Package ‘stagedtrees’

July 23, 2025

Type Package

Title Staged Event Trees

Version 2.3.0

Description Creates and fits staged event tree probability models,
which are probabilistic graphical models capable of representing
asymmetric conditional independence statements
for categorical variables.
Includes functions to create, plot and fit staged
event trees from data, as well as many efficient structure
learning algorithms.
References:
Carli F, Leonelli M, Riccomagno E, Varando G (2022).
<doi:10.18637/jss.v102.i06>.
Collazo R. A., Görgen C. and Smith J. Q.
(2018, ISBN:9781498729604).
Görgen C., Bigatti A., Riccomagno E. and Smith J. Q. (2018)
<doi:10.48550/arXiv.1705.09457>.
Thwaites P. A., Smith, J. Q. (2017) <doi:10.48550/arXiv.1510.00186>.
Barclay L. M., Hutton J. L. and Smith J. Q. (2013)
<doi:10.1016/j.ijar.2013.05.006>.
Smith J. Q. and Anderson P. E. (2008)
<doi:10.1016/j.artint.2007.05.004>.

License MIT + file LICENSE

Encoding UTF-8

LazyData true

RoxygenNote 7.3.1

URL https://github.com/stagedtrees/stagedtrees

BugReports https://github.com/stagedtrees/stagedtrees/issues

Imports stats, graphics, cli, rlang, matrixStats

Suggests testthat (>= 3.0.0), bnlearn, covr, clue, igraph

Config/testthat/edition 3

Depends R (>= 2.10)

NeedsCompilation no

Author Gherardo Varando [aut, cre] (ORCID:
<https://orcid.org/0000-0002-6708-1103>),
Federico Carli [aut],
Manuele Leonelli [aut] (ORCID: <https://orcid.org/0000-0002-2562-5192>),
Eva Riccomagno [aut]

Maintainer Gherardo Varando <gherardo.varando@gmail.com>

Repository CRAN

Date/Publication 2024-02-14 11:50:02 UTC

Contents

`as.character.parentslist` *Print a parentslist object*

Description

Nice print of a parentslist object

Usage

```
## S3 method for class 'parentslist'
as.character(x, only_parents = FALSE, ...)

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

Arguments

`x`	an object of class `parentslist`.
`only_parents`	logical, if the basic DAG encoding is to be returned.
`...`	additional arguments for compatibility.

Value

`as.character.parentslist` returns a string encoding the associated directed graph and eventually the context specific independences. The encoding is similar to the one returned by `modelstring` in package **bnlearn** and package **deal**. In particular, parents of a variable can be enclosed in:

- `( )` if a partial (conditional) independence is present.
- `{ }` if a context specific independence is present.
- `< >` if no context specific and partial (conditional) independences are present, but at least a local independence is detected.

If a parent is not enclosed in parenthesis the dependence is full.

If `only_parents = TRUE`, the simple DAG encoding as in **bnlearn** is returned.

Examples

```
model <- stages_hclust(full(Titanic), k = 2)
pl <- as_parentslist(model)
pl
as.character(pl)
as.character(pl, only_parents = TRUE)
```

`Asym` *Asym dataset*

Description

Artificial dataset with observations from four variables having a non-symmetrical conditional independence structure.

Usage

```
Asym
```

Format

A data frame with 1000 observations of 4 binary variables.

Source

The data has been generated by Federico Carli `<carli@dima.unige>`.

`as_adj_matrix` *Convert to an adjacency matrix*

Description

Convert to an adjacency matrix

Usage

```
as_adj_matrix(x, ...)

## S3 method for class 'parentslist'
as_adj_matrix(x, ...)

## S3 method for class 'ceg'
as_adj_matrix(x, ignore = x$name_unobserved, endnode = TRUE, ...)
```

Arguments

`x`	an R object
`...`	additional parameters
`ignore`	list of stages to be ignored.
`endnode`	logical value. If `TRUE` a final node fil be added.

Value

the equivalent adjacency matrix

for `as_adj_matrix.ceg`: the adj matrix corresponding to the CEG.

`as_bn` *Convert to a* **bnlearn** *object*

Description

Convert a staged tree object into an object of class bn from the **bnlearn** package.

Usage

```
as_bn(x)

## S3 method for class 'parentslist'
as_bn(x)

## S3 method for class 'sevt'
as_bn(x)
```

Arguments

`x`	an R object of class `sevt` or `parentslist`.

Value

an object of class `bn` from package **bnlearn**.

`as_parentslist` *Obtain the equivalent DAG as list of parents*

Description

Convert to the equivalent representation as list of parents.

Usage

```
as_parentslist(x, ...)

## S3 method for class 'bn'
as_parentslist(x, order = NULL, ...)

## S3 method for class 'bn.fit'
as_parentslist(x, order = NULL, ...)

## S3 method for class 'sevt'
as_parentslist(x, silent = FALSE, ...)
```

Arguments

`x`	an R object.
`...`	additional parameters.
`order`	order of the variables, usually a topological order.
`silent`	if function should be silent.

Details

The output of this function is an object of class `parentslist` which is one of the possible encoding for a directed graph. This is mainly an internal class and its specification can be changed in the future. For example, now it may also include information on the sample space of the variables and the context/partial/local independences.

In `as_parentslist.sevt`, if a context-specific or a local-partial independence is detected a message is printed (if `silent = FALSE`) and the minimal super-model is returned.

Value

An object of class `parentslist` for which a print method exists. Basically a list with one entries for each variable with fields:

- `parents` The parents of the variable.
- `context` Where context independences are detected.
- `partial` Where partial independences are detected.
- `local` Where no context/partial independences are detected, but local independences are present.
- `values` values for the variable.

See Also

`print.parentslist` and `as.character.parentslist` for the parenthesis-encoding of the DAG structure and the asymmetric independences.

Examples

```
model <- stages_hclust(full(Titanic), k = 2)
pl <- as_parentslist(model)
pl$Age
```

`as_sevt` *Coerce to sevt*

Description

Convert to an equivalent object of class `sevt`.

Usage

```
as_sevt(x, ...)

## S3 method for class 'bn.fit'
as_sevt(x, order = NULL, ...)

## S3 method for class 'bn'
as_sevt(x, order = NULL, values = NULL, ...)

## S3 method for class 'parentslist'
as_sevt(x, order = NULL, values = NULL, ...)
```

Arguments

`x`	an R object.
`...`	additional parameters to be used by specific methods.
`order`	order of the variables.
`values`	the values for each variable, the sample space.

Details

In `as_sevt.bn.fit` the `order` argument, if provided, must be a topological order of the `bn.fit` object (no check is performed). If the order is not provided a topological order will be used (the one returned by `bnlearn::node.ordering`).

In `as_sevt.parentslist` the `order` argument, if provided, must be a topological order of the corresponding DAG (no check is performed). If the order is not provided `names(x)` is used.

The `values` parameter is used to specify the sample space of each variable. For a `parentslist` object created with `as_parentslist` from an object of class `sevt`, it is, usually, not needed to specify the `values` parameter, since the sample space is saved in the `parentslist` object.

Value

the equivalent object of class `sevt`.

Examples

```
model <- stages_hclust(full(Titanic), k = 2)
plot(model)
pl <- as_parentslist(model)
model2 <- as_sevt(pl)
plot(model2) ## this is a super-model of the first staged tree
## we can check it with
inclusions_stages(model, model2)
```

`barplot.sevt` *Bar plots of stage probabilities*

Description

Create a bar plot visualizing probabilities associated to the different stages of a variable in a staged event tree.

Usage

```
## S3 method for class 'sevt'
barplot(
  height,
  var,
  ignore = height$name_unobserved,
  beside = TRUE,
  horiz = FALSE,
  legend.text = FALSE,
  col = NULL,
  xlab = ifelse(horiz, "probability", NA),
  ylab = ifelse(!horiz, "probability", NA),
  ...
)
```

Arguments

`height`	an object of class `sevt`.
`var`	name of a variable in `object`.
`ignore`	vector of stages which will be ignored and left untouched, by default the name of the unobserved stages stored in `object$name_unobserved`.
`beside`	a logical value. See `barplot`.
`horiz`	a logical value. See `barplot`.
`legend.text`	logical.
`col`	color mapping for the stages, see `col` argument in `plot.sevt`.
`xlab`	a label for the x axis.
`ylab`	a label for the y axis.
`...`	additional arguments passed to `barplot`.

Value

As `barplot`: A numeric vector (or matrix, when beside = TRUE), giving the coordinates of all the bar midpoints drawn, useful for adding to the graph.

Examples

```
model <- stages_fbhc(full(PhDArticles, lambda = 1))
barplot(model, "Kids", beside = TRUE)
```

`ceg` *Chain event graph (CEG)*

Description

Build the CEG representation from an object of class `sevt`.

Usage

```
ceg(object)
```

Arguments

`object`	an object of class `sevt`.

Details

An object of class ceg is a staged event tree object with additional information on the positions.

Value

an object of class ceg.

Examples

```
DD <- generate_xor_dataset(3, 100)
model <- stages_bhc(full(DD))
model.ceg <- ceg(model)
model.ceg$positions
```

`cid` *Context specific interventional discrepancy*

Description

Compute the context specific interventional discrepeancy of a staged tree with respect to a reference staged tree.

Usage

```
cid(object1, object2, FUN = mean)
```

Arguments

`object1`	an object of class `sevt`.
`object2`	an object of class `sevt`.
`FUN`	a function that is used to aggregate CID for each variable. The default `mean` will obtain the CID as defined in Leonelli and Varando (2023).

Value

A list with components:

- `wrong` a stages-like structure which record where `object2` wrongly infer the interventional distance with respect to `object1`.
- `cid` the value of the computed CID.

References

Leonelli M., Varando G. *Context-Specific Causal Discovery for Categorical Data Using Staged Trees*, The 26th International Conference on Artificial Intelligence and Statistics (AISTATS), 2023, https://arxiv.org/abs/2106.04416

Examples

```
model1 <- stages_bhc(full(Titanic))
model2 <- stages_bhc(full(Titanic,
  order = c("Survived", "Sex", "Age", "Class")
))
cid(model1, model2)$cid
cid(model1, model2)$wrong
```

`ci_matrices` *Conditional independences matrices of stages*

Description

Generate the sequence of all the conditional independences matrices of stages for a given variable in the model.

Usage

```
ci_matrices(object, var)
```

Arguments

`object`	an object of class `sevt`.
`var`	string, the name of one of the variables in `object`.

Value

A list with `i-1` matrices, where `i` is the depth of variable `var` in the tree.

Examples

```
mod <- sevt(list(A = c("a", "aa"),
                 B = c("b", "bb", "bbb"),
                 C = c("c", "cc")), full = TRUE)
stages(mod)["C", A = "a", B = c("b", "bb")] <- "stage1"
stages(mod)["C", A = "aa"] <- "stage2"
stages(mod)["C", A = "a", B = "bbb"] <- "stage2"

ci_matrices(mod, "C")
```

`compare_stages` *Compare two staged event tree*

Description

Compare two staged event trees, return the differences of the stages structure and plot the difference tree. Three different methods to compute the difference tree are available (see Details).

Usage

```
compare_stages(
  object1,
  object2,
  method = "naive",
  return_tree = FALSE,
  plot = FALSE,
  ...
)

hamming_stages(object1, object2, return_tree = FALSE)

diff_stages(object1, object2)
```

Arguments

`object1`	an object of class `sevt`.
`object2`	an object of class `sevt`.
`method`	character, method to compare staged event trees. One of: `"naive"`, `"hamming"` or `"stages"`.
`return_tree`	logical, if `TRUE` the difference tree is returned.
`plot`	logical.
`...`	additional parameters to be passed to `plot.sevt`.

Details

`compare_stages` tests if the stage structure of two `sevt` objects is the same. Three methods are available:

- `naive` first applies `stndnaming` to both objects and then simply compares the resulting stage names.
- `hamming` uses the `hamming_stages` function that finds a minimal subset of nodes which stages must be changed to obtain the same structure.
- `stages` uses the `diff_stages` function that compares stages to check whether the same stage structure is present in both models.

Setting `return_tree = TRUE` will return the stages difference obtained with the selected method. The stages difference is a list of numerical vectors with same lengths and structure as `stages(object1)` or `stages(object2)`, where values are 1 if the corresponding node has different (with respect to the selected `method`) associated stage, and 0 otherwise.

With `plot = TRUE` the plot of the difference tree is displayed.

If `return_tree = FALSE` and `plot = FALSE` the logical output is the same for the three methods and thus the `naive` method should be used since it is computationally faster.

`hamming_stages` finds a minimal set of nodes for which the associated stages should be changed to obtain equivalent structures. To do that, a maximum-weight bipartite matching problem between the stages of the two staged trees is solved using the Hungarian method implemented in the `solve_LSAP` function of the **clue** package. `hamming_stages` requires the package `clue`.

Value

`compare_stages`: if `return_tree = FALSE`, logical: `TRUE` if the two models are exactly equal, otherwise `FALSE`. Else if `return_tree = TRUE`, the differences between the two trees, according to the selected `method`.

`hamming_stages`: if `return_tree = FALSE`, integer, the minimum number of situations where the stage should be changed to obtain the same models. If `return_tree = TRUE` a stages-like structure showing which situations should be modified to obtain the same models.

`diff_stages`: a stages-like structure marking the situations belonging to stages which are not the exactly equal.

Examples

```
data("Asym")
mod1 <- stages_bhc(full(Asym, lambda = 1))
mod2 <- stages_fbhc(full(Asym, lambda = 1))
compare_stages(mod1, mod2)

##########
m0 <- full(PhDArticles[, 1:4], lambda = 0)
m1 <- stages_bhc(m0)
m2 <- stages_bj(m0, distance = "totvar", thr = 0.25)
diff_stages(m1, m2)
```

`confint.sevt` *Confidence intervals for staged event tree parameters*

Description

Confint method for class `sevt`.

Usage

```
## S3 method for class 'sevt'
confint(
  object,
  parm,
  level = 0.95,
  method = c("wald", "waldcc", "wilson", "goodman", "quesenberry-hurst"),
  ignore = object$name_unobserved,
  ...
)
```

Arguments

`object`	an object of class `sevt`.

`parm`	a specification of which parameters are to be given confidence intervals, either a vector of numbers or a vector of names. If missing, all parameters are considered.
`level`	the confidence level required.
`method`	a character string specifing which method to use: wald", "waldcc", "goodman", "quesenberry-hurst" or "wilson".
`ignore`	vector of stages which will be ignored, by default the name of the unobserved stages stored in `object$name_unobserved`.
`...`	additional argument(s) for compatibility with `confint` methods.

Details

Compute confidence intervals for staged event trees. Currently five methods are available:

- `wald`, `waldcc`: Wald method and with continuity correction.
- `wilson`, `quesenberry-hurst` and `goodman`.

Value

A matrix with columns giving lower and upper confidence limits for each parameter. These will be labelled as `(1-level)/2` and `1 - (1-level)/2` in % (by default 2.5% and 97.5%).

Author(s)

The function is partially inspired by code in the `MultinomCI` function from the **DescTools** package, implemented by Andri Signorelli and Pablo J. Villacorta Iglesias.

References

Goodman, L. A. (1965) On Simultaneous Confidence Intervals for Multinomial Proportions Technometrics, 7, 247-254.

Wald, A. Tests of statistical hypotheses concerning several parameters when the number of observations is large, Trans. Am. Math. Soc. 54 (1943) 426-482.

Wilson, E. B. Probable inference, the law of succession and statistical inference, J.Am. Stat. Assoc. 22 (1927) 209-212.

Quesenberry, C., & Hurst, D. (1964). Large Sample Simultaneous Confidence Intervals for Multinomial Proportions. Technometrics, 6(2), 191-195

Examples

```
m1 <- stages_bj(full(PhDArticles), distance = "kullback", thr = 0.01)
confint(m1, "Prestige", level = 0.90)
confint(m1, "Married", method = "goodman")
confint(m1, c("Married", "Kids"))
```

`covid_patients` *Trajectories of hospitalized SARS-CoV-2 patients*

Description

Dataset with observations from four variables (Sex, Age, ICU, death) for 10000 simulated SARS-CoV-2 hospital patients.

Usage

```
covid_patients
```

Format

A data frame with 10000 observations of 4 variables. The variables and their levels are as follows:

- Sex: Female, Male
- Age: 0-39, 40-49, 50-59, 60-69, 70-79, 80+
- ICU: yes, no
- death: yes, no

Details

The data are simulated from an event tree where conditional probabilities for ICU and death are taken from the results of Lefrancq et al. (2021). Lefrancq et al. (2021) estimated such probabilities from data on patients, recorded in the SI-VIC database, who started their hospitalization between 13 March and 30 November 2020.

Source

The data has been generated with the code in the Examples section. Conditional probabilities were copied from the tables in the Supplementary materials of Lefrancq et al. (2021). Marginal probabilities of gender and probabilities of age given gender were instead obtained from the linked GitHub repository `https://github.com/noemielefrancq/Evolution-Outcomes-COVID19-France`.

References

Leonelli, M. and Varando, G. (2023). Context-Specific Causal Discovery for Categorical Data Using Staged Trees. *Proceedings of The 26th International Conference on Artificial Intelligence and Statistics*, in *Proceedings of Machine Learning Research* 206:8871-8888 Available from https://proceedings.mlr.press/v206/le

Lefrancq N., Paireau J., Hozé N., Courtejoie N., Yazdanpanah Y., Bouadma L. (2021). Evolution of outcomes for patients hospitalised during the first 9 months of the SARS-CoV-2 pandemic in France: A retrospective national surveillance data analysis. *The Lancet Regional Health - Europe*, 5:100087.

Examples

```
library(stagedtrees)

data_model <- sevt(list(
  Sex = c("Female", "Male"),
  Age = c(
    "0-39", "40-49", "50-59", "60-69",
    "70-79", "80+"
  ),
  ICU = c("yes", "no"),
  death = c("yes", "no")
), full = TRUE)
data_model$prob <- list()
data_model$prob$Sex <- list("1" = c(Female = 0.45185, Male = 0.54815))

dist_age_male <- c(
  0.01616346, # 0 - 39
  0.04159445, # 40 - 49
  0.10130439, # 50 - 59
  0.16825686, # 60 - 69
  0.25217550, # 70 - 79
  0.42050534
) # 80+

dist_age_female <- c(
  0.01688613, # 0 - 39
  0.04271329, # 40 - 49
  0.10131681, # 50 - 59
  0.16841872, # 60 - 69
  0.25289366, # 70 - 79
  0.41777138
) # 80+

names(dist_age_male) <- data_model$tree$Age
names(dist_age_female) <- data_model$tree$Age
data_model$prob$Age <- list(
  "1" = dist_age_female,
  "2" = dist_age_male
)
data_model$prob$ICU <- list(
  "1" = c(yes = 0.125, no = 1 - 0.125), # Female 0-39
  "2" = c(yes = 0.149, no = 1 - 0.149), # Female 40-49
  "3" = c(yes = 0.193, no = 1 - 0.193), # Female 50-59
  "4" = c(yes = 0.225, no = 1 - 0.225), # Female 60-69
  "5" = c(yes = 0.175, no = 1 - 0.175), # Female 70-79
  "6" = c(yes = 0.037, no = 1 - 0.037), # Female 80+
  "7" = c(yes = 0.197, no = 1 - 0.197), # Male 0-39
  "8" = c(yes = 0.2687, no = 1 - 0.2687), # Male 40-49
  "9" = c(yes = 0.3171, no = 1 - 0.3171), # Male 50-59
  "10" = c(yes = 0.3415, no = 1 - 0.3415), # Male 60-69
  "11" = c(yes = 0.274, no = 1 - 0.274), # Male 70-79
  "12" = c(yes = 0.073, no = 1 - 0.073) # Male 80+
```

```
)
data_model$prob$death <- list(
  ################### FEMALE #################################
  "1"  = c(yes = 0.077, no = 1 - 0.077), # Female 0-39 ICU
  "2"  = c(yes = 0.004, no = 1 - 0.004), # Female 0-39 no-ICU
  "3"  = c(yes = 0.117, no = 1 - 0.117), # Female 40-49 ICU
  "4"  = c(yes = 0.017, no = 1 - 0.017), # Female 40-49 no-ICU
  "5"  = c(yes = 0.185, no = 1 - 0.185), # Female 50-59 ICU
  "6"  = c(yes = 0.030, no = 1 - 0.030), # Female 50-59 no-ICU
  "7"  = c(yes = 0.239, no = 1 - 0.239), # Female 60-69 ICU
  "8"  = c(yes = 0.058, no = 1 - 0.058), # Female 60-69 no-ICU
  "9"  = c(yes = 0.324, no = 1 - 0.324), # Female 70-79 ICU
  "10" = c(yes = 0.124, no = 1 - 0.124), # Female 70-79 no-ICU
  "11" = c(yes = 0.454, no = 1 - 0.454), # Female 80+ ICU
  "12" = c(yes = 0.266, no = 1 - 0.266), # Female 80+ no-ICU
  ################# MALE #################################
  "13" = c(yes = 0.079, no = 1 - 0.079), # Male 0-39 ICU
  "14" = c(yes = 0.008, no = 1 - 0.008), # Male 0-39 no-ICU
  "15" = c(yes = 0.098, no = 1 - 0.098), # Male 40-49 ICU
  "16" = c(yes = 0.016, no = 1 - 0.016), # Male 40-49 no-ICU
  "17" = c(yes = 0.171, no = 1 - 0.171), # Male 50-59 ICU
  "18" = c(yes = 0.030, no = 1 - 0.030), # Male 50-59 no-ICU
  "19" = c(yes = 0.278, no = 1 - 0.278), # Male 60-69 ICU
  "20" = c(yes = 0.067, no = 1 - 0.067), # Male 60-69 no-ICU
  "21" = c(yes = 0.383, no = 1 - 0.383), # Male 70-79 ICU
  "22" = c(yes = 0.150, no = 1 - 0.150), # Male 70-79 no-ICU
  "23" = c(yes = 0.478, no = 1 - 0.478), # Male 80+ ICU
  "24" = c(yes = 0.363, no = 1 - 0.363) # Male 80+ no-ICU
)

# covid_patients <- sample_from(data_model, 10000, seed = 123)
# usethis::use_data(covid_patients, overwrite = TRUE)
```

`depsubtree` *Extract dependency subtree*

Description

Extract the dependency subtree of a staged tree with respect to a variable

Usage

```
depsubtree(object, var, other_stages = c("NA", "indep", "full"))
```

Arguments

`object`	an object of class `sevt`.
`var`	the name of one of the variable of the staged event tree.
`other_stages`	how to set stages for other variables (if any).

Details

The dependency sub-tree is a staged event tree which is sufficient to describe the conditional distribution of the variable `var` given its predecessors in the original tree represented by `object`. In particular the preceding variables are restricted to the parents of `var` in the minimal-DAG obtained with `as_parentslist`. This is the minimal set of variables which contexts are sufficient to fully represent the conditional distribution of `var`. Stages for variables different from `var` are either set to NA, or to the full or indep model, depending on `other_stages`.

Value

an object of class `sevt` representing the dependency sub-tree.

Examples

```
mod <- stages_kmeans(full(Titanic), k = 2)
par(mfrow = c(1, 2))
plot(mod, main = "staged tree")
plot(depsubtree(mod, "Age"), main = "dependency subtree for Age")
par(mfrow = c(1, 1))
```

`full_indep` *Full and independent staged event tree*

Description

Build fitted staged event tree from data.

Usage

```
full(
  data,
  order = NULL,
  join_unobserved = TRUE,
  lambda = 0,
  name_unobserved = "UNOBSERVED"
)

## S3 method for class 'table'
full(
  data,
  order = names(dimnames(data)),
  join_unobserved = TRUE,
  lambda = 0,
  name_unobserved = "UNOBSERVED"
)

## S3 method for class 'data.frame'
```

```
full(
  data,
  order = colnames(data),
  join_unobserved = TRUE,
  lambda = 0,
  name_unobserved = "UNOBSERVED"
)

indep(
  data,
  order = NULL,
  join_unobserved = TRUE,
  lambda = 0,
  name_unobserved = "UNOBSERVED"
)

## S3 method for class 'table'
indep(
  data,
  order = names(dimnames(data)),
  join_unobserved = TRUE,
  lambda = 0,
  name_unobserved = "UNOBSERVED"
)

## S3 method for class 'data.frame'
indep(
  data,
  order = colnames(data),
  join_unobserved = TRUE,
  lambda = 0,
  name_unobserved = "UNOBSERVED"
)
```

Arguments

`data`	data to create the model, data.frame or table.
`order`	character vector, order of variables.
`join_unobserved`	logical, if situations with zero observations should be joined (default TRUE).
`lambda`	smoothing coefficient (default 0).
`name_unobserved`	name to pass to `join_unobserved`.

Details

Functions to create full or independent staged tree models from data. The full (or saturated) staged tree is the model where every situation is in a different stage, and thus the model has the maximum number of parameters. Conversely, the independent staged tree (`indep`) assigns all the situations related to the same variable to the same stage, thus it is equivalent to the independence factorization.

Examples

```
## full model
DD <- generate_xor_dataset(4, 100)
model_full <- full(DD, lambda = 1)

## independence model (data.frame)
DD <- generate_xor_dataset(4, 100)
model <- indep(DD, lambda = 1)
model
```

`generate_linear_dataset` *Generate a random binary dataset for classification*

Description

Randomly generate a simple classification problem.

Usage

```
generate_linear_dataset(
  p,
  n,
  eps = 1.2,
  gamma = runif(1, min = -p, max = p),
  alpha = runif(p, min = -p, max = p)
)
```

Arguments

`p`	number of variables.
`n`	number of observations.
`eps`	noise.
`gamma`	numeric.
`alpha`	numeric vector of length `n`.

Value

A data.frame with `n` independent random variables and one class variable `C` computed as `sign(sum(x * alpha) + runif(1, -eps, eps) + gamma)`.

Examples

```
DD <- generate_linear_dataset(p = 5, n = 1000)
```

`generate_random_dataset` *Generate a random binary dataset*

Description

Randomly generate a data.frame of independent binary variables.

Usage

```
generate_random_dataset(p, n)
```

Arguments

`p`	number of variables.
`n`	number of observations.

Value

A data.frame with `n` independent random variables.

Examples

```
DD <- generate_random_dataset(p = 5, n = 1000)
```

`generate_xor_dataset` *Generate a xor dataset*

Description

Generate a xor dataset

Usage

```
generate_xor_dataset(p, n, eps = 1.2)
```

Arguments

`p`	number of variables.
`n`	number of observations.
`eps`	error.

Value

The xor dataset with `n` + 1 variables, where the first one is the class variable `C` computed as a noisy xor.

Examples

```
DD <- generate_xor_dataset(p = 5, n = 1000, eps = 1.2)
```

`get_stage` *Get stage or path*

Description

Utility functions to obtain stages from paths and paths from stages.

Usage

```
get_stage(object, path)

get_path(object, var, stage)
```

Arguments

`object`	an object of class `sevt`.
`path`	character vector, the path from root or a two dimensional array where each row is a path from root.
`var`	character, one of the variable in the staged tree.
`stage`	character vector, the name of the stages for which the paths should be returned.

Value

`get_stage` returns the stage name(s) for given path(s).

`get_path` returns a data.frame containing the paths corresponding to the given stage(s).

Examples

```
model <- stages_fbhc(full(PhDArticles))
get_stage(model, c("0", "male"))
paths <- expand.grid(model$tree[2:1])[, 2:1]
get_stage(model, paths)
get_path(model, "Kids", "5")
get_path(model, "Gender", "2")
get_path(model, "Kids", c("5", "6"))
```

---

igraph-conversion **igraph** *conversion*

Description

Obtain the graph representation of a staged tree or a CEG as an object from the **igraph** package.

Usage

```
get_edges(x, ignore = x$name_unobserved, ...)

## S3 method for class 'sevt'
get_edges(x, ignore = x$name_unobserved, ...)

get_vertices(x, ignore = x$name_unobserved, ...)

## S3 method for class 'sevt'
get_vertices(x, ignore = x$name_unobserved, ...)

## S3 method for class 'ceg'
get_edges(x, ignore = x$name_unobserved, ...)

## S3 method for class 'ceg'
get_vertices(x, ignore = x$name_unobserved, ...)

as_igraph(x, ignore = x$name_unobserved, ...)

## S3 method for class 'sevt'
as_igraph(x, ignore = x$name_unobserved, ...)

## S3 method for class 'ceg'
as_igraph(x, ignore = x$name_unobserved, ...)
```

Arguments

`x`	an object of class `sevt` or `ceg`.
`ignore`	vector of stages which will be ignored and excluded, by default the name of the unobserved stages stored in `x$name_unobserved`.
`...`	additional parameters.

Details

Functions to transalte the graph structure of a `sevt` or `ceg` object to a graph object from the **igraph** package. Additional functions that extract the edge lists and the vertices are available. This can be useful, for example to plot the staged tree with **igraph** or additional packages (see the examples).

Value

for `get_edges`: the edges list corresponding to the graph associated to `x`.

for `get_vertices`: the vertices list corresponding to the graph associated to `x`.

for `as.igraph`: a graph object from the **igraph** package.

Examples

```
mod <- stages_bhc(full(Titanic))
get_edges(mod)
get_vertices(mod)
## Not run:
library(igraph)
library(ggraph)
######## sevt example ########
## convert to igraph object
g <- as_igraph(mod)
## plot with igraph directly
plot(g, layout = layout_with_sugiyama)
## plot with ggraph
ggraph(g, "sugiyama") +
  geom_edge_fan(
    aes(
      label = label,
      label_pos = 0.5 + runif(length(label), -0.1, 0.1)
    ),
    angle_calc = "along", show.legend = FALSE, check_overlap = FALSE,
    end_cap = circle(0.02, "npc"),
    arrow = grid::arrow(
      angle = 25,
      length = unit(0.025, "npc"),
      type = "closed"
    )
  ) +
  geom_node_point(aes(x = x, y = y, color = stage),
    size = 5,
    show.legend = FALSE
  ) +
  ggforce::theme_no_axes() + coord_flip() + scale_y_reverse()

######## ceg example ########
g.ceg <- as_igraph(ceg(mod))
### igraph plotting functions can be used
plot(g.ceg, layout = layout.sugiyama)
### igraph object can be also plotted with ggplot2 and ggraph
ggraph(g.ceg, "sugiyama") +
  geom_edge_fan(
    aes(
      label = label,
      color = label,
      label_pos = 0.5 + runif(length(label), -0.1, 0.1)
    ),
```

```
    angle_calc = "along", show.legend = FALSE, check_overlap = FALSE,
    end_cap = circle(0.02, "npc"),
    arrow = grid::arrow(
      angle = 25,
      length = unit(0.025, "npc"),
      type = "closed"
    )
  ) +
  geom_node_point(aes(x = x, y = y, color = stage), size = 3, show.legend = FALSE) +
  ggforce::theme_no_axes() + coord_flip() + scale_y_reverse()

## End(Not run)
```

`inclusions_stages` *Inclusions of stages*

Description

Display the relationship between two staged tree models over the same variables.

Usage

```
inclusions_stages(object1, object2)
```

Arguments

`object1`	an object of class `sevt`.
`object2`	an object of class `sevt`.

Details

Computes the relations between the stages structures of the two models.

The relations between stages of the same variable are stored in a data frame with three columns where each row represent a relation between a stage of the first model (`s1`) and a stage of the second model (`s2`). The relation can be one of the following: inclusion (`s1` < `s2` or `s1` > `s2`; equal (`s1` = `s2`); not-equal (`s1` != `s2`).

Value

a list with inclusion relations between stage structures for each variable in the models.

Examples

```
mod1 <- stages_bhc(full(PhDArticles[, 1:5], lambda = 1))
mod2 <- stages_fbhc(full(PhDArticles[, 1:5], lambda = 1))
inclusions_stages(mod1, mod2)
```

`join_positions` *join positions in a staged tree model*

Description

join positions in a staged tree model

Usage

```
join_positions(model, var, s1, s2)
```

Arguments

`model`	an object of class `sevt`.
`var`	the name of a variable in the model.
`s1`	stage to join
`s2`	stage to join

Details

this functions works similarly to the `join_stages` function in the `stagedtrees` package, but it also joins downstream stages to make nodes with stages `s1,s2` in the same position. This function works properly only when downstream variables from `var` have full stages vectors.

`join_unobserved` *Join situations with no observations*

Description

Join situations with no observations

Usage

```
join_unobserved(
  object,
  fit = TRUE,
  trace = 0,
  name = "UNOBSERVED",
  scope = sevt_varnames(object)[-1],
  lambda = object$lambda
)
```

Arguments

`object`	an object of class `sevt` with associated data.
`fit`	if TRUE update model's probabilities.
`trace`	if > `0` print information to console.
`name`	character, name for the new stage storing unobserved situations.
`scope`	character vector, list of variables in `object`.
`lambda`	smoothing parameter for the fitting.

Details

It takes as input a (fitted) staged event tree object and it joins, in the same stage, all the situations with zero recorded observations. Since such joining does not change the log-likelihood of the model, it is a useful (time-wise) pre-processing prior to others model selection algorithms.

Unobserved situations can be joined directly in `full` or `indep`, by setting `join_unobserved = TRUE`.

Value

a staged event tree with at most one stage per variable with no observations. If, as default, `fit=TRUE` the model will be re-fitted, if `fit=FALSE` probabilities in the output model are not estimated.

Examples

```
DD <- generate_xor_dataset(p = 5, n = 10)
model_full <- full(DD, lambda = 1, join_unobserved = FALSE)
model <- join_unobserved(model_full)
logLik(model_full)
logLik(model)
BIC(model_full, model)
```

`logLik.sevt` *Log-Likelihood of a staged event tree*

Description

Compute, or extract the log-likelihood of a staged event tree.

Usage

```
## S3 method for class 'sevt'
logLik(object, ...)
```

Arguments

`object`	an fitted object of class `sevt`.
`...`	additional parameters (compatibility).

Value

An object of class `logLik`.

Examples

```
data("PhDArticles")
mod <- indep(PhDArticles)
logLik(mod)
```

`lr_test` *Likelihood Ratio Test for staged trees models*

Description

Function to perform likelihood ratio test between two or multiple staged event tree models.

Usage

```
lr_test(object, ...)
```

Arguments

`object`	an object of class `sevt`.
`...`	further objects of class `sevt`. Must specify super-models of `object`. See below for details.

Details

If a single object of class `sevt` is passed as argument, it computes the likelihood-ratio test with respect to the independence model. If multiple objects are passed, likelihood-ratio tests between the first object and the followings are computed. In the latter case the function checks automatically if the first model is nested in the additional ones, via `inclusions_stages`, and throws an error if not.

Value

An object of class anova which contains the log-likelihood, degrees of freedom, difference in degrees of freedom, likelihood ratio statistics and corresponding p values.

Examples

```
data(PhDArticles)
order <- c("Gender", "Kids", "Married", "Articles")
phd.mod1 <- stages_hc(indep(PhDArticles, order))
phd.mod2 <- stages_hc(full(PhDArticles, order))

## compare two nested models
lr_test(phd.mod1, phd.mod2)
```

```
## compare a single model vs the independence model
lr_test(phd.mod1)
```

`PhDArticles` *PhD Students Publications*

Description

Number of publications of 915 PhD biochemistry students during the 1950's and 1960's.

Usage

```
PhDArticles
```

Format

A data frame with 915 rows and 6 variables:

Articles Number of articles during the last 3 years of PhD: either `0`, `1-2` or `>2`.

Gender `male` or `female`.

Kids `yes` if the student has at least one kid 5 or younger, `no` otherwise.

Married `yes` or `no`.

Mentor Number of publications of the student's mentor: `low` between 0 and 3, `medium` between 4 and 10, `high` otherwise.

Prestige `low` if the student is at a low-prestige university, `high` otherwise.

Source

The data has been modified from the `Rchoice` package.

References

Long, J. S. (1990). The origins of sex differences in science. *Social Forces*, 68(4), 1297-1316.

`plot.ceg` *igraph's plotting for CEG*

Description

igraph's plotting for CEG

Usage

```
## S3 method for class 'ceg'
plot(x, col = NULL, ignore = x$name_unobserved, layout = NULL, ...)
```

Arguments

`x`	an object of class `ceg`.
`col`	colors specification see `plot.sevt`.
`ignore`	vector of stages which will be ignored and left untouched, by default the name of the unobserved stages stored in `x$name_unobserved`.
`layout`	an igraph layout.
`...`	additional arguments passed to `plot.igraph`.

Details

This function is a simple wrapper around **igraph**'s `plot.igraph`. The ceg object is converted to an igraph object with `as_igraph`. If not specified, the default `layout` used is a rotated `layout.sugiyama`.

We use `palette()` as palette for the **igraph** plotting, while `plot.igraph` uses as default a different palette. This is to allow matching stages colors between `plot.ceg` and `plot.sevt`.

Examples

```
## Not run:
 model <- stages_bhc(full(Titanic))
 model.ceg <- ceg(model)
 plot(model.ceg, edge.arrow.size = 0.1, vertex.label.dist = -2)

## End(Not run)
```

`plot.sevt` *Plot method for staged event trees*

Description

Plot method for staged event tree objects. It allows easy plotting of staged event trees with some options (see Examples).

Usage

```
## S3 method for class 'sevt'
plot(
  x,
  y = 10,
  limit = y,
  xlim = c(0, 1),
  ylim = c(0, 1),
  main = NULL,
  sub = NULL,
  asp = 1,
  cex_label_nodes = 0,
  cex_label_edges = 1,
  cex_nodes = 2,
  cex_tree_y = 0.9,
  col = NULL,
  col_edges = "black",
  var_names = TRUE,
  ignore = x$name_unobserved,
  pch_nodes = 16,
  lwd_nodes = 1,
  lwd_edges = 1,
  ...
)

make_stages_col(x, col = NULL, ignore = x$name_unobserved, limit = NULL)
```

Arguments

`x`	an object of class `sevt`.
`y`	alias for `limit` for compatibility with `plot`.
`limit`	maximum number of variables plotted.
`xlim`	the x limits (x1, x2) of the plot.
`ylim`	the y limits of the plot.
`main`	an overall title for the plot.
`sub`	a sub title for the plot.

`asp`	the y/x aspect ratio.
`cex_label_nodes`	the magnification to be used for the node labels. If set to `0` (as default) node labels are not showed.
`cex_label_edges`	the magnification for the edge labels. If set to `0` edge labels are not displayed.
`cex_nodes`	the magnification for the nodes of the tree.
`cex_tree_y`	the magnification for the tree in the vertical direction. Default is `0.9` to leave some space for the variable names.
`col`	color mapping for stages, one of the following: NULL (color will be assigned based on the current palette); a named (variables) list of named (stages) vectors of colors; the character `"stages"`, in which case the stage names will be used as colors; a function that takes as input a vector of stages and output the corresponding colors. Check the provided examples. The function `make_stages_col` is used internally and `make_stages_col(x, NULL)` or `make_stages_col(x, "stages")` can be used as a starting point for colors tweaking.
`col_edges`	color for the edges.
`var_names`	logical, if variable names should be added to the plot, otherwise variable names can be added manually using `text.sevt`.
`ignore`	vector of stages which will be ignored and left untouched, by default the name of the unobserved stages stored in `x$name_unobserved`.
`pch_nodes`	either an integer specifying a symbol or a single character to be used as the default in plotting nodes shapes see `points`.
`lwd_nodes`	the line width for edges, a positive number, defaulting to 1.
`lwd_edges`	the line width for nodes, a positive number, defaulting to 1.
`...`	additional graphical parameters to be passed to `points`, `lines`, `title`, `text` and `plot.window`.

Examples

```
data("PhDArticles")
mod <- stages_bj(full(PhDArticles, join_unobserved = TRUE))

### simple plotting
plot(mod)

### labels in nodes
plot(mod, cex_label_nodes = 1, cex_nodes = 0)

### reduce nodes size
plot(mod, cex_nodes = 0.5)

### change line width and nodes style
plot(mod, lwd_edges = 3, pch_nodes = 5)

### changing palette
plot(mod, col = function(s) heat.colors(length(s)))
```

```
### or changing global palette
palette(hcl.colors(10, "Harmonic"))
plot(mod)
palette("default") ##

### forcing plotting of unobserved stages
plot(mod, ignore = NULL)

### use function to specify colors
plot(mod, col = function(stages) {
  hcl.colors(n = length(stages))
})

### manually give stages colors
### as an example we will assign colors only to the stages of two variables
### Gender (one stage named "1") and Mentor (six stages)
col <- list(
  Gender = c("1" = "blue"),
  Mentor = c(
    "UNOBSERVED" = "grey",
    "2" = "red",
    "3" = "purple",
    "10" = "pink",
    "18" = "green",
    "22" = "brown"
  )
)
### by setting ignore = NULL we will plot also the UNOBSERVED stage for Mentor
plot(mod, col = col, ignore = NULL)
```

`Pokemon` *Pokemon Go Users*

Description

Demographic information of a population of possible Pokemon Go users.

Usage

```
Pokemon
```

Format

A data frame with 999 rows and 5 variables:

Use `Y` if the individual used the app, `N` otherwise

Age `>30` if the individual is older than 30, `<=30` otherwise

Degree `Yes` if the individual completed a Higher Education degree, `No` otherwise

Gender `Male` or `Female`

Activity `Yes` if the individual was physically active (i.e. had a walk longer than 30 mins, went for a run or had a bike ride to get some exercise) in the past week before the experiment, `No` otherwise

Source

https://osf.io/xy5g6/

References

Gabbiadini, Alessandro, Christina Sagioglou, and Tobias Greitemeyer. "Does Pokémon Go lead to a more physically active life style?." *Computers in Human Behavior* 84 (2018): 258-263.

`predict.sevt` *Predict method for staged event tree*

Description

Predict class values from a staged event tree model.

Usage

```
## S3 method for class 'sevt'
predict(object, newdata = NULL, class = NULL, prob = FALSE, log = FALSE, ...)
```

Arguments

`object`	an object of class `sevt` with fitted probabilities.
`newdata`	the newdata to perform predictions
`class`	character, the name of the variable to use as the class variable, if NULL the first element `names(object$tree)` will be used.
`prob`	logical, if `TRUE` the probabilities of class are returned
`log`	logical, if `TRUE` log-probabilities are returned
`...`	additional parameters, see details

Details

Predict the most probable a posterior value for the class variable given all the other variables in the model. Ties are broken at random and if, for a given vector of predictor variables, all conditional probabilities are 0, NA is returned.

if `prob = TRUE`, a matrix with number of rows equals to the number of rows in the `newdata` and number of columns as the number of levels of the `class` variable is returned. if `log = TRUE`, log-probabilities are returned.

if `prob = FALSE`, a vector of length as the number of rows in the `newdata` with the level with higher estimated probability for each new observations is returned.

Value

A vector of predictions or the corresponding matrix of probabilities.

Examples

```
DD <- generate_xor_dataset(p = 4, n = 600)
order <- c("C", "X1", "X2", "X3", "X4")
train <- DD[1:500, order]
test <- DD[501:600, order]
model <- full(train)
model <- stages_bhc(model)
pr <- predict(model, newdata = test, class = "C")
table(pr, test$C)
# class values:
predict(model, newdata = test, class = "C")
# probabilities:
predict(model, newdata = test, class = "C", prob = TRUE)
# log-probabilities:
predict(model, newdata = test, class = "C", prob = TRUE, log = TRUE)
```

`print.sevt` *Print a staged event tree*

Description

Print a staged event tree

Usage

```
## S3 method for class 'sevt'
print(x, ..., max = 5)
```

Arguments

`x`	an object of class `sevt`.
`...`	additional parameters (compatibility).
`max`	integer, limit on the numebr of variables to print.

Details

The order of the variables in the staged tree is printed (from root). In addition the number of levels of each variable is shown in square brackets. If available the log-likelihood of the model is printed.

Value

An invisible copy of `x`.

Examples

```
DD <- generate_xor_dataset(5, 100)
model <- full(DD, lambda = 1)
print(model)
```

`prob` *Probabilities for a staged event tree*

Description

Compute (marginal and/or conditional) probabilities of elementary events with respect to the probability encoded in a staged event tree.

Usage

```
prob(object, x, conditional_on = NULL, log = FALSE, na0 = TRUE)
```

Arguments

`object`	an object of class `sevt` with probabilities.
`x`	the vector or data.frame of observations.
`conditional_on`	named vector, the conditioning event.
`log`	logical, if `TRUE` log-probabilities are returned.
`na0`	logical, if `NA` should be converted to 0.

Details

Computes probabilities related to a vector or a data.frame of observations.

Optionally, conditional probabilities can be obtained by specifying the conditioning event in `conditional_on`. This can be done either with a single named vector or with a data.frame object with the same number of rows of `x`. In the former, the same conditioning is used for all the computed probabilities (if `x` has multiple rows); while with the latter different conditioning events (but on the same variables) can be specified for each row of `x`.

Value

the probabilities to observe each observation in `x`, possibly conditional on the event(s) in `conditional_on`.

Examples

```
data(Titanic)
model <- full(Titanic, lambda = 1)
samples <- expand.grid(model$tree[c(1, 4)])
pr <- prob(model, samples)
## probabilities sum up to one
sum(pr)
## print observations with probabilities
```

```
print(cbind(samples, probability = pr))

## compute one probability
prob(model, c(Class = "1st", Survived = "Yes"))

## compute conditional probability
prob(model, c(Survived = "Yes"), conditional_on = c(Class = "1st"))

## compute conditional probabilities with different conditioning set
prob(model, data.frame(Age = rep("Adult", 8)),
  conditional_on = expand.grid(model$tree[2:1])
)
## the above should be the same as
summary(model)$stages.info$Age
```

`random_parentslist` *Generate a random* `parentslist` *object (DAG)*

Description

generate a random DAG coded as `parentslist` object.

Usage

```
random_parentslist(n, k = 2, maxp = n)
```

Arguments

`n`	number of variables.
`k`	maximum number of levels for each variable.
`maxp`	maximum cardinality of parents sets.

Details

For each variable a subset of random cardinality (maximum `maxp`) of the preceding variables is randomly selected as parents set. The possible levels of each variables are randomly selected in `2,...,k`.

Value

a `parentslist` object.

Examples

```
random_parentslist(5, 3, 2)

## we can generate the associated staged tree
pl <- random_parentslist(4, 2, 2)
plot(as_sevt(pl), main = as.character(pl))
```

`random_sevt` *Generate a random (fitted) sevt*

Description

Generate a random `sevt` from a DAG or a tree. Probabilities are also randomly generated.

Usage

```
random_sevt(x, q = 0.5, rfun = rexp)

## S3 method for class 'list'
random_sevt(x, q = 0.5, rfun = rexp)

## S3 method for class 'parentslist'
random_sevt(x, q = 0.5, rfun = rexp)

## S3 method for class 'sevt'
random_sevt(x, q = 0.5, rfun = rexp)
```

Arguments

`x`	a `sevt` object, a `parentslist` object or a `list`.
`q`	probability of joining stages.
`rfun`	a function which is used to generate random conditional probabilities associated to each stage.

Details

The generated staged tree is obtained by randomly joining stages with probability `q`.

For `random_sevt.list`, `x` should be a list representing an event tree, same format as lists provided to `sevt.list`. The random generated `sevt` will be obtained by randomly joining stages starting from a full staged event tree.

For `random_sevt.parentslist`, `x` should be a `parentslist` object representing a DAG, this could be obtained with `as_parentslist` or with `random_parentslist`. The random generated `sevt` will be obtained by randomly joining stages starting from a the staged tree equivalent to the DAG.

For `random_sevt.sevt`, `x` should be a `sevt`. The random generated `sevt` will be obtained by randomly joining stages starting from the provided sevt object.

```
Stages (conditional) probabilities are sampled from
the corresponding probability simplex by generating
a vector with the user-defined function \code{rfun} and
normalizing it to sum up to one.
Absolute value is applied to assure non-negativity.
The default \code{rfun = rexp} induces a uniform sampling
from the probability simplex.
```

Value

A randomly generated fitted `sevt` object.

Examples

```
model_gt <- random_sevt(list(
  X = c("a", "b"), Y = c("c", "d", "e"),
  Z = c("1", "2", "3"), W = c("yes", "no")
))

## sample data from model_gt and estimate a staged tree
data <- sample_from(model_gt, 100)
model_est <- stages_bhc(full(data))

## compare true and estimated model
hamming_stages(model_gt, model_est)
compare_stages(model_gt, model_est, method = "hamming", plot = TRUE)
```

`rename_stage` *Rename stage(s) in staged event tree*

Description

Change the name of a stage in a staged event tree.

Usage

```
rename_stage(object, var, stage, new)
```

Arguments

`object`	an object of class `sevt`.
`var`	name of a variable in `object`.
`stage`	name of the stage to be renamed.
`new`	new name for the stage.

Details

No internal checks are performed and as side effect stages can be joined, if e.g. `new` is equal to the name of a stage for variable `var`.

Value

a staged event tree object where stages `stage` have been renamed to `new`.

`sample_from` *Sample from a staged event tree*

Description

Generate a random sample from the distribution encoded in a staged event tree object.

Usage

```
sample_from(object, size = 1, seed = NULL)
```

Arguments

`object`	an object of class `sevt` with fitted probabilities.
`size`	number of observations to sample.
`seed`	an object specifying if and how the random number generator should be initialized (‘seeded’). Either NULL or an integer that will be used in a call to set.seed.

Details

It samples `size` observations according to the transition probabilities (`object$prob`) in the model.

Value

A data frame containing `size` observations from the variables in `object`.

Examples

```
model <- stages_fbhc(full(PhDArticles, lambda = 1))
sample_from(model, 10)
```

`search_best` *Optimal Order Search*

Description

Find the optimal staged event tree with a dynamic programming approach.

Usage

```
search_best(
  data,
  alg = stages_bhc,
  search_criterion = BIC,
  lambda = 0,
  join_unobserved = TRUE,
  ...
)
```

Arguments

`data`	either a data.frame or a table containing the data.
`alg`	a function that performs stages structure estimation. Similar to `stages_bhc` or `stages_hclust`. The function `alg` must accept the argument `scope`.
`search_criterion`	the criterion minimized in the order search.
`lambda`	numerical value passed to `full`.
`join_unobserved`	logical, passed to `full`.
`...`	additional arguments, passed to `alg`.

Details

This function is an implementation of the dynamic programming approach of Silander and Leong (2013). If the `search_criterion` is decomposable the returned model attains the best value among all possible orders.

Value

The estimated staged event tree model.

References

Silander T., Leong TY. A Dynamic Programming Algorithm for Learning Chain Event Graphs. In: Fürnkranz J., Hüllermeier E., Higuchi T. (eds) Discovery Science. DS 2013. *Lecture Notes in Computer Science*, vol 8140. Springer, Berlin, Heidelberg. 2013.

Cowell R and Smith J. Causal discovery through MAP selection of stratified chain event graphs. *Electronic Journal of Statistics*, 8(1):965–997, 2014.

Examples

```
## default search using BIC score
model <- search_best(Titanic, alg = stages_kmeans)

## use df as search_criterion
model1 <- search_best(Titanic, alg = stages_bhc,
                      search_criterion = function(m) attr(logLik(m), "df"))
```

`search_greedy` *Greedy Order Search*

Description

Search the optimal staged event tree with a greedy heuristic.

Usage

```
search_greedy(
  data,
  alg = stages_bhc,
  search_criterion = BIC,
  lambda = 0,
  join_unobserved = TRUE,
  ...
)
```

Arguments

`data`	either a data.frame or a table containing the data.
`alg`	a function that performs stages structure estimation. Similar to `stages_bhc` or `stages_hclust`. The function `alg` must accept the argument `scope`.
`search_criterion`	the criterion minimized in the order search.
`lambda`	numerical value passed to `full`.
`join_unobserved`	logical, passed to `full`.
`...`	additional arguments, passed to `alg`.

Details

The greedy approach implemented in this function iteratively adds variables to the staged tree that better improve the `search_criterion`.

Value

The estimated staged event tree model.

Examples

```
model <- search_greedy(Titanic, alg = stages_fbhc)
print(model)
```

`sevt` *Staged event tree (sevt) class*

Description

Structure and usage of S3 class `sevt`, used to store a staged event tree.

Usage

```
sevt(x, full = FALSE, order = NULL)

## S3 method for class 'table'
sevt(x, full = FALSE, order = names(dimnames(x)))

## S3 method for class 'data.frame'
sevt(x, full = FALSE, order = colnames(x))

## S3 method for class 'list'
sevt(x, full = FALSE, order = names(x))
```

Arguments

`x`	a list, a data frame or table object.
`full`	logical, if TRUE the full model is created otherwise the independence model.
`order`	character vector, order of the variables to build the tree, by default the order of the variables in `x`.

Details

A staged event tree object is a list with components:

- tree (required): A named list with one component for each variable in the model, a character vector with the names of the levels for that variable. The order of the variables in `tree` is the order of the event tree.
- stages (required): A named list with one component for each variable but the first, a character vector storing the stages for the situations related to path ending in that variable.
- ctables: A named list with one component for each variable, the flat contingency table of that variable given the previous variables.
- lambda: The smoothing parameter used to compute probabilities.
- name_unobserved: The stage name for unobserved situations.
- prob: The conditional probability tables for every variable and stage. Stored in a named list with one component for each variable, a list with one component for each stage.
- ll: The log-likelihood of the `estimated` model. If present, `logLik.sevt` will return this value instead of computing the log-likelihood.

The tree structure is never defined explicitly, instead it is implicitly defined by the list `tree` containing the order of the variables and the names of their levels. This is sufficient to define a complete symmetric tree where an internal node at a depth related to a variable `v` has a number of children equal to the cardinality of the levels of `v`. The stages information is instead stored as a list of vectors, where each vector is indexed as the internal nodes of the tree at a given depth.

To define a staged tree from data (data frame or table) the user can call either `full` or `indep` which both construct the staged tree object, attach the data in `ctables` and compute probabilities. After, one of the available model selection algorithm can be used, see for example `stages_hc`, `stages_bhc` or `stages_hclust`. If, mainly for development, only the staged tree structure is needed (without data or probabilities) the basic `sevt` constructor can be used.

Value

A staged event tree object, an object of class `sevt`.

Examples

```
######### from table
model.titanic <- sevt(Titanic, full = TRUE)

######### from data frame
DD <- generate_random_dataset(p = 4, n = 1000)
model.indep <- sevt(DD)
model.full <- sevt(DD, full = TRUE)

######### from list
model <- sevt(list(
  X = c("good", "bad"),
  Y = c("high", "low")
))
```

`sevt_add` *Add a variable to a staged event tree*

Description

Return an updated staged event tree with one additional variable at the end of the tree.

Usage

```
sevt_add(object, var, data, join_unobserved = TRUE, useNA = "ifany")
```

Arguments

`object`	an object of class `sevt`.
`var`	character, the name of the new variable to be added.
`data`	either a `data.frame` or a `table` containing the data from the variables in `object` plus `var`.
`join_unobserved`	logical, passed to `full`.
`useNA`	whether to include NA values in the tables. Argument passed to `table`.

Details

This function update a staged event tree object with an additional variable. The stages structure of the new variable is initialized as in the saturated model.

Value

An object of class `sevt` representing a staged event tree model with `var` added as last variable.

Examples

```
model <- full(Titanic, order = c("Age", "Class"))
print(model)
model <- sevt_add(model, "Survived", Titanic)
print(model)
```

`sevt_df` *Number of parameters of a staged event tree*

Description

Return the number of parameters of the model.

Usage

```
sevt_df(x)
```

Arguments

`x`	An object of class `sevt`.

Value

integer, degrees of freedom of the staged event tree.

`sevt_fit` *Fit a staged event tree*

Description

Estimate transition probabilities in a staged event tree from data. Probabilities are estimated with the relative frequencies plus, eventually, an additive (Laplace) smoothing.

Usage

```
sevt_fit(
  object,
  data = NULL,
  lambda = NULL,
  scope = NULL,
  compute_logLik = TRUE
)
```

Arguments

`object`	an object of class `sevt`.
`data`	data.frame or contingency table with observations of the variables in `object`.
`lambda`	smoothing parameter or pseudocount. Default (NULL) to lambda value stored in `object`. If no lambda value is stored nor provided, 0 will be used with a warning.
`scope`	which variable should be fitted. Default (NULL) to all variables in the model. A partial re-fit is possible only for model which are already fitted and in that case the provided `lambda` will be ignored if different from `object$lambda`.
`compute_logLik`	logical value. If `TRUE` the log-likelihood of the model is computed and stored.

Details

The data in form of contingency tables and the log-likelihood of the model is (eventually) stored in the returned staged event tree. Partial re-fit of a model can be performed with the `scope` argument. Partial re-fit can only be done over a fully fitted model, e.g. when changing the stages structure of one of the variables. In case of a partial re-fit, the `data` and `lambda` arguments will be ignored and the data and lambda value stored in the sevt object will be used (a warning is issued if such arguments are supplied).

Value

A fitted staged event tree, that is an object of class `sevt` with `ctables` and `prob` components. Additionally the chosen `lambda` is stored in the returned object and eventually the log-likelihood of the model is saved in the `ll` field.

Examples

```
#########
model <- sevt(list(
  X = c("good", "bad"),
  Y = c("high", "low")
))
D <- data.frame(
  X = c("good", "good", "bad"),
  Y = c("high", "low", "low")
)
model.fit <- sevt_fit(model, data = D, lambda = 1)
```

`sevt_nvar` *Number of variables*

Description

Utility returning the number of variables in a staged event tree model.

Usage

```
sevt_nvar(object)
```

Arguments

`object`	An object of class `sevt`.

Value

integer, the number of variables.

`sevt_simplify` *Simplify a staged tree model*

Description

Function to simplify a staged tree model.

Usage

```
sevt_simplify(object, fit = TRUE)
```

Arguments

`object`	an object of class `sevt`
`fit`	logical, if `TRUE` refit the model after simplification.

Details

The `simplify` function will produce the corresponding simple staged tree, that is a staged tree where stages and positions are equivalent. To do so the function `ceg` is used to compute positions, and then the stages' vectors are replaced with the positions' vectors. The model is the re-fitted if the input was a fitted staged tree. Despite the name, the simplified staged tree has always a number of stages greater or equal to the initial staged tree, thus it is a more complex statistical model.

Value

an object of class `sevt` representing the simplified model. The returned model will be fitted if the input `model` was.

Examples

```
mod <- stages_kmeans(full(Titanic), k = 2)
simpl <- sevt_simplify(mod)
plot(simpl)
```

`sevt_varnames` *Variable names*

Description

Utility returning variable-names in a staged event tree model.

Usage

```
sevt_varnames(object)
```

Arguments

`object`	an object of class `sevt`.

Value

A character vector.

`stagedtrees` *Staged event trees.*

Description

Algorithms to create, learn, fit and explore staged event tree models. Functions to compute probabilities, make predictions from the fitted models and to plot, analyze and manipulate staged event trees.

Details

A staged event tree is a representation of a particular factorization of a joint probability over a product space. In particular, given a vector of categorical random variables $X1, X2, \ldots$, a staged event tree represents the factorization $P(X1, X2, X3, \ldots) = P(X1)P(X2|X1)P(X3|X1, X2)\ldots$. Additionally, the stages structure indicates which conditional probabilities are equal.

Model selection algorithms:

- full model `full`
- independence model `indep`
- Hill-Climbing `stages_hc`
- Backward Hill-Climbing `stages_bhc`
- Fast Backward Hill-Climbing `stages_fbhc`
- Backward Hill-Climbing Random `stages_bhcr`
- Backward joining `stages_bj`
- Simple Backward Hill-Climbing `stages_simplebhc`

- Hierarchical Clustering `stages_hclust`
- K-Means Clustering `stages_kmeans`
- Optimal order search `search_best`
- Greedy order search `search_greedy`

Probabilities, log-likelihood and predictions:

- Marginal/Conditional probabilities `prob`
- Log-Likelihood `logLik.sevt`
- Predict method `predict.sevt`
- Confidence intervals `confint.sevt`

Plot, explore and compare:

- Plot `plot.sevt`
- Compare `compare_stages`
- Stages inclusion `inclusions_stages`
- Stages info `summary.sevt`
- List of parents `as_parentslist`
- Barplot construction `barplot.sevt`
- Likelihood-ratio test `lr_test`
- Context-specific interventional distance `cid`

Modify models:

- Join and isolate unobserved situations `join_unobserved`
- Join two stages `join_stages`
- Join two positions `join_positions`
- Rename a stage `rename_stage`

Author(s)

Maintainer: Gherardo Varando `<gherardo.varando@gmail.com>` (ORCID)

Authors:

- Federico Carli
- Manuele Leonelli (ORCID)
- Eva Riccomagno

References

Collazo R. A., Görgen C. and Smith J. Q. Chain event graphs. CRC Press, 2018.

Görgen C., Bigatti A., Riccomagno E. and Smith J. Q. Discovery of statistical equivalence classes using computer algebra. *International Journal of Approximate Reasoning*, vol. 95, pp. 167-184, 2018.

Barclay L. M., Hutton J. L. and Smith J. Q. Refining a Bayesian network using a chain event graph. *International Journal of Approximate Reasoning*, vol. 54, pp. 1300-1309, 2013.

Smith J. Q. and Anderson P. E. Conditional independence and chain event graphs. *Artificial Intelligence*, vol. 172, pp. 42-68, 2008.

Thwaites P. A., Smith, J. Q. A new method for tackling asymmetric decision problems. *International Journal of Approximate Reasoning*, vol. 88, pp. 624–639, 2017.

See Also

Useful links:

- `https://github.com/stagedtrees/stagedtrees`
- Report bugs at `https://github.com/stagedtrees/stagedtrees/issues`

Examples

```
data("PhDArticles")
mf <- full(PhDArticles, join_unobserved = TRUE)
mod <- stages_fbhc(mf)
plot(mod)
```

`stages` *The stages of a staged event tree*

Description

Functions to get or set the stages of an object of class `sevt`.

Usage

```
stages(object)

## S3 method for class 'sevt'
stages(object)

## S3 method for class 'sevt.stgs'
print(x, ..., max = 5)

stages(object) <- value

## S3 method for class 'sevt.stgs'
```

```
x[i, ...]

## S3 replacement method for class 'sevt.stgs'
x[i, ..., fit = TRUE] <- value

## S3 method for class 'sevt.stgs'
x[[...]]

## S3 replacement method for class 'sevt.stgs'
x[[..., fit = TRUE]] <- value
```

Arguments

`object`	an object of class `sevt`.
`x`	an object of class `sevt.stgs` (obtained by `stages(object)`).
`...`	a path or context in the event tree.
`max`	integer, limit on the number of variables to print.
`value`	the stages replacement value.
`i`	index of variables in the tree.
`fit`	logical, if TRUE (default) the model will be re-fitted.

Details

This functions are the preferred way to access and modify directly the stages of an object of class `sevt`. In particular the indexing and replacing methods for the object extracted with the function `stages()` take care of checking the stages sanity and refit the object probabilities when needed. This is useful for manually setting some independence statements (see the Examples).

Value

For `stages()`: returns an object of class `sevt.stgs` which encode the stages of `object`. Objects of class `sevt.stgs` have dedicated method for sub-setting and replacing.

Stages indexing

Stages can be indexed, retrieved and replaced by the corresponding variables names and/or by paths or contexts.

In particular, `stages(object)[[var]]` extracts the stages vector corresponding to variable `var` (similarly to `object$stages[[var]]`. Alternatively `stages(object)[[path]]` indexes a stage via the corresponding path from root (similar to `get_stage`); a path is recognized as such if named or if of length > 2.

`stages(object)[var, context]` extracts multiple stages corresponding to a variable and eventually filtered by a specific context on the preceding variables.

Examples

```
# start with full model
mod <- full(Titanic)

# impose the context independence Survived indep Sex, Age | Class = 1st
stages(mod)["Survived", Class = "1st"] <- "C1"

# impose Survived indep Class | Class in (2nd 3rd)
stages(mod)["Survived", Class = "3rd"] <- stages(mod)["Survived", Class = "2nd"]

# impose Age indep Class | Sex
stages(mod)["Age", Sex = "Female"] <- "S-female"
stages(mod)["Age", Sex = "Male"] <- "S-male"

# stages of Survived
stages(mod)[["Survived"]]

# stages of Survived and Age
stages(mod)[c("Survived", "Age")]

# stages of Survived in the context Class 2nd or 3rd
stages(mod)["Survived", Class = c("2nd", "3rd")]

# check independencies
as_parentslist(mod)
```

`stages_bhc` *Backward hill-climbing*

Description

Greedy search of staged event trees with iterative joining of stages.

Usage

```
stages_bhc(
  object,
  score = function(x) {
     return(-BIC(x))
 },
  max_iter = Inf,
  scope = NULL,
  ignore = object$name_unobserved,
  trace = 0
)
```

Arguments

`object`	an object of class `sevt` with fitted probabilities and data, as returned by `full` or `sevt_fit`.
`score`	the score function to be maximized.
`max_iter`	the maximum number of iterations per variable.
`scope`	names of variables that should be considered for the optimization.
`ignore`	vector of stages which will be ignored and left untouched, by default the name of the unobserved stages stored in `object$name_unobserved`.
`trace`	if >0 increasingly amount of info is printed (via `message`).

Details

For each variable the algorithm tries to join stages and moves to the best model that increases the score. When no increase is possible it moves to the next variable.

Value

The final staged event tree obtained.

Examples

```
DD <- generate_xor_dataset(p = 4, n = 100)
model <- stages_bhc(full(DD), trace = 2)
summary(model)
```

`stages_bhcr` *Backward random hill-climbing*

Description

Randomly try to join stages. This is a pretty-useless function, used for comparisons.

Usage

```
stages_bhcr(
  object,
  score = function(x) {
     return(-BIC(x))
 },
  max_iter = 100,
  trace = 0
)
```

Arguments

`object`	an object of class `sevt`.
`score`	the score function to be maximized.
`max_iter`	the maximum number of iteration.
`trace`	if >0 increasingly amount of info is printed (via `message`).

Details

At each iteration a variable and two of its stages are randomly selected. If joining the stages increases the score, the model is updated. The procedure is repeated until the number of iterations reaches `max_iter`.

Value

an object of class `sevt`.

Examples

```
DD <- generate_xor_dataset(p = 4, n = 100)
model <- stages_bhcr(full(DD), trace = 2)
summary(model)
```

`stages_bj` *Backward joining of stages*

Description

Join stages from more complex to simpler models using a distance and a threshold value.

Usage

```
stages_bj(
  object,
  distance = "kullback",
  thr = 0.1,
  scope = NULL,
  ignore = object$name_unobserved,
  trace = 0
)
```

Arguments

`object`	an object of class `sevt` with fitted probabilities and data, as returned by `full` or `sevt_fit`.
`distance`	character, see details.
`thr`	the threshold for joining stages

`scope`	names of variables that should be considered for the optimization.
`ignore`	vector of stages which will be ignored and left untouched, by default the name of the unobserved stages stored in `object$name_unobserved`.
`trace`	if >0 increasingly amount of info is printed (via `message`).

Details

For each variable in the model stages are joined iteratively. At each iteration the two stages with minimum distance are selected and joined if their distance is less than `thr`.

Available distances are: manhattan (`manhattan`), euclidean (`euclidean`), Renyi divergence (`reny`), Kullback-Liebler (`kullback`), total-variation (`totvar`), squared Hellinger (`hellinger`), Bhattacharyya (`bhatt`), Chan-Darwiche (`chandarw`). See also probdist.

Value

The final staged event tree obtained.

Examples

```
DD <- generate_xor_dataset(p = 5, n = 1000)
model <- stages_bj(full(DD, lambda = 1), trace = 2)
summary(model)
```

`stages_csbhc` *Context-specific Backward hill-climbing*

Description

Greedy search of staged event trees with iterative joining of stages.

Usage

```
stages_csbhc(
  object,
  score = function(x) {
     return(-BIC(x$ll))
 },
  max_iter = Inf,
  scope = NULL,
  ignore = object$name_unobserved
)
```

Arguments

`object`	an object of class `sevt` with fitted probabilities and data, as returned by `full` or `sevt_fit`.
`score`	the score function to be maximized.
`max_iter`	the maximum number of iterations per variable.
`scope`	names of variables that should be considered for the optimization.
`ignore`	vector of stages which will be ignored and left untouched, by default the name of the unobserved stages stored in `object$name_unobserved`.

Details

For each variable the algorithm tries to join stages , by adding context specific independences, and moves to the best model that increases the score. When no increase is possible it moves to the next variable.

Value

The final staged event tree obtained.

Examples

```
model <- stages_csbhc(full(Titanic))
summary(model)
```

`stages_fbhc` *Fast backward hill-climbing*

Description

Greedy search of staged event trees with iterative joining of stages.

Usage

```
stages_fbhc(
  object,
  score = function(x) {
     return(-BIC(x))
 },
  max_iter = Inf,
  scope = NULL,
  ignore = object$name_unobserved,
  trace = 0
)
```

Arguments

`object`	an object of class `sevt` with fitted probabilities and data, as returned by `full` or `sevt_fit`.
`score`	the score function to be maximized.
`max_iter`	the maximum number of iteration.
`scope`	names of variables that should be considered for the optimization.
`ignore`	vector of stages which will be ignored and left untouched, by default the name of the unobserved stages stored in `object$name_unobserved`.
`trace`	if >0 increasingly amount of info is printed (via `message`).

Details

For each variable the algorithm tries to join stages and moves to the first model that increases the score. When no increase is possible it moves to the next variable.

Value

The final staged event tree obtained.

Examples

```
DD <- generate_xor_dataset(p = 5, n = 100)
model <- stages_fbhc(full(DD), trace = 2)
summary(model)
```

`stages_hc` *Hill-climbing*

Description

Greedy search of staged event trees with iterative moving of nodes between stages.

Usage

```
stages_hc(
  object,
  score = function(x) {
     return(-BIC(x))
 },
  max_iter = Inf,
  scope = NULL,
  ignore = object$name_unobserved,
  trace = 0
)
```

Arguments

`object`	an object of class `sevt` with fitted probabilities and data, as returned by `full` or `sevt_fit`.
`score`	the score function to be maximized.
`max_iter`	the maximum number of iterations per variable.
`scope`	names of variables that should be considered for the optimization
`ignore`	vector of stages which will be ignored and left untouched, by default the name of the unobserved stages stored in `object$name_unobserved`.
`trace`	if >0 increasingly amount of info is printed (via `message`).

Details

For each variable node-moves that best increases the score are performed until no increase is possible. A node-move is either changing the stage associate to a node or move the node to a new stage.

The `ignore` argument can be used to specify stages that should not be affected during the search, that is left untouched. This is useful for preserving structural zeroes and to speed-up computations.

Value

The final staged event tree obtained.

Examples

```
start <- indep(PhDArticles[, 1:5], join_unobserved = TRUE)
model <- stages_hc(start)
```

`stages_hclust` *Learn a staged tree with hierarchical clustering*

Description

Build a stage event tree with `k` stages for each variable by clustering stage probabilities with hierarchical clustering.

Usage

```
stages_hclust(
  object,
  distance = "totvar",
  k = NA,
  method = "complete",
  ignore = object$name_unobserved,
  limit = length(object$tree),
  scope = NULL,
```

```
    score = function(x) {
      return(-BIC(x))
    }
  )
```

Arguments

`object`	an object of class `sevt` with fitted probabilities and data, as returned by `full` or `sevt_fit`.
`distance`	character, the distance measure to be used, either a possible `method` for `dist` or one of the following: `"totvar"`, `"hellinger"`.
`k`	integer or (named) vector: number of clusters, that is stages per variable. Values will be recycled if needed. If `NA` (default) a search of the number of stage is performed with respect to the maximization of the `score` function. `NA` and integer can be mixed to fix the number of stage for some variables and use the score to select others.
`method`	the agglomeration method to be used in `hclust`.
`ignore`	vector of stages which will be ignored and left untouched. By default the name of the unobserved stages stored in `object$name_unobserved`.
`limit`	the maximum number of variables to consider.
`scope`	names of the variables to consider.
`score`	A function. Score to maximize for automatic selection of the number of stages. Used if `k=NA` for some variables.

Details

`hclust_sevt` performs hierarchical clustering of the initial stage probabilities in `object` and it aggregates them into the specified number of stages (`k`). A different number of stages for the different variables in the model can be specified by supplying a (named) vector via the argument `k`. If `k` is `NA` for some variables, all possible number of stages will be checked and the one that maximize the `score` will be selected.

Value

A staged event tree object.

Examples

```
data("Titanic")
model <- stages_hclust(full(Titanic, join_unobserved = TRUE, lambda = 1), k = 2)
summary(model)

### or search k via BIC minimization
model1 <- stages_hclust(full(Titanic), k = NA)
```

`stages_kmeans` *Learn a staged tree with k-means clustering*

Description

Build a stage event tree with `k` stages for each variable by clustering (transformed) probabilities with k-means.

Usage

```
stages_kmeans(
  object,
  k = length(object$tree[[1]]),
  algorithm = "Hartigan-Wong",
  transform = sqrt,
  ignore = object$name_unobserved,
  limit = length(object$tree),
  scope = NULL,
  nstart = 1
)
```

Arguments

`object`	an object of class `sevt` with fitted probabilities and data, as returned by `full` or `sevt_fit`.
`k`	integer or (named) vector: number of clusters, that is stages per variable. Values will be recycled if needed.
`algorithm`	character: as in `kmeans`.
`transform`	function applied to the probabilities before clustering.
`ignore`	vector of stages which will be ignored and left untouched, by default the name of the unobserved stages stored in `object$name_unobserved`.
`limit`	the maximum number of variables to consider.
`scope`	names of the variables to consider.
`nstart`	as in `kmeans`

Details

`kmenas_sevt` performs k-means clustering to aggregate the stage probabilities of the initial staged tree `object`. Different values for `k` can be specified by supplying a (named) vector to `k`. `kmeans` from the `stats` package is used internally and arguments `algorithm` and `nstart` refer to the same arguments as `kmeans`.

Value

A staged event tree.

Examples

```
data("Titanic")
model <- stages_kmeans(full(Titanic, join_unobserved = TRUE, lambda = 1), k = 2)
summary(model)
```

`stages_simplebhc` *Backward hill-climbing for simple staged trees*

Description

Greedy search of simple staged event trees with iterative joining of positions.

Usage

```
stages_simplebhc(
  object,
  score = function(x) {
     return(-BIC(x))
 },
  scope = NULL,
  max_iter = Inf,
  ignore = object$name_unobserved
)
```

Arguments

`object`	an object of class `sevt` with fitted probabilities and data, as returned by `full` or `sevt_fit`.
`score`	the score function to be maximized.
`scope`	names of variables that should be considered for the optimization.
`max_iter`	the maximum number of iterations per variable.
`ignore`	vector of stages which will be ignored and left untouched, by default the name of the unobserved stages stored in `object$name_unobserved`.

Details

This function is similar to the classical backward hill-climbing implemented in `stages_bhc`, but instead of joining stages it consider joining of *positions* via `join_positions`. Thus, the search is in the space of simple staged tree models if the initial stage tree is simple. See the references for additional details.

Value

an object of class `sevt`, the simple staged tree resulting from the search.

References

Leonelli M, Varando G. Structural Learning of Simple Staged Trees, *arXiv preprint* arXiv:2203.04390v1

See Also

`join_positions()` `sevt_simplify()`

Examples

```
mod <- stages_simplebhc(full(Titanic))
plot(mod)
```

`stndnaming` *Standard renaming of stages*

Description

Rename all stages in a staged event tree.

Usage

```
stndnaming(
  object,
  uniq = FALSE,
  prefix = FALSE,
  ignore = object$name_unobserved
)
```

Arguments

`object`	an object of class `sevt`.
`uniq`	logical, if stage numbers should be unique over all tree.
`prefix`	logical, if stage names should be prefixed with variable name.
`ignore`	vector of stages which will be ignored and left untouched, by default the name of the unobserved stages stored in `object$name_unobserved`.

Value

a staged event tree object with stages named with consecutive integers.

Examples

```
model <- stages_fbhc(full(PhDArticles, join_unobserved = TRUE))
model$stages
model1 <- stndnaming(model)
model1$stages

### unique stage names in all tree
model2 <- stndnaming(model, uniq = TRUE)
model2$stages

### prefix stage names with variable name
model3 <- stndnaming(model, prefix = TRUE)
model3$stages

### manuallty select stage names left untouched
model4 <- stndnaming(model, ignore = c("2", "6"), prefix = TRUE)
model4$stages
```

`subtree` *Extract subtree*

Description

Extract subtree

Usage

```
subtree(object, path)
```

Arguments

`object`	an object of class `sevt`.
`path`	the path from root after which extract the subtree.

Details

Returns the subtree of the staged event tree, starting from `path`.

Value

A staged event tree object corresponding to the subtree.

Examples

```
DD <- generate_random_dataset(4, 100)
model <- sevt(DD, full = TRUE)
plot(model)
model1 <- subtree(model, path = c("-1", "1"))
plot(model1)
```

`summary.sevt` *Summarizing staged event trees*

Description

Summary method for class `sevt`.

Usage

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

## S3 method for class 'summary.sevt'
print(x, max = 10, ...)
```

Arguments

`object`	an object of class `sevt`.
`...`	arguments for compatibility.
`x`	an object of class `summary.sevt`.
`max`	the maximum number of variables for which information is printed.

Details

Print model information and summary of stages.

Value

An object of class `summary.sevt` for which a `print` method exist.

Examples

```
model <- stages_fbhc(full(PhDArticles, lambda = 1))
summary(model)
```

`text.sevt` *Add text to a staged event tree plot*

Description

Add text to a staged event tree plot

Usage

```
## S3 method for class 'sevt'
text(x, y = ylim[1], limit = 10, xlim = c(0, 1), ylim = c(0, 1), ...)
```

Arguments

`x`	An object of class `sevt`.
`y`	the position of the labels.
`limit`	maximum number of variables plotted.
`xlim`	graphical parameter.
`ylim`	graphical parameter.
`...`	additional parameters passed to `text`.

`trajectories` *Hospital trajectories*

Description

Generated dataset with observations from five variables (SEX, AGE, ICU, RSP, OUT) describing imaginary patients' trajectories in a hospital.

Usage

```
trajectories
```

Format

A data frame with 10000 observations of 5 variables.

Source

The data has been generated with the code in the Examples section.

Examples

```
library("stagedtrees")

tree <- list(SEX = c("male", "female"),
             AGE = c("child", "adult", "elder"),
             ICU = c("0", "1"),
             RSP = c("intub", "mask", "no"),
             OUT = c("death", "survived"))

model <- sevt(tree, full = TRUE)

stages(model)["ICU", AGE = "child"] <- "ICUchild"

stages(model)["ICU", SEX = "male", AGE = "elder"] <-
  stages(model)["ICU", SEX = "female", AGE = "elder"]

stages(model)["RSP", AGE = c("child"), ICU = "0"] <- "childnoICU"
stages(model)["RSP", AGE = c("child"), ICU = "1"] <- "childICU"
```

```
stages(model)["RSP", AGE = c("adult")] <- stages(model)["RSP", AGE = c("elder")]


stages(model)["OUT", AGE = "adult",
              SEX = "female",
              ICU = "1",
              RSP = c("intub", "mask")] <- "femaleICUresp"

stages(model)["OUT", AGE = "child",
                     ICU = "1",
                     RSP = "intub"] <- "childICUintub"

stages(model)["OUT", AGE = "child",
              ICU = "1",
              RSP = "mask"] <- "childICUmask"

stages(model)["OUT", AGE = "child",
              ICU = "1",
              RSP = "no"] <- "childICUno"

stages(model)["OUT", AGE = "adult", SEX = "male"] <-
  stages(model)["OUT", AGE = "elder", SEX = "female"]

stages(model)["OUT", ICU = "0", RSP = "intub"] <- "UNOBS"
stages(model)["OUT", ICU = "0", RSP = "intub"] <- "UNOBS"
stages(model)["OUT", AGE = "child", ICU = "0"] <- "UNOBS"

model$prob <- list()
model$prob$SEX <- list( "NA" = c(male = 0.4, female = 0.6))
model$prob$AGE <- list("1" = c("child" = 0.1, "adult" = 0.5, "elder" = 0.4),
                       "2" = c("child" = 0.1, "adult" = 0.3, "elder" = 0.6))

model$prob$ICU <- list("ICUchild" = c("0" = 0, "1" = 1),
                       "2" = c("0" = 0.4, "1" = 0.6), ## male adult
                       "5" = c("0" = 0.2, "1" = 0.8), ## female adult
                       "6" = c("0" = 0.7, "1" = 0.3)) ## elder

model$prob$RSP <- list("childnoICU" = c("intub" = NA, "mask" = NA, "no" = NA),
                       "childICU" = c("intub" = 0.1, "mask" = 0.7, "no" = 0.2),
                       "5" = c("intub" = 0, "mask" = 0.7, "no" = 0.3), # male noICU
                       "6" = c("intub" = 0.4, "mask" = 0.5, "no" = 0.1), # male ICU
                       "11" = c("intub" = 0, "mask" = 0.5, "no" = 0.5), # female noICU
                       "12" = c("intub" = 0.4, "mask" = 0.5, "no" = 0.1)) # female ICU


model$prob$OUT <- list("UNOBS" = c("death" = NA, "survived" = NA),
                       "childICUintub" = c("death" = 0.03, "survived" = 0.97),
                       "childICUmask" = c("death" = 0.02, "survived" = 0.98),
                       "childICUno" = c("death" = 0.01, "survived" = 0.99),
                       ### male adult and female elder ICU = 0 :
                       "32" = c("death" = 0.05, "survived" = 0.95), ## mask
                       "33" = c("death" = 0.01, "survived" = 0.99), ## no
```

```
                       ### male adult and female elder ICU = 1 :
                       "34" = c("death" = 0.15, "survived" = 0.85), ## intub
                       "35" = c("death" = 0.08, "survived" = 0.92), ## mask
                       "36" = c("death" = 0.04, "survived" = 0.96), ## no
                       ##############
                       "14" = c("death" = 0.2, "survived" = 0.8), # male elder 0 mask
                       "15" = c("death" = 0.1, "survived" = 0.9), # male elder 0 no
                       "16" = c("death" = 0.3, "survived" = 0.7), # male elder 1 intub
                       "17" = c("death" = 0.25, "survived" = 0.75), # male elder 1 mask
                       "18" = c("death" = 0.3, "survived" = 0.7), # male elder 1 no
                       ##############
                       "26" = c("death" = 0.1, "survived" = 0.9), # female adult 0 mask
                       "27" = c("death" = 0.15, "survived" = 0.85), # female adult 0 no
                       "30" = c("death" = 0.2, "survived" = 0.8),  # female adult 1 no
                       ##############
                       "femaleICUresp" = c("death" = 0.1, "survived" = 0.9)
                       )

# trajectories <- sample_from(model, 10000, seed = 1)
# usethis::use_data(trajectories, overwrite = TRUE)
```

`write_tikz` *Export the staged tree or CEG graph to tikz*

Description

Generate tikz code to draw the staged tree or CEG graph.

Usage

```
write_tikz(
  x,
  layout = NULL,
  file = "",
  col = NULL,
  ignore = x$name_unobserved,
  node_label = function(node) {
     ifelse(is.na(node$stage), "", node$stage)
 },
  edge_label = function(edge) {
     ifelse(is.na(edge$label), "", edge$label)
 },
  edge_label_options = function(edge) {
     return("sloped")
 },
  scale = 10,
  normalize_layout = TRUE,
  node_shape = "circle",
  node_inner_sep = "1mm",
```

```
  node_minimum_size = "0.3cm",
  node_draw_color = "black",
  node_thickness = "very thick",
  node_text_color = "black"
)

## S3 method for class 'sevt'
write_tikz(
  x,
  layout = NULL,
  file = "",
  col = NULL,
  ignore = x$name_unobserved,
  node_label = function(node) {
     ifelse(is.na(node$stage), "", node$stage)
 },
  edge_label = function(edge) {
     ifelse(is.na(edge$label), "", edge$label)
 },
  edge_label_options = function(edge) {
     return("sloped")
 },
  scale = 10,
  normalize_layout = TRUE,
  node_shape = "circle",
  node_inner_sep = "1mm",
  node_minimum_size = "0.3cm",
  node_draw_color = "black",
  node_thickness = "very thick",
  node_text_color = "black"
)
```

Arguments

`x`	an object of class `sevt` or `ceg`.
`layout`	the layout of the graph, given as matrix with two columns and as many rows as nodes in the staged tree. By default, a modified sugiyama layout is used. The layout matrix can be obtained with **igraph** layout functions.
`file`	A connection or a character string naming the file to print to. Passed to `cat`.
`col`	color specifications for the stages of the staged even tree. Same as `plot.sevt` and `make_stages_col`.
`ignore`	vector of stages which will be ignored and not plotted, by default the name of the unobserved stages stored in `x$name_unobserved`.
`node_label`	a function that produces nodes labels.
`edge_label`	a function that produces edge labels.
`edge_label_options`	a function that produces edge label options.

`scale`	for the tikzfigure.
`normalize_layout`	a logical value. If `TRUE` layout positions are scaled to the `[0,1]` interval.
`node_shape`	the shape to be used for nodes.
`node_inner_sep`	the `inner sep` parameter.
`node_minimum_size`	the `minimum size` parameter for the nodes.
`node_draw_color`	the color for line drawing the nodes.
`node_thickness`	the thickness of the lines.
`node_text_color`	the color for label in nodes.

Details

This function can be used to create a working tikz code that compile to a graph similar to the one obtained by `plot.sevt(x, ...)` or `plot.ceg(x, ...)`.

References

Code partially inspired by the code in *Exporting graphs to LaTeX, using igraph and TikZ* `http://igraph.wikidot.com/r-recipes#toc2`

Index