Usage

```
## S3 method for class 'vlmc'
draw(ct, format, control = draw_control(), prob = TRUE, ...)

## S3 method for class 'vlmc_cpp'
draw(ct, format, control = draw_control(), prob = TRUE, ...)
```

Arguments

`ct`	a fitted vlmc.
`format`	a character string that specifies the output format of the function. Possible values are `"text"` (default) and `"latex"`. See details.
`control`	a list of low level control parameters of the text representation. See details and `draw_control()`.
`prob`	this parameter controls the display of node level information in the tree. The default `prob=TRUE` represents the conditional distribution of the states given the (partial) context associated to the node. Setting `prob=FALSE` replaces the conditional distribution by the frequency of the states that follow the context as in `draw.ctx_tree()`. Setting `prob=NULL` removes all additional information.
`...`	additional arguments for draw.

Details

The function uses different text based formats (plain "ascii art" and LaTeX) to represent the context tree. Fine tuning of the representation can be done via the `draw_control()` function.

In addition to the structure of the context tree, `draw()` can represent information attached to the nodes (contexts and partial contexts). This is controlled by additional parameters depending on the type of the context tree. In general, parameters given directly to `draw()` specify *what* information is represented while details on *how* this representation is made can be controlled via the `control` parameter and the associated `draw_control()` function.

Value

the context tree (invisibly).

Format

The `format` parameter specifies the format used for the textual output. With the default value `"text"` the output is produced in "ascii art" using by default only ascii characters (notice that `draw_control()` can be used to specified non ascii characters, but this is discouraged).

With the `latex` value, the output is produced in LaTeX, leveraging the forest Latex package (see `https://ctan.org/pkg/forest`). Each call to `draw()` produces a full `forest` LaTeX environment. This can be included as is in a LaTeX document, provided the `forest` package is loaded in the preamble of the document. The LaTeX output is sanitized to avoid potential problems induced by special characters in the names of the states of the context tree.

Examples

```
rdts <- sample(c("A", "B", "C"), 500, replace = TRUE)
model <- vlmc(rdts, alpha = 0.05)
draw(model)
draw(model, prob = FALSE)
draw(model, prob = NULL)
```

`draw_control` *Control parameters for* `draw`

Description

This function returns a list used to fine tune the `draw()` function behaviour.

Usage

```
draw_control(
  digits = 4,
  charset = NULL,
  orientation = c("vertical", "horizontal"),
  tabular = TRUE,
  tab_orientation = c("vertical", "horizontal"),
  decoration = c("none", "rectangle", "circle", "ellipse"),
  fontsize = "normalsize",
  prob_fontsize = "small"
)
```

Arguments

`digits`	numerical parameters and p-values are represented using the `base::signif()` function, using the number of significant digits specified with this parameter (defaults to 4).
`charset`	specifies the characters used for the "ascii art" represention when the format is "text", see details.

`orientation` specifies the global orientation of the tree, either ”vertical” (default) or ”horizontal” (”latex”).

`tabular` if TRUE (default value), the ”latex” format will use tables for each node, with one row for the state value and other rows for additional information (such as the conditional probability associated to the context). Notice that `draw.covlmc()` always uses tables regardless of the value of `tabular`.

`tab_orientation` specifies the way the models are represented when used by `draw.covlmc()` (”latex”). The default value is `"vertical"`: this is well adapted to models with long covariate dependencies (see `covariate_depth()`). The other possible value is `"horizontal"`.

`decoration` specifies node decoration in the ”latex” format, see details.

`fontsize` font size for the state names in the ”latex” format (using latex standard font size, default to `"normalsize"`).

`prob_fontsize` font size for the context counts, probabilities or models in the ”latex” format (using latex standard font size, defaults to `"small"`).

Details

Parameters are generally specific to the `format` used for `draw()`. If this is the case, the format is given at the end of the parameter description. Some parameters are also specific to some functions inheriting from `draw()`.

Value

a list

Decoration

The LaTeX format (`"latex"`) can ”decorate” the nodes of the context tree by drawing borders. We support only basic decorations, but in theory all TikZ possibilities could be used (see the documentation of the forest LaTeX package). Supported decorations:

- `"none"`: default, no decoration;
- `"rectangle"`: adds a rectangular border to all nodes;
- `"circle"`: adds a circular border to all nodes;
- `"ellipse"`: adds an ellipsoidal border to all nodes.

Charset

The ”ascii art” format (`"text"`) uses a collection of characters to display a context tree. The default collection is specified by the package option `"mixvlmc.charset"` and is used when `charset=NULL` (default value). If `charset` is set to a character value, this value is used to select the collection in the same way that `"mixvlmc.charset"` specifies it:

- `"ascii"`: the collection uses only standard ASCII characters and should be compatible with all environments;
- `"utf8"`: the collection uses UTF-8 symbols and needs a compatible display.

Finally, `charset` can a user supplied list of characters as the one returned by `charset_ascii()` and `charset_utf8()`.

See Also

`draw()`, `charset_ascii()` and `charset_utf8()`.

Examples

```
draw_control(digits = 2, tabular = FALSE)
```

---

dts — *Convert a vector to a discrete time series.*

Description

This function creates a representation of a discrete time series that can be further processed by model estimation functions.

Usage

```
dts(x, vals = NULL)
```

Arguments

`x`	a discrete time series; can be numeric, character, factor or logical.
`vals`	the set of values that can be taken by the time series, a.k.a. the state space, see details (defaults to `NULL`)

Details

The discrete time series `x` can be a vector of numeric, character, factor or logical type. If the state space of the series is not specified, that is when `vals` is `NULL`, it is computed in a way that depends on the type of `x`:

- for a factor, `vals` is set to the `levels()` of `x`;
- for a logical vector, `vals` is set to `c(FALSE, TRUE)`;
- for other types, `vals` is set to all the unique values taken by the time series (as returned by `sort(unique(x))`).

If `vals` is specified, the function makes sure that `x` contains only the specified values.

Value

a discrete time series (of class that inherits from `dts`).

Examples

```
x_dts <- dts(sample(c("A", "B"), 20, replace = TRUE))
x_dts
```

dts_data *Extract the plain discrete time series from a dts object*

Description

This function returns a copy of the discrete time series used to build the dts object (see dts()).

Usage

```
dts_data(x)
```

Arguments

x	a dts object

Value

a vector representing the time seris

Examples

```
raw_dts <- sample(c("A", "B", "C"), 50, replace = TRUE)
odts <- dts(raw_dts)
back_to_raw <- dts_data(odts)
## should be TRUE
identical(raw_dts, back_to_raw)
```

find_sequence *Find the node of a sequence in a context tree*

Description

This function checks whether the sequence `ctx` is represented in the context tree `ct`. If this is the case, it returns a description of matching node, an object of class `ctx_node`. If the sequence is not represented in the tree, the function return `NULL`.

Usage

```
find_sequence(ct, ctx, reverse = FALSE, ...)

## S3 method for class 'ctx_tree'
find_sequence(ct, ctx, reverse = FALSE, ...)

## S3 method for class 'ctx_tree_cpp'
find_sequence(ct, ctx, reverse = FALSE, ...)
```

Arguments

`ct`	a context tree.
`ctx`	a sequence to search in the context tree
`reverse`	specifies whether the sequence `ctx` is given the temporal order (`FALSE`, default value) or in the reverse temporal order (`TRUE`). See the dedicated section.
`...`	additional parameters for the find_sequence function

Details

The function looks for sequences in general. The `is_context()` function can be used on the resulting object to test if the sequence is in addition a proper context.

Value

an object of class `ctx_node` if the sequence `ctx` is represented in the context tree, `NULL` when this is not the case.

State order in a sequence

sequence are given by default in the temporal order and not in the "reverse" order used by many VLMC research papers: older values are on the left. For instance, the context `c(1, 0)` is reported if the sequence 0, then 1 appeared in the time series used to build the context tree. In the present function, `reverse` refers both to the order used for the `ctx` parameter and for the default order used by the resulting `ctx_node` object.

Examples

```
rdts <- c("A", "B", "C", "A", "A", "B", "B", "C", "C", "A")
rdts_tree <- ctx_tree(rdts, max_depth = 3)
find_sequence(rdts_tree, "A")
## returns NULL as "A" "C" does not appear in rdts
find_sequence(rdts_tree, c("A", "C"))
```

`find_sequence.covlmc` *Find the node of a sequence in a COVLMC context tree*

Description

This function checks whether the sequence `ctx` is represented in the context tree of the COVLMC model `ct`. If this is the case, it returns a description of matching node, an object of class `ctx_node_covlmc`. If the sequence is not represented in the tree, the function return `NULL`.

Usage

```
## S3 method for class 'covlmc'
find_sequence(ct, ctx, reverse = FALSE, ...)
```

Arguments

`ct`	a context tree.
`ctx`	a sequence to search in the context tree
`reverse`	specifies whether the sequence `ctx` is given the temporal order (`FALSE`, default value) or in the reverse temporal order (`TRUE`). See the dedicated section.
`...`	additional parameters for the find_sequence function

Details

The function looks for sequences in general. The `is_context()` function can be used on the resulting object to test if the sequence is in addition a proper context.

Value

an object of class `ctx_node_covlmc` if the sequence `ctx` is represented in the context tree, `NULL` when this is not the case

State order in a sequence

sequence are given by default in the temporal order and not in the "reverse" order used by many VLMC research papers: older values are on the left. For instance, the context `c(1, 0)` is reported if the sequence 0, then 1 appeared in the time series used to build the context tree. In the present function, `reverse` refers both to the order used for the `ctx` parameter and for the default order used by the resulting `ctx_node` object.

Examples

```
pc <- powerconsumption[powerconsumption$week == 5, ]
rdts <- cut(pc$active_power, breaks = c(0, quantile(pc$active_power, probs = c(0.5, 1))))
rdts_cov <- data.frame(day_night = (pc$hour >= 7 & pc$hour <= 17))
m_cov <- covlmc(rdts, rdts_cov, min_size = 10)

## not in the tree
vals <- states(m_cov)
find_sequence(m_cov, c(vals[2], vals[2]))
## in the tree but not a context
node <- find_sequence(m_cov, c(vals[1]))
node
is_context(node)
## in the tree and a context
node <- find_sequence(m_cov, c(vals[1], vals[1]))
node
is_context(node)
model(node)
```

`globalearthquake` *Significant Earthquake Dataset*

Description

A data set containing Earthquake that have occured during the period of 1900-2022 with GPS coordinates and magnitudes.

Usage

```
globalearthquake
```

Format

A data frame with 98785 rows and 12 variables:

date_time Date and time in POSIXct format

latitude latitude of the earthquake, from -90° to 90°

longitude longitude of the earthquake, from -180° to 180°

mag the magnitude of the earthquake, indicating its strenth

Date date when the seisme occured

nbweeks number of weeks since 1900/01/01

year year

month month of the year

month_day day of the month

week week number

week_day day of the week from 1 = Sunday to 7 = Saturday

year_day day of the year from 1 to 366

Details

This is a compiled version of the full data set available on U.S. Geological Survey Earthquake Events (USGS) which is in the public domain.

The data set contains only the earthquake between 1900 and 2022 with a magnitude higher than 5.

Source

Earthquake Catalog, U.S. Geological Survey, Department of the Interior. `https://www.usgs.gov/programs/earthquake-hazards`

is_context — *Report the nature of a node in a context tree*

Description

This function returns TRUE if the node is a proper context, FALSE in the other case.

Usage

```
is_context(node)
```

Arguments

node — a ctx_node object as returned by find_sequence()

Value

TRUE if the node node is a proper context, FALSE when this is not the case

Examples

```
rdts <- c(0, 1, 1, 1, 0, 0, 1, 0, 1, 0)
rdts_ctree <- ctx_tree(rdts, min_size = 1, max_depth = 3)
draw(rdts_ctree)
## 0, 0 is a context but 1, 0 is not
is_context(find_sequence(rdts_ctree, c(0, 0)))
is_context(find_sequence(rdts_ctree, c(1, 0)))
```

is_covlmc — *Test if the object is a covlmc model*

Description

This function returns TRUE for VLMC models with covariates and FALSE for other objects.

Usage

```
is_covlmc(x)
```

Arguments

x — an R object.

Value

TRUE for VLMC models with covariates.

Examples

```
pc <- powerconsumption[powerconsumption$week == 5, ]
rdts <- cut(pc$active_power, breaks = c(0, quantile(pc$active_power, probs = c(0.5, 1))))
rdts_cov <- data.frame(day_night = (pc$hour >= 7 & pc$hour <= 17))
m_cov <- covlmc(rdts, rdts_cov, min_size = 5)
# should be true
is_ctx_tree(m_cov)
# should be true
is_covlmc(m_cov)
# should be false
is_vlmc(m_cov)
```

`is_ctx_tree` *Test if the object is a context tree*

Description

This function returns `TRUE` for context trees and `FALSE` for other objects.

Usage

```
is_ctx_tree(x)
```

Arguments

`x`	an R object.

Value

`TRUE` for context trees.

Examples

```
rdts <- c(0, 1, 1, 1, 0, 0, 1, 0, 1, 0)
rdts_ctree <- ctx_tree(rdts, min_size = 1, max_depth = 2)
is_ctx_tree(rdts_ctree)
is_ctx_tree(rdts)
```

is_dts *Test if the object is a discrete time series*

Description

This function returns TRUE for discrete time series and FALSE for other objects.

Usage

```
is_dts(x)
```

Arguments

x an R object.

Value

TRUE for discrete time series.

Examples

```
pre_dts <- sample(c("A", "B"), 20, replace = TRUE)
x_dts <- dts(pre_dts)
is_dts(x_dts)
is_dts(pre_dts)
```

is_merged *Merging status of a COVLMC context*

Description

The function returns TRUE if the context represented by this node is merged with at least another one and FALSE if this is not the case.

Usage

```
is_merged(node)
```

Arguments

node A ctx_node_covlmc object as returned by find_sequence() or contexts.covlmc()

Details

When a COVLMC is built on a time series with at least three distinct states, some contexts can be merged: they use the same logistic model, leading to a more parsimonious model. Those contexts are reported individually by functions such as contexts.covlmc(). The present function can be used to detect such merging, while merged_with() can be used to recover the other contexts.

Value

TRUE or FALSE, depending on the nature of the context

See Also

merged_with()

Examples

```
pc <- powerconsumption[powerconsumption$week == 15, ]
rdts <- cut(pc$active_power, breaks = c(0, 1, 2, 3, 8))
rdts_cov <- data.frame(day_night = (pc$hour >= 7 & pc$hour <= 17))
m_cov <- covlmc(rdts, rdts_cov, min_size = 5, alpha = 0.1)
ctxs <- contexts(m_cov)
## no merging
sapply(ctxs, is_merged)
```

---

is_reversed — *Report the ordering convention of the node*

Description

This function returns `TRUE` if the node is using a reverse temporal ordering and `FALSE` in the other case.

Usage

```
is_reversed(node)
```

Arguments

node	a `ctx_node` object as returned by find_sequence()

Value

`TRUE` if the node `node` use a reverse temporal ordering, `FALSE` when this is not the case

See Also

rev.ctx_node()

Examples

```
rdts <- c(0, 1, 1, 1, 0, 0, 1, 0, 1, 0)
rdts_ctree <- ctx_tree(rdts, min_size = 1, max_depth = 3)
is_reversed(find_sequence(rdts_ctree, c(0, 0)))
is_reversed(find_sequence(rdts_ctree, c(1, 0), reverse = TRUE))
```

is_vlmc *Test if the object is a vlmc model*

Description

This function returns TRUE for VLMC models and FALSE for other objects.

Usage

```
is_vlmc(x)
```

Arguments

x	an R object.

Value

TRUE for VLMC models.

Examples

```
pc <- powerconsumption[powerconsumption$week == 5, ]
rdts <- cut(pc$active_power,
  breaks = c(0, quantile(pc$active_power,
    probs = c(0.25, 0.5, 0.75, 1)
  ))
)
model <- vlmc(rdts)
# should be true
is_ctx_tree(model)
# should be true
is_vlmc(model)
# should be false
is_covlmc(model)
```

logLik.covlmc *Log-Likelihood of a VLMC with covariates*

Description

This function evaluates the log-likelihood of a VLMC with covariates fitted on a discrete time series.

Usage

```
## S3 method for class 'covlmc'
logLik(object, initial = c("truncated", "specific", "extended"), ...)
```

Arguments

`object`	the covlmc representation.
`initial`	specifies the likelihood function, more precisely the way the first few observations for which contexts cannot be calculated are integrated in the likelihood. Defaults to `"truncated"`. See `loglikelihood()` for details.
`...`	additional parameters for logLik.

Value

an object of class `logLik`. This is a number, the log-likelihood of the (CO)VLMC with the following attributes:

- `df`: the number of parameters used by the VLMC for this likelihood calculation
- `nobs`: the number of observations included in this likelihood calculation
- `initial`: the value of the `initial` parameter used to compute this likelihood

See Also

`loglikelihood()`

Examples

```
## Likelihood for a fitted VLMC with covariates.
pc <- powerconsumption[powerconsumption$week == 5, ]
breaks <- c(
  0,
  median(powerconsumption$active_power, na.rm = TRUE),
  max(powerconsumption$active_power, na.rm = TRUE)
)
labels <- c(0, 1)
rdts <- cut(pc$active_power, breaks = breaks, labels = labels)
rdts_cov <- data.frame(day_night = (pc$hour >= 7 & pc$hour <= 17))
m_cov <- covlmc(rdts, rdts_cov, min_size = 5)
ll <- logLik(m_cov)
attributes(ll)
```

`logLik.vlmc` *Log-Likelihood of a VLMC*

Description

This function evaluates the log-likelihood of a VLMC fitted on a discrete time series.

Usage

```
## S3 method for class 'vlmc'
logLik(object, initial = c("truncated", "specific", "extended"), ...)

## S3 method for class 'vlmc_cpp'
logLik(object, initial = c("truncated", "specific", "extended"), ...)
```

Arguments

`object`	the vlmc representation.
`initial`	specifies the likelihood function, more precisely the way the first few observations for which contexts cannot be calculated are integrated in the likelihood. Defaults to `"truncated"`. See `loglikelihood()` for details.
`...`	additional parameters for logLik.

Value

an object of class `logLik`. This is a number, the log-likelihood of the (CO)VLMC with the following attributes:

- `df`: the number of parameters used by the VLMC for this likelihood calculation
- `nobs`: the number of observations included in this likelihood calculation
- `initial`: the value of the `initial` parameter used to compute this likelihood

See Also

`loglikelihood()`

Examples

```
pc <- powerconsumption[powerconsumption$week == 5, ]
breaks <- c(
  0,
  median(powerconsumption$active_power, na.rm = TRUE),
  max(powerconsumption$active_power, na.rm = TRUE)
)
labels <- c(0, 1)
rdts <- cut(pc$active_power, breaks = breaks, labels = labels)
m_nocovariate <- vlmc(rdts)
ll <- logLik(m_nocovariate)
ll
attributes(ll)
```

`loglikelihood` *Log-Likelihood of a VLMC*

Description

This function evaluates the log-likelihood of a VLMC fitted on a discrete time series. When the optional argument `newdata` is provided, the function evaluates instead the log-likelihood for this (new) discrete time series.

Usage

```
loglikelihood(
  vlmc,
  newdata,
  initial = c("truncated", "specific", "extended"),
  ignore,
  ...
)

## S3 method for class 'vlmc'
loglikelihood(
  vlmc,
  newdata,
  initial = c("truncated", "specific", "extended"),
  ignore,
  ...
)

## S3 method for class 'vlmc_cpp'
loglikelihood(
  vlmc,
  newdata,
  initial = c("truncated", "specific", "extended"),
  ignore,
  ...
)
```

Arguments

`vlmc`	the vlmc representation.
`newdata`	an optional object that can be interpreted as a discrete time series (for instance a `dts` object).
`initial`	specifies the likelihood function, more precisely the way the first few observations for which contexts cannot be calculated are integrated in the likelihood. Defaults to `"truncated"`. See below for details.

`ignore` specifies the number of initial values for which the loglikelihood will not be computed. The minimal number depends on the likelihood function as detailed below.

`...` additional parameters for loglikelihood.

Details

The definition of the likelihood function depends on the value of the `initial` parameters, see the section below as well as the dedicated vignette: `vignette("likelihood", package = "mixvlmc")`.

For VLMC objects, the method `loglikelihood.vlmc` will be used. For VLMC with covariables, `loglikelihood.covlmc` will instead be called. For more informations on `loglikelihood` methods, use `methods(loglikelihood)` and their associated documentation.

Value

an object of class `logLikMixVLMC` and `logLik`. This is a number, the log-likelihood of the (CO)VLMC with the following attributes:

- `df`: the number of parameters used by the VLMC for this likelihood calculation
- `nobs`: the number of observations included in this likelihood calculation
- `initial`: the value of the `initial` parameter used to compute this likelihood

likelihood calculation

In a (CO)VLMC of `depth()`=k, we need k past values in order to compute the context of a given observation. As a consequence, in a time series `x`, the contexts of `x[1]` to `x[k]` are unknown. Depending on the value of `initial` different likelihood functions are used to tackle this difficulty:

- `initial=="truncated"`: the likelihood is computed using only `x[(k+1):length(x)]`
- `initial=="specific"`: the likelihood is computed on the full time series using a specific context for the initial values, `x[1]` to `x[k]`. Each of the specific context is unique, leading to a perfect likelihood of 1 (0 in log scale). Thus the numerical value of the likelihood is identical as the one obtained with `initial=="truncated"` but it is computed on `length(x)` with a model with more parameters than in this previous case.
- `initial=="extended"` (default): the likelihood is computed on the full time series using an extended context matching for the initial values, `x[1]` to `x[k]`. This can be seen as a compromised between the two other possibilities: the relaxed context matching needs in general to turn internal nodes of the context tree into actual context, increasing the number of parameters, but not as much as with ”specific”. However, the likelihood of say `x[1]` with an empty context is generally not 1 and thus the full likelihood is smaller than the one computed with ”specific”.

In all cases, the `ignore` first values of the time series are not included in the computed likelihood, but still used to compute contexts. If `ignore` is not specified, it is set to the minimal possible value, that is k for the `truncated` likelihood and 0 for the other ones. If it is specified, it must be larger or equal to k for `truncated`.

See the dedicated vignette for a more mathematically oriented discussion: `vignette("likelihood", package = "mixvlmc")`.

See Also

`stats::logLik()`

Examples

```
## Likelihood for a fitted VLMC.
pc <- powerconsumption[powerconsumption$week == 5, ]
breaks <- c(
  0,
  median(powerconsumption$active_power, na.rm = TRUE),
  max(powerconsumption$active_power, na.rm = TRUE)
)
labels <- c(0, 1)
rdts <- cut(pc$active_power, breaks = breaks, labels = labels)
m_nocovariate <- vlmc(rdts)
ll <- loglikelihood(m_nocovariate)
ll
attr(ll, "nobs")
attr(ll, "df")

## Likelihood for a new time series with previously fitted VLMC.
pc_new <- powerconsumption[powerconsumption$week == 11, ]
rdts_new <- cut(pc_new$active_power, breaks = breaks, labels = labels)
ll_new <- loglikelihood(m_nocovariate, newdata = rdts_new)
ll_new
attributes(ll_new)
ll_new_specific <- loglikelihood(m_nocovariate, initial = "specific", newdata = rdts_new)
ll_new_specific
attributes(ll_new_specific)
ll_new_extended <- loglikelihood(m_nocovariate, initial = "extended", newdata = rdts_new)
ll_new_extended
attributes(ll_new_extended)
```

`loglikelihood.covlmc` *Log-Likelihood of a VLMC with covariates*

Description

This function evaluates the log-likelihood of a VLMC with covariates fitted on a discrete time series. When the optional arguments `newdata` is provided, the function evaluates instead the log-likelihood for this (new) discrete time series on the new covariates which must be provided through the `newcov` parameter.

Usage

```
## S3 method for class 'covlmc'
loglikelihood(
  vlmc,
  newdata,
  initial = c("truncated", "specific", "extended"),
  ignore,
  newcov,
  ...
)
```

Arguments

`vlmc`	the covlmc representation.
`newdata`	an optional object that can be interpreted as a discrete time series (for instance a `dts` object).
`initial`	specifies the likelihood function, more precisely the way the first few observations for which contexts cannot be calculated are integrated in the likelihood. Defaults to `"truncated"`. See below for details.
`ignore`	specifies the number of initial values for which the loglikelihood will not be computed. The minimal number depends on the likelihood function as detailed below.
`newcov`	an optional data frame with the new values for the covariates.
`...`	additional parameters for loglikelihood.

Details

The definition of the likelihood function depends on the value of the `initial` parameters, see the section below as well as the dedicated vignette: `vignette("likelihood", package = "mixvlmc")`.

Value

an object of class `logLikMixVLMC` and `logLik`. This is a number, the log-likelihood of the (CO)VLMC with the following attributes:

- `df`: the number of parameters used by the VLMC for this likelihood calculation
- `nobs`: the number of observations included in this likelihood calculation
- `initial`: the value of the `initial` parameter used to compute this likelihood

likelihood calculation

In a (CO)VLMC of `depth()`=k, we need k past values in order to compute the context of a given observation. As a consequence, in a time series `x`, the contexts of `x[1]` to `x[k]` are unknown. Depending on the value of `initial` different likelihood functions are used to tackle this difficulty:

- `initial=="truncated"`: the likelihood is computed using only `x[(k+1):length(x)]`

- `initial=="specific"`: the likelihood is computed on the full time series using a specific context for the initial values, `x[1]` to `x[k]`. Each of the specific context is unique, leading to a perfect likelihood of 1 (0 in log scale). Thus the numerical value of the likelihood is identical as the one obtained with `initial=="truncated"` but it is computed on `length(x)` with a model with more parameters than in this previous case.
- `initial=="extended"` (default): the likelihood is computed on the full time series using an extended context matching for the initial values, `x[1]` to `x[k]`. This can be seen as a compromised between the two other possibilities: the relaxed context matching needs in general to turn internal nodes of the context tree into actual context, increasing the number of parameters, but not as much as with "specific". However, the likelihood of say `x[1]` with an empty context is generally not 1 and thus the full likelihood is smaller than the one computed with "specific".

In all cases, the `ignore` first values of the time series are not included in the computed likelihood, but still used to compute contexts. If `ignore` is not specified, it is set to the minimal possible value, that is k for the `truncated` likelihood and 0 for the other ones. If it is specified, it must be larger or equal to k for `truncated`.

See the dedicated vignette for a more mathematically oriented discussion: `vignette("likelihood", package = "mixvlmc")`.

See Also

`stats::logLik()`

Examples

```
## Likelihood for a fitted VLMC with covariates.
pc <- powerconsumption[powerconsumption$week == 5, ]
breaks <- c(
  0,
  median(powerconsumption$active_power, na.rm = TRUE),
  max(powerconsumption$active_power, na.rm = TRUE)
)
labels <- c(0, 1)
rdts <- cut(pc$active_power, breaks = breaks, labels = labels)
rdts_cov <- data.frame(day_night = (pc$hour >= 7 & pc$hour <= 17))
m_cov <- covlmc(rdts, rdts_cov, min_size = 5)
ll <- loglikelihood(m_cov)
ll
attr(ll, "nobs")

## Likelihood for new time series and covariates with previously
## fitted VLMC with covariates
pc_new <- powerconsumption[powerconsumption$week == 11, ]
rdts_new <- cut(pc_new$active_power, breaks = breaks, labels = labels)
rdts_cov_new <- data.frame(day_night = (pc_new$hour >= 7 & pc_new$hour <= 17))
ll_new <- loglikelihood(m_cov, newdata = rdts_new, newcov = rdts_cov_new)
ll_new
attributes(ll_new)
```

`merged_with` *Merged contexts in a COVLMC*

Description

The function returns `NULL` when the context represented by the `node` parameter is not merged with another context (see `is_merged()`). In the other case, it returns a list of contexts with which this one is merged.

Usage

```
merged_with(node)
```

Arguments

`node`	A `ctx_node_covlmc` object as returned by `find_sequence()` or `contexts.covlmc()`

Details

If the context is merged, the function returns a list with one value for each element in the state space (see `states()`). The value is `NULL` if the corresponding context is not merged with the `node` context, while it is a `ctx_node_covlmc` object in the other case. A context merged with `node` differs from the context represented by `node` only in its last value (in temporal order) which is used as its name in the list. For instance, if the context `ABC` is merged only with `CBC` (when represented in temporal ordering), then the resulting list is of the form `list("A" = NULL, "B" = NULL, "C"= ctx_node_covlmc(CBX))`.

Value

NULL or a list of contexts merged with `node` represented by `ctx_node_covlmc` objects

See Also

`is_merged()`

Examples

```
pc_week_15_16 <- powerconsumption[powerconsumption$week %in% c(15, 16), ]
elec <- pc_week_15_16$active_power
elec_rdts <- cut(elec, breaks = c(0, 0.4, 2, 8), labels = c("low", "typical", "high"))
elec_cov <- data.frame(day = (pc_week_15_16$hour >= 7 & pc_week_15_16$hour <= 18))
elec_tune <- tune_covlmc(elec_rdts, elec_cov, min_size = 5)
elec_model <- prune(as_covlmc(elec_tune), alpha = 3.961e-10)
ctxs <- contexts(elec_model)
for (ctx in ctxs) {
  if (is_merged(ctx)) {
    print(ctx)
```

```
    cat("\nis merged with\n\n")
    print(merged_with(ctx))
  }
}
```

`metrics` *Predictive quality metrics for context based models*

Description

This function computes and returns predictive quality metrics for context based models such as VLMC and VLMC with covariates.

Usage

```
metrics(model, ...)
```

Arguments

`model`	The context based model on which to compute predictive metrics.
`...`	Additional parameters for predictive metrics computation.

Details

A context based model computes transition probabilities for its contexts. Using a maximum transition probability decision rule, this can be used to predict the new state that is the more likely to follow the current one, given the context (see `predict.vlmc()`). The quality of these predictions is evaluated using standard metrics including:

- accuracy
- the full confusion matrix
- the area under the roc curve (AUC), considering the context based model as a (conditional) probability estimator. We use Hand and Till (2001) multiclass AUC in case of a state space with more than 2 states

Value

The returned value is guaranteed to have at least three components

- `accuracy`: the accuracy of the predictions
- `conf_mat`: the confusion matrix of the predictions, with predicted values in rows and true values in columns
- `auc`: the AUC of the predictive model

References

David J. Hand and Robert J. Till (2001). "A Simple Generalisation of the Area Under the ROC Curve for Multiple Class Classification Problems." *Machine Learning* 45(2), p. 171–186. DOI: doi:10.1023/A:1010920819831.

See Also

metrics.vlmc(), metrics.ctx_node(), contexts.vlmc(), predict.vlmc().

Examples

```
pc <- powerconsumption[powerconsumption$week == 5, ]
breaks <- c(
  0,
  median(powerconsumption$active_power, na.rm = TRUE),
  max(powerconsumption$active_power, na.rm = TRUE)
)
labels <- c(0, 1)
rdts <- cut(pc$active_power, breaks = breaks, labels = labels)
model <- vlmc(rdts)
metrics(model)
```

---

metrics.covlmc *Predictive quality metrics for VLMC with covariates*

Description

This function computes and returns predictive quality metrics for context based models such as VLMC and VLMC with covariates.

Usage

```
## S3 method for class 'covlmc'
metrics(model, ...)

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

Arguments

`model`	The context based model on which to compute predictive metrics.
`...`	Additional parameters for predictive metrics computation.
`x`	A metrics.covlmc object, results of a call to metrics.covlmc()

Details

A context based model computes transition probabilities for its contexts. Using a maximum transition probability decision rule, this can be used to predict the new state that is the more likely to follow the current one, given the context (see predict.vlmc()). The quality of these predictions is evaluated using standard metrics including:

- accuracy
- the full confusion matrix

- the area under the roc curve (AUC), considering the context based model as a (conditional) probability estimator. We use Hand and Till (2001) multiclass AUC in case of a state space with more than 2 states

Value

An object of class `metrics.covlmc` with the following components:

- `accuracy`: the accuracy of the predictions
- `conf_mat`: the confusion matrix of the predictions, with predicted values in rows and true values in columns
- `auc`: the AUC of the predictive model

The object has a print method that recalls basic information about the model together with the values of the components above.

Methods (by generic)

- `print(metrics.covlmc)`: Prints the predictive metrics of the VLMC model with covariates.

Extended contexts

As explained in details in `loglikelihood.covlmc()` documentation and in the dedicated `vignette("likelihood", package = "mixvlmc")`, the first initial values of a time series do not in general have a proper context for a COVLMC with a non zero order. In order to predict something meaningful for those values, we rely on the notion of extended context defined in the documents mentioned above. This follows the same logic as using `loglikelihood.covlmc()` with the parameter `initial="extended"`. All covlmc functions that need to manipulate initial values with no proper context use the same approach.

References

David J. Hand and Robert J. Till (2001). "A Simple Generalisation of the Area Under the ROC Curve for Multiple Class Classification Problems." *Machine Learning* 45(2), p. 171–186. DOI: doi:10.1023/A:1010920819831.

See Also

`metrics.vlmc()`, `metrics.ctx_node()`, `contexts.vlmc()`, `predict.vlmc()`.

Examples

```
pc <- powerconsumption[powerconsumption$week == 5, ]
breaks <- c(
  0,
  median(powerconsumption$active_power, na.rm = TRUE),
  max(powerconsumption$active_power, na.rm = TRUE)
)
labels <- c(0, 1)
rdts <- cut(pc$active_power, breaks = breaks, labels = labels)
```

```
rdts_cov <- data.frame(day_night = (pc$hour >= 7 & pc$hour <= 17))
m_cov <- covlmc(rdts, rdts_cov, min_size = 5)
metrics(m_cov)
```

`metrics.ctx_node` *Predictive quality metrics for a node of a context tree*

Description

This function computes and returns predictive quality metrics for a node (`ctx_node`) extracted from a context tree.

Usage

```
## S3 method for class 'ctx_node'
metrics(model, ...)
```

Arguments

`model`	T `ctx_node` object as returned by `find_sequence()`.
`...`	Additional parameters for predictive metrics computation.

Details

Compared to `metrics.vlmc()`, this function focuses on a single context and assesses the quality of its predictions, disregarding observations that have other contexts. Apart from this limited scope, the function operates as `metrics.vlmc()`.

Value

The returned value is guaranteed to have at least three components

- `accuracy`: the accuracy of the predictions
- `conf_mat`: the confusion matrix of the predictions, with predicted values in rows and true values in columns
- `auc`: the AUC of the predictive model

References

David J. Hand and Robert J. Till (2001). "A Simple Generalisation of the Area Under the ROC Curve for Multiple Class Classification Problems." *Machine Learning* 45(2), p. 171–186. DOI: doi:10.1023/A:1010920819831.

See Also

`metrics.vlmc()`, `metrics.ctx_node()`, `contexts.vlmc()`, `predict.vlmc()`.

Examples

```
pc <- powerconsumption[powerconsumption$week == 5, ]
rdts <- cut(pc$active_power,
  breaks = c(0, quantile(pc$active_power,
    probs = c(0.25, 0.5, 0.75, 1)
  ))
)
model <- vlmc(rdts)
model_ctxs <- contexts(model)
metrics(model_ctxs[[4]])
```

---

metrics.ctx_node_covlmc

Predictive quality metrics for a node of a COVLMC context tree

---

Description

This function computes and returns predictive quality metrics for a node (`ctx_node_covlmc`) extracted from a covlmc

Usage

```
## S3 method for class 'ctx_node_covlmc'
metrics(model, ...)
```

Arguments

`model`	A `ctx_node_covlmc` object as returned by `find_sequence()` or `contexts.covlmc()`
`...`	Additional parameters for predictive metrics computation.

Details

Compared to `metrics.covlmc()`, this function focuses on a single context and assesses the quality of its predictions, disregarding observations that have other contexts. Apart from this limited scope, the function operates as `metrics.covlmc()`.

Value

an object of class `metrics.covlmc` with the following components:

- `accuracy`: the accuracy of the predictions
- `conf_mat`: the confusion matrix of the predictions, with predicted values in rows and true values in columns
- `auc`: the AUC of the predictive model

References

David J. Hand and Robert J. Till (2001). "A Simple Generalisation of the Area Under the ROC Curve for Multiple Class Classification Problems." *Machine Learning* 45(2), p. 171–186. DOI: doi:10.1023/A:1010920819831.

See Also

`metrics.vlmc()`, `metrics.ctx_node()`, `contexts.vlmc()`, `predict.vlmc()`.

Examples

```
pc <- powerconsumption[powerconsumption$week == 5, ]
breaks <- c(
  0,
  median(powerconsumption$active_power, na.rm = TRUE),
  max(powerconsumption$active_power, na.rm = TRUE)
)
labels <- c(0, 1)
rdts <- cut(pc$active_power, breaks = breaks, labels = labels)
rdts_cov <- data.frame(day_night = (pc$hour >= 7 & pc$hour <= 17))
m_cov <- covlmc(rdts, rdts_cov, min_size = 5)
m_ctxs <- contexts(m_cov)
## get the predictive metrics for each context
lapply(m_ctxs, metrics)
```

`metrics.vlmc` *Predictive quality metrics for VLMC*

Description

This function computes and returns predictive quality metrics for context based models such as VLMC and VLMC with covariates.

Usage

```
## S3 method for class 'vlmc'
metrics(model, ...)

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

Arguments

`model`	The context based model on which to compute predictive metrics.
`...`	Additional parameters for predictive metrics computation.
`x`	A metrics.vlmc object, results of a call to `metrics.vlmc()`

Details

A context based model computes transition probabilities for its contexts. Using a maximum transition probability decision rule, this can be used to predict the new state that is the more likely to follow the current one, given the context (see `predict.vlmc()`). The quality of these predictions is evaluated using standard metrics including:

- accuracy
- the full confusion matrix
- the area under the roc curve (AUC), considering the context based model as a (conditional) probability estimator. We use Hand and Till (2001) multiclass AUC in case of a state space with more than 2 states

Value

An object of class `metrics.vlmc` with the following components:

- `accuracy`: the accuracy of the predictions
- `conf_mat`: the confusion matrix of the predictions, with predicted values in rows and true values in columns
- `auc`: the AUC of the predictive model

The object has a print method that recalls basic information about the model together with the values of the components above.

Methods (by generic)

- `print(metrics.vlmc)`: Prints the predictive metrics of the VLMC model.

Extended contexts

As explained in details in `loglikelihood.vlmc()` documentation and in the dedicated `vignette("likelihood", package = "mixvlmc")`, the first initial values of a time series do not in general have a proper context for a VLMC with a non zero order. In order to predict something meaningful for those values, we rely on the notion of extended context defined in the documents mentioned above. This follows the same logic as using `loglikelihood.vlmc()` with the parameter `initial="extended"`. All vlmc functions that need to manipulate initial values with no proper context use the same approach.

References

David J. Hand and Robert J. Till (2001). "A Simple Generalisation of the Area Under the ROC Curve for Multiple Class Classification Problems." *Machine Learning* 45(2), p. 171–186. DOI: doi:10.1023/A:1010920819831.

See Also

`metrics.vlmc()`, `metrics.ctx_node()`, `contexts.vlmc()`, `predict.vlmc()`.

Examples

```
pc <- powerconsumption[powerconsumption$week == 5, ]
breaks <- c(
  0,
  median(powerconsumption$active_power, na.rm = TRUE),
  max(powerconsumption$active_power, na.rm = TRUE)
)
labels <- c(0, 1)
rdts <- cut(pc$active_power, breaks = breaks, labels = labels)
model <- vlmc(rdts)
metrics(model)
```

---

model — *Logistic model of a COVLMC context*

Description

This function returns a representation of the logistic model associated to a COVLMC context from its node in the associated context tree.

Usage

```
model(node, type = c("coef", "full"))
```

Arguments

`node`	A `ctx_node_covlmc` object as returned by `find_sequence()` or `contexts.covlmc()`
`type`	specifies the model information to return, either the coefficients only (`type="coef"` default case) or the full model object (`type="full"`)

Details

Full model extraction is only possible if the COVLMC model what not fully trimmed (see `trim.covlmc()`). Notice that `find_sequence.covlmc()` can produce node that are not context: in this case this function return `NULL`.

Value

if `node` is a context, the coefficients of the logistic model (as a vector or a matrix depending on the size of the state space) or a logistic model as a R object. If `node` is not a context, `NULL`.

Examples

```
pc <- powerconsumption[powerconsumption$week == 5, ]
rdts <- cut(pc$active_power, breaks = c(0, quantile(pc$active_power, probs = c(0.5, 1))))
rdts_cov <- data.frame(day_night = (pc$hour >= 7 & pc$hour <= 17))
m_cov <- covlmc(rdts, rdts_cov, min_size = 10)
vals <- states(m_cov)
node <- find_sequence(m_cov, c(vals[1], vals[1]))
node
model(node)
model(node, type = "full")
```

---

parent — *Find the parent of a node in a context tree*

Description

This function returns the parent node of the node represented by the `node` parameter. The result is `NULL` if `node` is the root node of its context tree (representing the empty sequence).

Usage

```
parent(node)

## S3 method for class 'ctx_node'
parent(node)

## S3 method for class 'ctx_node_cpp'
parent(node)
```

Arguments

`node`	a `ctx_node` object as returned by `find_sequence()`

Details

Each node of a context tree represents a sequence. When `find_sequence()` is called with success, the returned object represents the corresponding node in the context tree. Unless the original sequence is empty, this node has a parent node which is returned as a `ctx_node` object by the present function. Another interpretation is that the function returns the `node` object associated to the sequence obtained by removing the oldest value from the original sequence.

Value

a `ctx_node` object if `node` does correspond to the empty sequence or `NULL` when this is not the case

Examples

```
rdts <- c(0, 1, 1, 1, 0, 0, 1, 0, 1, 0)
rdts_ctree <- ctx_tree(rdts, min_size = 1, max_depth = 3)
ctx_00 <- find_sequence(rdts_ctree, c(0, 0))
## the parent sequence/node corresponds to the 0 context
parent(ctx_00)
identical(parent(ctx_00), find_sequence(rdts_ctree, c(0)))
## C++ backend
rdts <- c(0, 1, 1, 1, 0, 0, 1, 0, 1, 0)
rdts_ctree <- ctx_tree(rdts, min_size = 1, max_depth = 3, backend = "C++")
ctx_00 <- find_sequence(rdts_ctree, c(0, 0))
## the parent sequence/node corresponds to the 0 context
parent(ctx_00)
identical(parent(ctx_00), find_sequence(rdts_ctree, c(0)))
```

---

plot.tune_vlmc — *Plot the results of automatic (CO)VLMC complexity selection*

Description

This function plots the results of `tune_vlmc()` or `tune_covlmc()`.

Usage

```
## S3 method for class 'tune_vlmc'
plot(
  x,
  value = c("criterion", "likelihood"),
  cutoff = c("quantile", "native"),
  ...
)

## S3 method for class 'tune_covlmc'
plot(
  x,
  value = c("criterion", "likelihood"),
  cutoff = c("quantile", "native"),
  ...
)
```

Arguments

`x`	a `tune_vlmc` object
`value`	the criterion to plot (default ”criterion”).
`cutoff`	the scale used for the cut off criterion (default ”quantile”)
`...`	additional parameters passed to `base::plot()`

Details

The standard plot consists in showing the evolution of the criterion used to select the model (AIC() or BIC()) as a function of the cut off criterion expressed in the quantile scale (the quantile is used by default to offer a common default behaviour between vlmc() and covlmc()). Parameters can be used to display instead the loglikelihood() of the model (by setting value="likelihood") and to use the native scale for the cut off when available (by setting cutoff="native").

Value

the tune_vlmc object invisibly

Customisation

The function sets several default before calling base::plot(), namely:

- type: "l" by default to use a line representation;
- xlab: "Cut off (quantile scale)" by default, adapted to the actual scale;
- ylab: the name of the criterion or "Log likelihood".

These parameters can be overridden by specifying other values when calling the function. All parameters specified in addition to x, value and cutoff are passed to base::plot().

Examples

```
rdts <- sample(as.factor(c("A", "B", "C")), 100, replace = TRUE)
tune_result <- tune_vlmc(rdts)
## default plot
plot(tune_result)
## likelihood
plot(tune_result, value = "likelihood")
## parameters overriding
plot(tune_result,
  value = "likelihood",
  xlab = "Cut off", type = "b"
)
pc <- powerconsumption[powerconsumption$week %in% 10:12, ]
rdts <- cut(pc$active_power, breaks = c(0, quantile(pc$active_power, probs = c(0.5, 1))))
rdts_cov <- data.frame(day_night = (pc$hour >= 7 & pc$hour <= 17))
rdts_best_model_tune <- tune_covlmc(rdts, rdts_cov, criterion = "AIC")
plot(rdts_best_model_tune)
plot(rdts_best_model_tune, value = "likelihood")
```

positions	*Report the positions of a sequence associated to a node*

Description

This function returns the positions of the sequence represented by `node` in the time series used to build the context tree in which the sequence is represented. This is only possible is those positions were saved during the construction of the context tree. In positions were not saved, a call to this function produces an error.

Usage

```
positions(node)

## S3 method for class 'ctx_node'
positions(node)

## S3 method for class 'ctx_node_cpp'
positions(node)
```

Arguments

`node`	a `ctx_node` object as returned by `find_sequence()`

Details

A position of a sequence `ctx` in the time series `x` is an index value `t` such that the sequence ends with `x[t]`. Thus `x[t+1]` is after the context. For instance if `x=c(0, 0, 1, 1)` and `ctx=c(0, 1)` (in standard state order), then the position of `ctx` in `x` is 3.

Value

positions of the sequence represented by `node` is the original time series as a integer vector

Examples

```
rdts <- sample(as.factor(c("A", "B", "C")), 100, replace = TRUE)
rdts_tree <- ctx_tree(rdts, max_depth = 3, min_size = 5)
subseq <- find_sequence(rdts_tree, factor(c("B", "A"), levels = c("A", "B", "C")))
if (!is.null(subseq)) {
  positions(subseq)
}
```

`powerconsumption` *Individual household electric power consumption*

Description

A data set containing measurements of the electric power consumption of one household with a time resolution of 10 minutes for the full year of 2008.

Usage

```
powerconsumption
```

Format

A data frame with 52704 rows and 15 variables:

month month of 2008

month_day day of the month

hour hour (0 to 23)

minute starting minute of the 10 minutes period of this row

active_power global average active power on the 10 minute period (in kilowatt)

reactive_power global average reactive power on the 10 minute period (in kilowatt)

voltage Average voltage on the 10 minute period (in volt)

intensity global average current intensity on the 10 minute period (in ampere)

sub_metering_1 energy sub-metering No. 1 (in watt-hour of active energy averaged over the 10 minute period). It corresponds to the kitchen, containing mainly a dishwasher, an oven and a microwave (hot plates are not electric but gas powered)

sub_metering_2 energy sub-metering No. 2 (in watt-hour of active energy averaged over the 10 minute period). It corresponds to the laundry room, containing a washing-machine, a tumble-drier, a refrigerator and a light.

sub_metering_3 energy sub-metering No. 3 (in watt-hour of active energy averaged over the 10 minute period). It corresponds to an electric water-heater and an air-conditioner.

week week number

week_day day of the week from 1 = Sunday to 7 = Saturday

year_day day of the year from 1 to 366 (2008 is a leap year)

date_time Date and time in POSIXct format

Details

This is a simplified version of the full data available on the UCI Machine Learning Repository under a Creative Commons Attribution 4.0 International (CC BY 4.0) license, and provided by Georges Hebrail and Alice Berard.

The original data have been averaged over a 10 minute time period (discarding missing data in each period). The data set contains only the measurements from year 2008.

Notice that the different variables are expressed in the adapted units. In particular, the sub-meters are measuring active energy (in watt-hour) while the global active power is expressed in kilowatt.

Source

Individual household electric power consumption, 2012, G. Hebrail and A. Berard, UC Irvine Machine Learning repository. doi:10.24432/C58K54

`predict.covlmc` *Next state prediction in a discrete time series for a VLMC with covariates*

Description

This function computes one step ahead predictions for a discrete time series based on a VLMC with covariates.

Usage

```
## S3 method for class 'covlmc'
predict(
  object,
  newdata,
  newcov,
  type = c("raw", "probs"),
  final_pred = TRUE,
  ...
)
```

Arguments

`object`	a fitted covlmc object.
`newdata`	a time series adapted to the covlmc object.
`newcov`	a data frame with the new values for the covariates.
`type`	character indicating the type of prediction required. The default `"raw"` returns actual predictions in the form of a new time series. The alternative `"probs"` returns a matrix of prediction probabilities (see details).

`final_pred` if `TRUE` (default value), the predictions include a final prediction step, made by computing the context of the full time series. When `FALSE` this final prediction is not included.

`...` additional arguments.

Details

Given a time series `X`, at time step `t`, a context is computed using observations from `X[1]` to `X[t-1]` (see the dedicated section). The prediction is then the most probable state for `X[t]` given this logistic model of the context and the corresponding values of the covariates. The time series of predictions is returned by the function when `type="raw"` (default case).

When `type="probs"`, the function returns of the probabilities of each state for `X[t]` as estimated by the logistic models. Those probabilities are returned as a matrix of probabilities with column names given by the state names.

Value

A vector of predictions if `type="raw"` or a matrix of state probabilities if `type="probs"`.

Extended contexts

As explained in details in `loglikelihood.covlmc()` documentation and in the dedicated `vignette("likelihood", package = "mixvlmc")`, the first initial values of a time series do not in general have a proper context for a COVLMC with a non zero order. In order to predict something meaningful for those values, we rely on the notion of extended context defined in the documents mentioned above. This follows the same logic as using `loglikelihood.covlmc()` with the parameter `initial="extended"`. All covlmc functions that need to manipulate initial values with no proper context use the same approach.

Examples

```
pc <- powerconsumption[powerconsumption$week == 10, ]
rdts <- cut(pc$active_power, breaks = c(0, quantile(pc$active_power, probs = c(0.2, 0.7, 1))))
rdts_cov <- data.frame(day_night = (pc$hour >= 7 & pc$hour <= 17))
m_cov <- covlmc(rdts, rdts_cov, min_size = 5, alpha = 0.5)
rdts_probs <- predict(m_cov, rdts[1:144], rdts_cov[1:144, , drop = FALSE], type = "probs")
rdts_preds <- predict(m_cov, rdts[1:144], rdts_cov[1:144, , drop = FALSE],
  type = "raw", final_pred = FALSE
)
```

`predict.vlmc` *Next state prediction in a discrete time series for a VLMC*

Description

This function computes one step ahead predictions for a discrete time series based on a VLMC.

Usage

```
## S3 method for class 'vlmc'
predict(object, newdata, type = c("raw", "probs"), final_pred = TRUE, ...)

## S3 method for class 'vlmc_cpp'
predict(object, newdata, type = c("raw", "probs"), final_pred = TRUE, ...)
```

Arguments

`object`	a fitted vlmc object.
`newdata`	a time series adapted to the vlmc object.
`type`	character indicating the type of prediction required. The default `"raw"` returns actual predictions in the form of a new time series. The alternative `"probs"` returns a matrix of prediction probabilities (see details).
`final_pred`	if `TRUE` (default value), the predictions include a final prediction step, made by computing the context of the full time series. When `FALSE` this final prediction is not included.
`...`	additional arguments.

Details

Given a time series `X`, at time step `t`, a context is computed using observations from `X[1]` to `X[t-1]` (see the dedicated section). The prediction is then the most probable state for `X[t]` given this contexts. Ties are broken according to the natural order in the state space, favouring "small" values. The time series of predictions is returned by the function when `type="raw"` (default case).

When `type="probs"`, each `X[t]` is associated to the conditional probabilities of the next state given the context. Those probabilities are returned as a matrix of probabilities with column names given by the state names.

Value

A vector of predictions if `type="raw"` or a matrix of state probabilities if `type="probs"`.

Extended contexts

As explained in details in `loglikelihood.vlmc()` documentation and in the dedicated `vignette("likelihood", package = "mixvlmc")`, the first initial values of a time series do not in general have a proper context for a VLMC with a non zero order. In order to predict something meaningful for those values, we rely on the notion of extended context defined in the documents mentioned above. This follows the same logic as using `loglikelihood.vlmc()` with the parameter `initial="extended"`. All vlmc functions that need to manipulate initial values with no proper context use the same approach.

Examples

```
pc <- powerconsumption[powerconsumption$week == 5, ]
rdts <- cut(pc$active_power,
  breaks = c(0, quantile(pc$active_power,
    probs = c(0.25, 0.5, 0.75, 1)
  ))
)
model <- vlmc(rdts, min_size = 5)
predict(model, rdts[1:5])
predict(model, rdts[1:5], "probs")
## C++ backend
pc <- powerconsumption[powerconsumption$week == 5, ]
rdts <- cut(pc$active_power,
  breaks = c(0, quantile(pc$active_power,
    probs = c(0.25, 0.5, 0.75, 1)
  ))
)
model <- vlmc(rdts, min_size = 5, backend = "C++")
predict(model, rdts[1:5])
predict(model, rdts[1:5], "probs")
```

---

print.contexts *Print a context list*

Description

This function prints a list of contexts i.e. a `contexts` object listing `ctx_node` objects.

Usage

```
## S3 method for class 'contexts'
print(x, reverse = TRUE, ...)
```

Arguments

x	the `contexts` object to print
reverse	specifies whether the contexts should be reported in temporal order (`FALSE`, default value) or in reverse temporal order (`TRUE`). If the parameter is not specified, the contexts are displayed in order specified by the call to `contexts()` used to build the context list.
...	additional arguments for the print function.

Value

the `x` object, invisibly

See Also

`contexts()`

Examples

```
rdts <- c("A", "B", "C", "A", "A", "B", "B", "C", "C", "A")
rdts_tree <- ctx_tree(rdts, max_depth = 3)
print(contexts(rdts_tree))
```

---

print.dts *Print a discrete time series*

Description

This function prints a discrete time series.

Usage

```
## S3 method for class 'dts'
print(x, n = 5, ...)
```

Arguments

x	the dts object to print
n	the number of time steps of time series to print (defaults to 5)
...	additional arguments for the print function.

Value

the x object, invisibly

Examples

```
x_dts <- dts(sample(c("A", "B"), 20, replace = TRUE))
print(x_dts, n = 10)
```

---

prune *Prune a Variable Length Markov Chain (VLMC)*

Description

This function prunes a VLMC.

Usage

```
prune(vlmc, alpha = 0.05, cutoff = NULL, ...)

## S3 method for class 'vlmc'
prune(vlmc, alpha = 0.05, cutoff = NULL, ...)

## S3 method for class 'vlmc_cpp'
prune(vlmc, alpha = 0.05, cutoff = NULL, ...)
```

Arguments

`vlmc`	a fitted VLMC model.
`alpha`	number in (0,1] (default: 0.05) cut off value in quantile scale for pruning.
`cutoff`	positive number: cut off value in native (log likelihood ratio) scale for pruning. Defaults to the value obtained from `alpha`. Takes precedence over `alpha` if specified.
`...`	additional arguments for the prune function.

Details

In general, pruning a VLMC is more efficient than constructing two VLMC (the base one and pruned one). Up to numerical instabilities, building a VLMC with a `a` cut off and then pruning it with a `b` cut off (with `a>b`) should produce the same VLMC than building directly the VLMC with a `b` cut off. Interesting cut off values can be extracted from a VLMC using the `cutoff()` function.

As automated model selection is provided by `tune_vlmc()`, the direct use of `cutoff` should be reserved to advanced exploration of the set of trees that can be obtained from a complex one, e.g. to implement model selection techniques that are not provided by `tune_vlmc()`.

Value

a pruned VLMC

See Also

`cutoff()` and `tune_vlmc()`

Examples

```
pc <- powerconsumption[powerconsumption$week == 5, ]
rdts <- cut(pc$active_power,
  breaks = c(0, quantile(pc$active_power,
    probs = c(0.25, 0.5, 0.75, 1)
  ))
)
base_model <- vlmc(rdts, alpha = 0.1)
model_cuts <- cutoff(base_model)
pruned_model <- prune(base_model, model_cuts[3])
draw(pruned_model)
direct_simple <- vlmc(rdts, alpha = model_cuts[3])
draw(direct_simple)
# pruned_model and direct_simple should be identical
all.equal(pruned_model, direct_simple)
```

prune.covlmc *Prune a Variable Length Markov Chain with covariates*

Description

This function prunes a vlmc with covariates. This model must have been estimated with `keep_data=TRUE` to enable the pruning.

Usage

```
## S3 method for class 'covlmc'
prune(vlmc, alpha = 0.05, cutoff = NULL, ...)
```

Arguments

`vlmc`	a fitted VLMC model with covariates.
`alpha`	number in (0,1) (default: 0.05) cutoff value in quantile scale for pruning.
`cutoff`	not supported by the vlmc with covariates.
`...`	additional arguments for the prune function.

Details

Post pruning a VLMC with covariates is not as straightforward as the same procedure applied to `vlmc()` (see `cutoff.vlmc()` and `prune.vlmc()`). For efficiency reasons, `covlmc()` estimates only the logistic models that are considered useful for a given set construction parameters. With a more aggressive pruning threshold, some contexts become leaves of the context tree and new logistic models must be estimated. Thus the pruning opportunities given by `cutoff.covlmc()` are only a subset of interesting cut offs for a given covlmc.

Nevertheless, `covlmc` share with `vlmc()` the principle that post pruning a covlmc should give the same model as buidling directly the covlmc, provided that the post pruning alpha is smaller than the alpha used to build the initial model.

Value

a pruned covlmc.

Examples

```
pc <- powerconsumption[powerconsumption$week == 5, ]
rdts <- cut(pc$active_power, breaks = c(0, quantile(pc$active_power, probs = c(0.5, 1))))
rdts_cov <- data.frame(day_night = (pc$hour >= 7 & pc$hour <= 17))
m_cov <- covlmc(rdts, rdts_cov, min_size = 5, keep_data = TRUE)
draw(m_cov)
m_cov_cuts <- cutoff(m_cov)
p_cov <- prune(m_cov, m_cov_cuts[1])
draw(p_cov)
```

rev.ctx_node *Reverse Sequence*

Description

This function reverses the order in which the sequence represented by the `ctx_node` parameter will be reported in other functions, mainly `as_sequence()`.

Usage

```
## S3 method for class 'ctx_node'
rev(x)
```

Arguments

x	a `ctx_node` object as returned by `find_sequence()`

Value

a `ctx_node` using the opposite ordering convention as the parameter of the function

See Also

`is_reversed()`

Examples

```
rdts <- c("A", "B", "C", "A", "A", "B", "B", "C", "C", "A")
rdts_tree <- ctx_tree(rdts, max_depth = 3)
res <- find_sequence(rdts_tree, c("A", "B"))
print(res)
r_res <- rev(res)
print(r_res)
as_sequence(r_res)
```

simulate.covlmc *Simulate a discrete time series for a covlmc*

Description

This function simulates a time series from the distribution estimated by the given covlmc object.

Usage

```
## S3 method for class 'covlmc'
simulate(object, nsim = 1, seed = NULL, covariate, init = NULL, ...)
```

Arguments

`object`	a fitted covlmc object.
`nsim`	length of the simulated time series (defaults to 1).
`seed`	an optional random seed (see the dedicated section).
`covariate`	values of the covariates.
`init`	an optional initial sequence for the time series given by an object that can be interpreted as a discrete time series.
`...`	additional arguments.

Details

A VLMC with covariates model needs covariates to compute its transition probabilities. The covariates must be submitted as a data frame using the `covariate` argument. In addition, the time series can be initiated by a fixed sequence specified via the `init` parameter.

Value

a simulated discrete time series of the same type as the one used to build the covlmc with a `seed` attribute (see the Random seed section). The results has also the `dts_simulated` class to hide the `seed` attribute when using `print` or similar function.

Extended contexts

As explained in details in `loglikelihood.covlmc()` documentation and in the dedicated `vignette("likelihood", package = "mixvlmc")`, the first initial values of a time series do not in general have a proper context for a COVLMC with a non zero order. In order to simulate something meaningful for those values, we rely on the notion of extended context defined in the documents mentioned above. This follows the same logic as using `loglikelihood.covlmc()` with the parameter `initial="extended"`. All covlmc functions that need to manipulate initial values with no proper context use the same approach.

Random seed

This function reproduce the behaviour of `stats::simulate()`. If `seed` is `NULL` the function does not change the random generator state and returns the value of .Random.seed as a `seed` attribute in the return value. This can be used to reproduce exactly the simulation results by setting .Random.seed to this value. Notice that if the random seed has not be initialised by R so far, the function issues a call to `runif(1)` to perform this initialisation (as is done in `stats::simulate()`).

It `seed` is an integer, it is used in a call to `set.seed()` before the simulation takes place. The integer is saved as a `seed` attribute in the return value. The integer seed is completed by an attribute `kind` which contains the value `as.list([RNGkind()])` exactly as with `stats::simulate()`. The random generator state is reset to its original value at the end of the call.

See Also

`stats::simulate()` for details and examples on the random number generator setting

Examples

```
pc <- powerconsumption[powerconsumption$week == 5, ]
rdts <- cut(pc$active_power, breaks = c(0, quantile(pc$active_power, probs = c(0.5, 1))))
rdts_cov <- data.frame(day_night = (pc$hour >= 7 & pc$hour <= 17))
m_cov <- covlmc(rdts, rdts_cov, min_size = 5)
# new week with day light from 6:00 to 18:00
new_cov <- data.frame(day_night = rep(c(rep(FALSE, 59), rep(TRUE, 121), rep(FALSE, 60)), times = 7))
new_rdts <- simulate(m_cov, nrow(new_cov), seed = 0, covariate = new_cov)
new_rdts_2 <- simulate(m_cov, nrow(new_cov), seed = 0, covariate = new_cov, init = rdts[1:10])
```

---

simulate.vlmc — *Simulate a discrete time series for a vlmc*

Description

This function simulates a time series from the distribution estimated by the given vlmc object.

Usage

```
## S3 method for class 'vlmc'
simulate(object, nsim = 1L, seed = NULL, init = NULL, burnin = 0L, ...)
```

Arguments

`object`	a fitted vlmc object.
`nsim`	length of the simulated time series (defaults to 1).
`seed`	an optional random seed (see the dedicated section).
`init`	an optional initial sequence for the time series given by an object that can be interpreted as a discrete time series.
`burnin`	number of initial observations to discard or `"auto"` (see the dedicated section).
`...`	additional arguments.

Details

The time series can be initiated by a fixed sequence specified via the `init` parameter.

Value

a simulated discrete time series of the same type as the one used to build the vlmc with a `seed` attribute (see the Random seed section). The results has also the `dts_simulated` class to hide the `seed` attribute when using `print` or similar function.

Burn in (Warm up) period

When using a VLMC for simulation purposes, we are generally interested in the stationary distribution of the corresponding Markov chain. To reduce the dependence of the samples from the initial values and get closer to this stationary distribution (if it exists), it is recommended to discard the first samples which are produced in a so-called "burn in" (or "warm up") period. The `burnin` parameter can be used to implement this approach. The VLMC is used to produce a sample of size `burnin + nsim` but the first `burnin` values are discarded. Notice that this burn in values can be partially given by the `init` parameter if it is specified.

If `burnin` is set to `"auto"`, the `burnin` period is set to `64 * context_number(object)`, following the heuristic proposed in Mächler and Bühlmann (2004).

Random seed

This function reproduce the behaviour of `stats::simulate()`. If `seed` is `NULL` the function does not change the random generator state and returns the value of .Random.seed as a `seed` attribute in the return value. This can be used to reproduce exactly the simulation results by setting .Random.seed to this value. Notice that if the random seed has not be initialised by R so far, the function issues a call to `runif(1)` to perform this initialisation (as is done in `stats::simulate()`).

It `seed` is an integer, it is used in a call to `set.seed()` before the simulation takes place. The integer is saved as a `seed` attribute in the return value. The integer seed is completed by an attribute `kind` which contains the value `as.list([RNGkind()])` exactly as with `stats::simulate()`. The random generator state is reset to its original value at the end of the call.

Extended contexts

As explained in details in `loglikelihood.vlmc()` documentation and in the dedicated `vignette("likelihood", package = "mixvlmc")`, the first initial values of a time series do not in general have a proper context for a VLMC with a non zero order. In order to simulate something meaningful for those values when `init` is not provided, we rely on the notion of extended context defined in the documents mentioned above. This follows the same logic as using `loglikelihood.vlmc()` with the parameter `initial="extended"`. All vlmc functions that need to manipulate initial values with no proper context use the same approach.

References

Mächler, M. and Bühlmann, P. (2004) "Variable Length Markov Chains: Methodology, Computing, and Software" Journal of Computational and Graphical Statistics, 13 (2), 435-455, doi:10.1198/1061860043524

See Also

`stats::simulate()` for details and examples on the random number generator setting

Examples

```
pc <- powerconsumption[powerconsumption$week == 5, ]
rdts <- cut(pc$active_power,
  breaks = c(0, quantile(pc$active_power,
    probs = c(0.25, 0.5, 0.75, 1)
  ))
)
model <- vlmc(rdts, min_size = 5)
new_rdts <- simulate(model, 500, seed = 0)
new_rdts_2 <- simulate(model, 500, seed = 0, init = rdts[1:5])
new_rdts_3 <- simulate(model, 500, seed = 0, burnin = 500)
```

---

simulate.vlmc_cpp *Simulate a discrete time series for a vlmc*

Description

This function simulates a time series from the distribution estimated by the given vlmc object.

Usage

```
## S3 method for class 'vlmc_cpp'
simulate(
  object,
  nsim = 1,
  seed = NULL,
  init = NULL,
  burnin = 0L,
  sample = c("fast", "slow", "R"),
  ...
)
```

Arguments

`object`	a fitted vlmc object.
`nsim`	length of the simulated time series (defaults to 1).
`seed`	an optional random seed (see the dedicated section).
`init`	an optional initial sequence for the time series given by an object that can be interpreted as a discrete time series.
`burnin`	number of initial observations to discard or `"auto"` (see the dedicated section).
`sample`	specifies which implementation of `base::sample()` to use. See the dedicated section.
`...`	additional arguments.

Details

The time series can be initiated by a fixed sequence specified via the `init` parameter.

Value

a simulated discrete time series of the same type as the one used to build the vlmc with a `seed` attribute (see the Random seed section). The results has also the `dts_simulated` class to hide the `seed` attribute when using `print` or similar function.

sampling method

The R backend for `vlmc()` uses `base::sample()` to generate samples for each context. Internally, this function sorts the probabilities of each state in decreasing probability order (among other things), which is not needed in our case. The C++ backend can be used with three different implementations:

- `sample="fast"` uses a dedicated C++ implementation adapted to the data structures used internally. In general, the simulated time series obtained with this implementation will be different from the one generated with the R backend, even using the same seed.
- `sample="slow"` uses another C++ implementation that mimics `base::sample()` in order to maximize the chance to provide identical simulation results regardless of the backend (when using the same random seed). This process is not perfect as we use the std::lib sort algorithm which is not guaranteed to give identical results as the ones of R internal 'revsort'.
- `sample="R"` uses direct calls to `base::sample()`. Results are guaranteed to be identical between the two backends, but at the price of higher running time.

Burn in (Warm up) period

When using a VLMC for simulation purposes, we are generally interested in the stationary distribution of the corresponding Markov chain. To reduce the dependence of the samples from the initial values and get closer to this stationary distribution (if it exists), it is recommended to discard the first samples which are produced in a so-called "burn in" (or "warm up") period. The `burnin` parameter can be used to implement this approach. The VLMC is used to produce a sample of size `burnin + nsim` but the first `burnin` values are discarded. Notice that this burn in values can be partially given by the `init` parameter if it is specified.

If `burnin` is set to `"auto"`, the `burnin` period is set to `64 * context_number(object)`, following the heuristic proposed in Mächler and Bühlmann (2004).

Random seed

This function reproduce the behaviour of `stats::simulate()`. If `seed` is `NULL` the function does not change the random generator state and returns the value of .Random.seed as a `seed` attribute in the return value. This can be used to reproduce exactly the simulation results by setting .Random.seed to this value. Notice that if the random seed has not be initialised by R so far, the function issues a call to `runif(1)` to perform this initialisation (as is done in `stats::simulate()`).

It `seed` is an integer, it is used in a call to `set.seed()` before the simulation takes place. The integer is saved as a `seed` attribute in the return value. The integer seed is completed by an attribute `kind` which contains the value `as.list([RNGkind()])` exactly as with `stats::simulate()`. The random generator state is reset to its original value at the end of the call.

Extended contexts

As explained in details in `loglikelihood.vlmc()` documentation and in the dedicated `vignette("likelihood", package = "mixvlmc")`, the first initial values of a time series do not in general have a proper context for a VLMC with a non zero order. In order to simulate something meaningful for those values when `init` is not provided, we rely on the notion of extended context defined in the documents mentioned above. This follows the same logic as using `loglikelihood.vlmc()` with the parameter `initial="extended"`. All vlmc functions that need to manipulate initial values with no proper context use the same approach.

References

Mächler, M. and Bühlmann, P. (2004) "Variable Length Markov Chains: Methodology, Computing, and Software" Journal of Computational and Graphical Statistics, 13 (2), 435-455, doi:10.1198/1061860043524

See Also

`stats::simulate()` for details and examples on the random number generator setting

Examples

```
pc <- powerconsumption[powerconsumption$week == 5, ]
rdts <- cut(pc$active_power,
  breaks = c(0, quantile(pc$active_power,
    probs = c(0.25, 0.5, 0.75, 1)
  ))
)
model <- vlmc(rdts, min_size = 5)
new_rdts <- simulate(model, 500, seed = 0)
new_rdts_2 <- simulate(model, 500, seed = 0, init = rdts[1:5])
new_rdts_3 <- simulate(model, 500, seed = 0, burnin = 500)
```

`states` *State space of an object*

Description

This function returns the state space of an object for which this is meaningful such as a discrete time series or a context tree.

Usage

```
states(x)

## S3 method for class 'ctx_tree'
states(x)

## S3 method for class 'dts'
states(x)
```

Arguments

`x`	an object with a state space.

Value

the state space of the context tree.

Examples

```
rdts <- c(0, 1, 1, 1, 0, 0, 1, 0, 1, 0)
rdts_ctree <- ctx_tree(rdts, min_size = 1, max_depth = 2)
## should be c(0, 1)
states(rdts_ctree)
x_dts <- dts(sample(c("A", "B", "C"), 20, replace = TRUE))
## should be c("A", "B", "C")
states(x_dts)
```

---

trim *Trim a context tree*

Description

This function returns a trimmed context tree from which match positions have been removed.

Usage

```
trim(ct, ...)
```

Arguments

`ct`	a context tree.
`...`	additional arguments for the trim function.

Value

a trimmed context tree.

Examples

```
## context tree trimming
rdts <- sample(as.factor(c("A", "B", "C")), 1000, replace = TRUE)
rdts_tree <- ctx_tree(rdts, max_depth = 10, min_size = 5, keep_position = TRUE)
print(object.size(rdts_tree))
rdts_tree <- trim(rdts_tree)
print(object.size(rdts_tree))
```

`trim.covlmc` *Trim a COVLMC*

Description

This function returns a trimmed COVLMC from which cached data have been removed.

Usage

```
## S3 method for class 'covlmc'
trim(ct, keep_model = FALSE, ...)
```

Arguments

`ct`	a context tree.
`keep_model`	specifies whether to keep the internal models (or not)
`...`	additional arguments for the trim function.

Details

Called with `keep_model` set to `FALSE` (default case), the trimming is maximal and reduces further usability of the model. In particular `loglikelihood.covlmc()` cannot be used for new data, `contexts.covlmc()` do not support model extraction, and `simulate.covlmc()`, `metrics.covlmc()` and `prune.covlmc()` cannot be used at all.

Called with `keep_model` set to `TRUE`, the trimming process is less complete. In particular internal models are simplified using `butcher::butcher()` and some additional minor reductions. This saves less memory but enables the use of `loglikelihood.covlmc()` for new data as well as the use of `simulate.covlmc()`.

Value

a trimmed context tree.

See Also

`tune_covlmc()`

Examples

```
pc <- powerconsumption[powerconsumption$week %in% 5:7, ]
rdts <- cut(pc$active_power, breaks = c(0, quantile(pc$active_power, probs = c(0.5, 1))))
rdts_cov <- data.frame(day_night = (pc$hour >= 7 & pc$hour <= 17))
m_cov <- covlmc(rdts, rdts_cov, min_size = 10, keep_data = TRUE)
print(object.size(m_cov), units = "Mb")
t_m_cov_model <- trim(m_cov, keep_model = TRUE)
print(object.size(t_m_cov_model), units = "Mb")
t_m_cov <- trim(m_cov)
print(object.size(t_m_cov), units = "Mb")
```

`trim.vlmc` *This function returns a trimmed VLMC from which match positions have been removed.*

Description

This function returns a trimmed context tree from which match positions have been removed.

Usage

```
## S3 method for class 'vlmc'
trim(ct, ...)
```

Arguments

`ct`	a VLMC.
`...`	additional arguments for the trim function.

Value

a trimmed VLMC

Examples

```
## VLMC trimming is generally useless unless match positions were kept
pc <- powerconsumption[powerconsumption$week %in% 5:6, ]
rdts <- cut(pc$active_power, breaks = 4)
model <- vlmc(rdts, keep_match = TRUE)
print(object.size(model))
model <- trim(model)
## memory use should be reduced
print(object.size(model))
nm_model <- vlmc(rdts)
print(object.size(nm_model))
nm_model <- trim(nm_model)
## no effect when match positions are not kept
print(object.size(nm_model))
```

`trim.vlmc_cpp` *This function returns a trimmed VLMC from which match positions have been removed.*

Description

This function returns a trimmed context tree from which match positions have been removed.

Usage

```
## S3 method for class 'vlmc_cpp'
trim(ct, ...)
```

Arguments

`ct` a VLMC.

`...` additional arguments for the trim function.

Details

Trimming in the C++ backend is done directly in the `Rcpp` managed memory and cannot be detected at R level using e.g. `utils::object.size()`.

Value

a trimmed VLMC

Examples

```
## VLMC trimming is generally useless unless match positions were kept
pc <- powerconsumption[powerconsumption$week %in% 5:6, ]
rdts <- cut(pc$active_power, breaks = 4)
model <- vlmc(rdts, backend = "C++", keep_match = TRUE)
model <- trim(model)
```

`tune_covlmc` *Fit an optimal Variable Length Markov Chain with Covariates (coVLMC)*

Description

This function fits a Variable Length Markov Chain with Covariates (coVLMC) to a discrete time series coupled with a time series of covariates by optimizing an information criterion (BIC or AIC).

Usage

```
tune_covlmc(
  x,
  covariate,
  criterion = c("BIC", "AIC"),
  initial = c("truncated", "specific", "extended"),
  alpha_init = NULL,
  min_size = 5,
  max_depth = 100,
  verbose = 0,
  save = c("best", "initial", "all"),
  trimming = c("full", "partial", "none"),
  best_trimming = c("none", "partial", "full")
)
```

Arguments

`x`	an object that can be interpreted as a discrete time series, such as an integer vector or a `dts` object (see `dts()`).
`covariate`	a data frame of covariates.
`criterion`	criterion used to select the best model. Either `"BIC"` (default) or `"AIC"` (see details).
`initial`	specifies the likelihood function, more precisely the way the first few observations for which contexts cannot be calculated are integrated in the likelihood. See `loglikelihood()` for details.
`alpha_init`	if non `NULL` used as the initial cut off parameter (in quantile scale) to build the initial VLMC
`min_size`	integer >= 1 (default: 5). Tune the minimum number of observations for a context in the growing phase of the context tree (see `covlmc()` for details).
`max_depth`	integer >= 1 (default: 100). Longest context considered in growing phase of the initial context tree (see details).
`verbose`	integer >= 0 (default: 0). Verbosity level of the pruning process.
`save`	specify which BIC models are saved during the pruning process. The default value `"best"` asks the function to keep only the best model according to the `criterion`. When `save="initial"` the function keeps *in addition* the initial (complex) model which is then pruned during the selection process. When `save="all"`, the function returns all the models considered during the selection process. See details for memory occupation.
`trimming`	specify the type of trimming used when saving the intermediate models, see details.
`best_trimming`	specify the type of trimming used when saving the best model and the initial one (see details).

Details

This function automates the process of fitting a large coVLMC to a discrete time series with covlmc() and of pruning the tree (with cutoff() and prune()) to get an optimal with respect to an information criterion. To avoid missing long term dependencies, the function uses the max_depth parameter as an initial guess but then relies on an automatic increase of the value to make sure the initial context tree is only limited by the min_size parameter. The initial value of the alpha parameter of covlmc() is also set to a conservative value (0.5) to avoid prior simplification of the context tree. This can be overridden by setting the alpha_init parameter to a more adapted value.

Once the initial coVLMC is obtained, the cutoff() and prune() functions are used to build all the coVLMC models that could be generated using smaller values of the alpha parameter. The best model is selected from this collection, including the initial complex tree, as the one that minimizes the chosen information criterion.

Value

a list with the following components:

- best_model: the optimal COVLMC
- criterion: the criterion used to select the optimal VLMC
- initial: the likelihood function used to select the optimal VLMC
- results: a data frame with details about the pruning process
- saved_models: a list of intermediate COVLMCs if save="initial" or save="all". It contains an initial component with the large coVLMC obtained first and an all component with a list of all the *other* coVLMC obtained by pruning the initial one.

Memory occupation

covlmc objects tend to be large and saving all the models during the search for the optimal model can lead to an unreasonable use of memory. To avoid this problem, models are kept in trimmed form only using trim.covlmc() with keep_model=FALSE. Both the initial model and the best one are saved untrimmed. This default behaviour corresponds to trimming="full". Setting trimming="partial" asks the function to use keep_model=TRUE in trim.covlmc() for intermediate models. Finally, trimming="none" turns off trimming, which is discouraged expected for small data sets.

In parallel processing contexts (e.g. using foreach::%dopar%), the memory occupation of the results can become very large as models tend to keep environments attached to the formulas. In this situation, it is highly recommended to trim all saved models, including the best one and the initial one. This can be done via the best_trimming parameter whose possible values are identical to the ones of trimming.

See Also

covlmc(), cutoff() and prune()

Examples

```
pc <- powerconsumption[powerconsumption$week %in% 6:7, ]
rdts <- cut(pc$active_power, breaks = c(0, quantile(pc$active_power, probs = c(0.5, 1))))
rdts_cov <- data.frame(day_night = (pc$hour >= 7 & pc$hour <= 17))
rdts_best_model_tune <- tune_covlmc(rdts, rdts_cov)
draw(as_covlmc(rdts_best_model_tune))
```

`tune_vlmc` *Fit an optimal Variable Length Markov Chain (VLMC)*

Description

This function fits a Variable Length Markov Chain (VLMC) to a discrete time series by optimizing an information criterion (BIC or AIC).

Usage

```
tune_vlmc(
  x,
  criterion = c("BIC", "AIC"),
  initial = c("truncated", "specific", "extended"),
  alpha_init = NULL,
  cutoff_init = NULL,
  min_size = 2L,
  max_depth = 100L,
  backend = getOption("mixvlmc.backend", "R"),
  verbose = 0,
  save = c("best", "initial", "all")
)
```

Arguments

`x`	an object that can be interpreted as a discrete time series, such as an integer vector or a `dts` object (see `dts()`).
`criterion`	criterion used to select the best model. Either `"BIC"` (default) or `"AIC"` (see details).
`initial`	specifies the likelihood function, more precisely the way the first few observations for which contexts cannot be calculated are integrated in the likelihood. Default to `"truncated"`. See `loglikelihood()` for details.
`alpha_init`	if non `NULL` used as the initial cut off parameter (in quantile scale) to build the initial VLMC
`cutoff_init`	if non `NULL` used as the initial cut off parameter to build the initial VLMC. Takes precedence over `alpha_init` if specified.
`min_size`	integer >= 1 (default: 2). Minimum number of observations for a context in the growing phase of the initial context tree.

max_depth integer >= 1 (default: 100). Longest context considered in growing phase of the initial context tree (see details).

backend backend "R" or "C++" (default: as specified by the "mixvlmc.backend" option). Specifies the implementation used to represent the context tree and to built it. See `vlmc()` for details.

verbose integer >= 0 (default: 0). Verbosity level of the pruning process.

save specify which BIC models are saved during the pruning process. The default value `"best"` asks the function to keep only the best model according to the `criterion`. When `save="initial"` the function keeps *in addition* the initial (complex) model which is then pruned during the selection process. When `save="all"`, the function returns all the models considered during the selection process.

Details

This function automates the process of fitting a large VLMC to a discrete time series with `vlmc()` and of pruning the tree (with `cutoff()` and `prune()`) to get an optimal with respect to an information criterion. To avoid missing long term dependencies, the function uses the `max_depth` parameter as an initial guess but then relies on an automatic increase of the value to make sure the initial context tree is only limited by the `min_size` parameter. The initial value of the `cutoff` parameter of `vlmc()` is also set to conservative values (depending on the criterion) to avoid prior simplification of the context tree. This default value can be overridden using the `cutoff_init` or `alpha_init` parameter.

Once the initial VLMC is obtained, the `cutoff()` and `prune()` functions are used to build all the VLMC models that could be generated using larger values of the initial cut off parameter. The best model is selected from this collection, including the initial complex tree, as the one that minimizes the chosen information criterion.

Value

a list with the following components:

- `best_model`: the optimal VLMC
- `criterion`: the criterion used to select the optimal VLMC
- `initial`: the likelihood function used to select the optimal VLMC
- `results`: a data frame with details about the pruning process
- `saved_models`: a list of intermediate VLMCs if `save="initial"` or `save="all"`. It contains an `initial` component with the large VLMC obtained first and an `all` component with a list of all the *other* VLMC obtained by pruning the initial one.

See Also

`vlmc()`, `cutoff()` and `prune()`

Examples

```
rdts <- sample(as.factor(c("A", "B", "C")), 100, replace = TRUE)
tune_result <- tune_vlmc(rdts)
draw(tune_result$best_model)
```

`vlmc` *Fit a Variable Length Markov Chain (VLMC)*

Description

This function fits a Variable Length Markov Chain (VLMC) to a discrete time series.

Usage

```
vlmc(
  x,
  alpha = 0.05,
  cutoff = NULL,
  min_size = 2L,
  max_depth = 100L,
  prune = TRUE,
  keep_match = FALSE,
  backend = getOption("mixvlmc.backend", "R")
)
```

Arguments

`x`	an object that can be interpreted as a discrete time series, such as an integer vector or a `dts` object (see `dts()`)
`alpha`	number in (0,1] (default: 0.05) cut off value in quantile scale in the pruning phase.
`cutoff`	non negative number: cut off value in native (likelihood ratio) scale in the pruning phase. Defaults to the value obtained from `alpha`. Takes precedence over `alpha` is specified.
`min_size`	integer >= 1 (default: 2). Minimum number of observations for a context in the growing phase of the context tree.
`max_depth`	integer >= 1 (default: 100). Longest context considered in growing phase of the context tree.
`prune`	logical: specify whether the context tree should be pruned (default behaviour).
`keep_match`	logical: specify whether to keep the context matches (default to FALSE)
`backend`	"R" or "C++" (default: as specified by the "mixvlmc.backend" option). Specifies the implementation used to represent the context tree and to built it. See details.

Details

The VLMC is built using Bühlmann and Wyner's algorithm which consists in fitting a context tree (see `ctx_tree()`) to a time series and then pruning it in such as way that the conditional distribution of the next state of the time series given the context is significantly different from the distribution given a truncated version of the context.

The construction of the context tree is controlled by `min_size` and `max_depth`, exactly as in `ctx_tree()`. Significativity is measured using a likelihood ratio test (threshold can be specified in terms of the ratio itself with `cutoff`) or in quantile scale with `alpha`.

Pruning can be postponed by setting `prune=FALSE`. Using a combination of `cutoff()` and `prune()`, the complexity of the VLMC can then be adjusted. Any VLMC model can be pruned after construction, `prune=FALSE` is a convenience parameter to avoid setting `alpha=1` (which essentially prevents any pruning). Automated model selection is provided by `tune_vlmc()`.

Value

a fitted vlmc model.

Back ends

Two back ends are available to compute context trees:

- the "R" back end represents the tree in pure R data structures (nested lists) that be easily processed further in pure R (C++ helper functions are used to speed up the construction).
- the "C++" back end represents the tree with C++ classes. This back end is considered experimental. The tree is built with an optimised suffix tree algorithm which speeds up the construction by at least a factor 10 in standard settings. As the tree is kept outside of R direct reach, context trees built with the C++ back end must be restored after a `saveRDS()`/`readRDS()` sequence. This is done automatically by recomputing completely the context tree.

References

Bühlmann, P. and Wyner, A. J. (1999), "Variable length Markov chains. Ann. Statist." 27 (2) 480-513 doi:10.1214/aos/1018031204

See Also

`cutoff()`, `prune()` and `tune_vlmc()`

Examples

```
pc <- powerconsumption[powerconsumption$week == 5, ]
rdts <- cut(pc$active_power,
  breaks = c(0, quantile(pc$active_power, probs = c(0.25, 0.5, 0.75, 1)))
)
model <- vlmc(rdts)
draw(model)
depth(model)
## reduce the depth of the model
shallow_model <- vlmc(rdts, max_depth = 3)
draw(shallow_model, prob = FALSE)
```

```
## improve probability estimates
robust_model <- vlmc(rdts, min_size = 25)
draw(robust_model, prob = FALSE) ## show the frequencies
draw(robust_model)
```

`vlmc.default` *Fit a Variable Length Markov Chain (VLMC)*

Description

This function fits a Variable Length Markov Chain (VLMC) to a discrete time series.

Usage

```
## Default S3 method:
vlmc(
  x,
  alpha = 0.05,
  cutoff = NULL,
  min_size = 2L,
  max_depth = 100L,
  prune = TRUE,
  keep_match = FALSE,
  backend = getOption("mixvlmc.backend", "R")
)
```

Arguments

`x`	a numeric, character, factor or logical vector
`alpha`	number in (0,1] (default: 0.05) cut off value in quantile scale in the pruning phase.
`cutoff`	non negative number: cut off value in native (likelihood ratio) scale in the pruning phase. Defaults to the value obtained from `alpha`. Takes precedence over `alpha` is specified.
`min_size`	integer >= 1 (default: 2). Minimum number of observations for a context in the growing phase of the context tree.
`max_depth`	integer >= 1 (default: 100). Longest context considered in growing phase of the context tree.
`prune`	logical: specify whether the context tree should be pruned (default behaviour).
`keep_match`	logical: specify whether to keep the context matches (default to FALSE)
`backend`	"R" or "C++" (default: as specified by the "mixvlmc.backend" option). Specifies the implementation used to represent the context tree and to built it. See details.

Details

The VLMC is built using Bühlmann and Wyner's algorithm which consists in fitting a context tree (see `ctx_tree()`) to a time series and then pruning it in such as way that the conditional distribution of the next state of the time series given the context is significantly different from the distribution given a truncated version of the context.

The construction of the context tree is controlled by `min_size` and `max_depth`, exactly as in `ctx_tree()`. Significativity is measured using a likelihood ratio test (threshold can be specified in terms of the ratio itself with `cutoff`) or in quantile scale with `alpha`.

Pruning can be postponed by setting `prune=FALSE`. Using a combination of `cutoff()` and `prune()`, the complexity of the VLMC can then be adjusted. Any VLMC model can be pruned after construction, `prune=FALSE` is a convenience parameter to avoid setting `alpha=1` (which essentially prevents any pruning). Automated model selection is provided by `tune_vlmc()`.

Value

a fitted vlmc model.

Back ends

Two back ends are available to compute context trees:

- the "R" back end represents the tree in pure R data structures (nested lists) that be easily processed further in pure R (C++ helper functions are used to speed up the construction).
- the "C++" back end represents the tree with C++ classes. This back end is considered experimental. The tree is built with an optimised suffix tree algorithm which speeds up the construction by at least a factor 10 in standard settings. As the tree is kept outside of R direct reach, context trees built with the C++ back end must be restored after a `saveRDS()`/`readRDS()` sequence. This is done automatically by recomputing completely the context tree.

References

Bühlmann, P. and Wyner, A. J. (1999), "Variable length Markov chains. Ann. Statist." 27 (2) 480-513 doi:10.1214/aos/1018031204

See Also

`cutoff()`, `prune()` and `tune_vlmc()`

Examples

```
pc <- powerconsumption[powerconsumption$week == 5, ]
rdts <- cut(pc$active_power,
  breaks = c(0, quantile(pc$active_power, probs = c(0.25, 0.5, 0.75, 1)))
)
model <- vlmc(rdts)
draw(model)
```

```
depth(model)
## reduce the depth of the model
shallow_model <- vlmc(rdts, max_depth = 3)
draw(shallow_model, prob = FALSE)
## improve probability estimates
robust_model <- vlmc(rdts, min_size = 25)
draw(robust_model, prob = FALSE) ## show the frequencies
draw(robust_model)
```

`vlmc.dts` *Fit a Variable Length Markov Chain (VLMC)*

Description

This function fits a Variable Length Markov Chain (VLMC) to a discrete time series.

Usage

```
## S3 method for class 'dts'
vlmc(
  x,
  alpha = 0.05,
  cutoff = NULL,
  min_size = 2L,
  max_depth = 100L,
  prune = TRUE,
  keep_match = FALSE,
  backend = getOption("mixvlmc.backend", "R")
)
```

Arguments

`x`	a discrete time series represented by a `dts` object as created by `dts()`
`alpha`	number in (0,1] (default: 0.05) cut off value in quantile scale in the pruning phase.
`cutoff`	non negative number: cut off value in native (likelihood ratio) scale in the pruning phase. Defaults to the value obtained from `alpha`. Takes precedence over `alpha` is specified.
`min_size`	integer >= 1 (default: 2). Minimum number of observations for a context in the growing phase of the context tree.
`max_depth`	integer >= 1 (default: 100). Longest context considered in growing phase of the context tree.
`prune`	logical: specify whether the context tree should be pruned (default behaviour).
`keep_match`	logical: specify whether to keep the context matches (default to FALSE)
`backend`	"R" or "C++" (default: as specified by the "mixvlmc.backend" option). Specifies the implementation used to represent the context tree and to built it. See details.

Details

The VLMC is built using Bühlmann and Wyner's algorithm which consists in fitting a context tree (see `ctx_tree()`) to a time series and then pruning it in such as way that the conditional distribution of the next state of the time series given the context is significantly different from the distribution given a truncated version of the context.

The construction of the context tree is controlled by `min_size` and `max_depth`, exactly as in `ctx_tree()`. Significativity is measured using a likelihood ratio test (threshold can be specified in terms of the ratio itself with `cutoff`) or in quantile scale with `alpha`.

Pruning can be postponed by setting `prune=FALSE`. Using a combination of `cutoff()` and `prune()`, the complexity of the VLMC can then be adjusted. Any VLMC model can be pruned after construction, `prune=FALSE` is a convenience parameter to avoid setting `alpha=1` (which essentially prevents any pruning). Automated model selection is provided by `tune_vlmc()`.

Value

a fitted vlmc model.

Back ends

Two back ends are available to compute context trees:

- the "R" back end represents the tree in pure R data structures (nested lists) that be easily processed further in pure R (C++ helper functions are used to speed up the construction).
- the "C++" back end represents the tree with C++ classes. This back end is considered experimental. The tree is built with an optimised suffix tree algorithm which speeds up the construction by at least a factor 10 in standard settings. As the tree is kept outside of R direct reach, context trees built with the C++ back end must be restored after a `saveRDS()/readRDS()` sequence. This is done automatically by recomputing completely the context tree.

References

Bühlmann, P. and Wyner, A. J. (1999), "Variable length Markov chains. Ann. Statist." 27 (2) 480-513 doi:10.1214/aos/1018031204

See Also

`cutoff()`, `prune()` and `tune_vlmc()`

Examples

```
pc <- powerconsumption[powerconsumption$week == 5, ]
power_dts <- dts(cut(pc$active_power,
  breaks = c(0, quantile(pc$active_power, probs = c(0.25, 0.5, 0.75, 1)))
))
model <- vlmc(power_dts)
draw(model)
```

```
depth(model)
## reduce the depth of the model
shallow_model <- vlmc(power_dts, max_depth = 3)
draw(shallow_model, prob = FALSE)
## improve probability estimates
robust_model <- vlmc(power_dts, min_size = 25)
draw(robust_model, prob = FALSE) ## show the frequencies
draw(robust_model)
```

Index