Package 'R6causal'

November 4, 2022

Type Package

Title R6 Class for Structural Causal Models

Version 0.7.0

Maintainer Juha Karvanen <juha.karvanen@iki.fi>

Description The implemented R6 class 'SCM' aims to simplify working with structural causal models. The missing data mechanism can be defined as a part of the structural model. The class contains methods for 1) defining a structural causal model via functions, text or conditional probability tables, 2) printing basic information on the model, 3) plotting the graph for the model using packages 'igraph' or 'qgraph', 4) simulating data from the model, 5) applying an intervention, 6) checking the identifiability of a query using the R packages 'causaleffect' and 'dosearch', 7) defining the missing data mechanism, 8) simulating incomplete data from the model according to the specified missing data mechanism and 9) checking the identifiability in a missing data problem using the R package 'dosearch'. In addition, there are functions for running experiments and doing counterfactual inference using simulation.

License AGPL-3

Encoding UTF-8

RoxygenNote 7.2.1

Imports causaleffect, data.table, dosearch, igraph, R6, stats

Suggests rmarkdown, knitr, qgraph, sqldf

VignetteBuilder knitr

NeedsCompilation no

Author Juha Karvanen [aut, cre] (<https://orcid.org/0000-0001-5530-769X>)

Repository CRAN

Date/Publication 2022-11-04 08:00:02 UTC

R topics documented:

`backdoor` *SCM "backdoor" used in the examples.*

Description

Variable z fulfills the back-door criterion for P(y|do(x))

Usage

```
backdoor
```

Format

An object of class `SCM` (inherits from `R6`) of length 27.

Examples

```
backdoor
backdoor$plot()
```

`backdoor_md` *SCM "backdoor_md" used in the examples.*

Description

Variable z fulfills the back-door criterion for P(y|do(x)). Variable z is missing completely at random. The missingness of variables x and y depend on z.

Usage

```
backdoor_md
```

Format

An object of class `SCM` (inherits from `R6`) of length 27.

Examples

```
backdoor_md
backdoor_md$plot()
```

`counterfactual` *Counterfactual inference via simulation*

Description

Counterfactual inference via simulation

Usage

```
counterfactual(
  scm,
  situation,
  n,
  target = NULL,
  ifunction = NULL,
  returnscm = FALSE,
  control = list()
)
```

Arguments

`scm`	An SCM object
`situation`	A list or a character string. The list has the following elements: • do : NULL or a list containing named elements 'target' and 'ifunction' that specify the intervention carried out in the situation • dolist : NULL or a list of lists containing named elements 'target' and 'ifunction' that specify the intervention carried out in each parallel world • condition : either a string that gives an SQL query (e.g. "select x,y,z from DATA where") or a data.table consisting of the valid rows (e.g. data.table::data.table(x = 0, y = 0))
`n`	Size of the data to be simulated
`target`	NULL or a vector of variable names that specify the target variable(s) of the counterfactual intervention.
`ifunction`	NULL or a list of functions for the counterfactual intervention.
`returnscm`	A logical, should the internally created twin SCM or parallel world SCM returned?
`control`	List of parameters to be passed to the simulation method

Value

A data table representing the situation after the counterfactual intervention

Examples

```
cfdata <- counterfactual(backdoor,
                         situation = list(
                             do = list(target = "x", ifunction = 0),
                             condition = data.table::data.table( x = 0, y = 0)),
                         target = "x",
                         ifunction = 1,
                         n = 1000)
mean(cfdata$y)

backdoor_parallel <- ParallelWorld$new(backdoor,
                                       dolist=list(
                                         list(target = "x", ifunction = 0),
                                       list(target = list("z","x"), ifunction = list(1,0))
                                         )
)
cfdata2 <- counterfactual(backdoor_parallel,
                         situation = list(
                             do = NULL,
                           condition = data.table::data.table( y = 0, y_1 = 0, y_2 = 0)),
                         target = "x",
                         ifunction = 1,
                         n = 1000)
mean(cfdata2$y)
```

`frontdoor` *SCM "frontdoor" used in the examples.*

Description

Variable z fulfills the front-door criterion for P(y|do(x))

Usage

```
frontdoor
```

Format

An object of class `SCM` (inherits from `R6`) of length 27.

Examples

```
frontdoor
frontdoor$plot()
```

generate_condprob *Define structural function by a conditional probability table*

Description

Define structural function by a conditional probability table

Usage

```
generate_condprob(ycondx, x, Umerge_expr = NULL)
```

Arguments

ycondx	A data table or a data frame with the following structure • 1st column: variable to be generated, "Y" • middle columns: the parents of the the 1st column variable, "X" • last column: the probability the case specified be the other columns, "P(Y|do(X))"
x	A data table or a data frame that contains data on the variables in the middle columns of ycondx, "X" and one or more columns giving data on U-variables.
Umerge_expr	A character string specifying how the U-variables will be combined when the value "Y" is generated, e.g. "u" or "(u1+u2)/2". The result of the expression should be a random number in the interval [0,1].

Value

A data table containing the generated variable, "Y"

Examples

```
ycondx <- data.table::data.table(y =rep(c(0,1), each = 3), x=rep(1:3, 2),
                     prob = c(0.2,0.6,0.1,0.8,0.4,0.9))
x <- data.table::data.table(x = sample(1:3, 20, replace = TRUE),
                       uy = stats::runif(20), uy2 = stats::runif(20))
generate_condprob(ycondx, x, Umerge_expr = "(uy+uy2)/2")
```

ParallelWorld *R6 Class for parallel world models*

Description

R6 Class for parallel world models

R6 Class for parallel world models

Details

Inherits R6 class SCM.

Super class

`R6causal::SCM` -> `ParallelWorld`

Active bindings

`num_worlds` Number of parallel worlds.

`worldnames` Names of parallel worlds.

`worldsuffix` Suffix used for parallel world variables.

`originalscm` SCM from which the parallel worlds are derived.

`dolist` List containing the interventions for each world.

Methods

Public methods:

- `ParallelWorld$new()`
- `ParallelWorld$clone()`

Method `new()`**:** Create a new ParallelWorld object from an SCM object.

Usage:

```
ParallelWorld$new(scm, dolist, worldnames = NULL, worldsuffix = "_")
```

Arguments:

`scm` An SCM object.

`dolist` A list containing the interventions for each world. Each element of the list has the fields:

- target: a vector of variable names that specify the target variable(s) of the counterfactual intervention.
- ifunction: a list of functions for the counterfactual intervention.

`worldnames` A character vector giving the names of the parallel worlds.

`worldsuffix` A text giving the suffix used for parallel world variables before the world number. Defaults to "_" and the worlds have then suffixes "_1", "_2", "_3", ...

Returns: A new 'ParallelWorld' object that also belongs to class 'SCM'.

Examples:

```
backdoor_parallel <- ParallelWorld$new(
                        backdoor,
                        dolist=list(
                            list(target = "x",
                                 ifunction = 0),
                            list(target = list("z","x"),
                                 ifunction = list(1,0))
                        )
```

```
)
backdoor_parallel
plot(backdoor_parallel)
```

Method `clone()`**:** The objects of this class are cloneable with this method.

Usage:

```
ParallelWorld$clone(deep = FALSE)
```

Arguments:

`deep` Whether to make a deep clone.

Examples

```
## ------------------------------------------------
## Method `ParallelWorld$new`
## ------------------------------------------------

backdoor_parallel <- ParallelWorld$new(
                        backdoor,
                        dolist=list(
                            list(target = "x",
                                 ifunction = 0),
                            list(target = list("z","x"),
                                 ifunction = list(1,0))
                        )
)
backdoor_parallel
plot(backdoor_parallel)
```

`R6causal` *R6causal: R6 class for structural causal models*

Description

Package R6causal implements an R6 class for structural causal models (SCM) with latent variables and missing data mechanism. The class contains methods for 1) defining a structural causal model via functions, text or conditional probability tables, 2) printing basic information on the model, 3) plotting the graph for the model using packages 'igraph' or 'qgraph', 4) simulating data from the model, 5) applying an intervention, 6) checking the identifiability of a query using the R packages 'causaleffect' and 'dosearch', 7) defining the missing data mechanism, 8) simulating incomplete data from the model according to the specified missing data mechanism and 9) checking the identifiability in a missing data problem using the R package 'dosearch'. In addition, there are functions for running experiments and doing counterfactual inference using simulation.

References

J. Pearl (2009). Causality, 2nd edition, Cambridge University Press.

run_experiment *Conduct a sequence of interventions and collect the simulated data.*

Description

Conduct a sequence of interventions and collect the simulated data.

Usage

```
run_experiment(SCM, intervene, response, n)
```

Arguments

SCM	An SCM object
intervene	A list where the names of the elements are the variables to be intervened and the values of the elements are vectors specifying the values set in the intervention
response	A vector of the names of the response variables
n	Size of the data to be simulated for each intervention

Value

A list containing the values of the response variables for all intervention combinations

Examples

```
backdoor_experiment <- run_experiment(backdoor,
                                      intervene = list(x = c(0,1)),
                                      response = "y",
                                      n = 10000)
colMeans(backdoor_experiment$response_list$y)
```

SCM *R6 Class for structural causal models*

Description

R6 Class for structural causal models

R6 Class for structural causal models

Details

An R6 class for structural causal models (SCM) with latent variables and missing data mechanism. There are methods for defining, printing, plotting, intervening and simulating SCMs.

Active bindings

`vflist` List of the structural functions of observed variables.

`vfsymb` List of the names of observed variables.

`uflist` List of the structural functions of unobserved variables.

`ufsymb` List of the names of unobserved variables.

`rflist` List of the structural functions of missingness indicators.

`rfsymb` List of the names of missingness indicators.

`rprefix` Prefix used to mark missingness indicators.

`starsuffix` Suffix used to mark variables with missing data.

`simdata` Data table containing data simulated from the SCM.

`simdata_md` Data table containing data simulated from the SCM where missing values are indicated by `NA`.

`igraph` The graph of the SCM in the `igraph` form (without the missing data mechanism).

`igraph_bidirected` The graph of the SCM in the `igraph` form where latent variables are presented by bidirected arcs.

`igraph_md` The graph of the SCM in the `igraph` form including the missing data mechanism.

`toporder` A vector giving the topological order of variables.

`toporderv` A vector giving the topological order of observed variables.

`graphtext` A character string that gives the edges of the graph of the SCM (without the missing data mechanism).

`graphtext_md` A character string that gives the edges of the graph of the SCM including the missing data mechanism.

`name` The name of the SCM.

Methods

Public methods:

- `SCM$new()`
- `SCM$print()`
- `SCM$plot()`
- `SCM$intervene()`
- `SCM$simulate()`
- `SCM$causal.effect()`
- `SCM$dosearch()`
- `SCM$clone()`

Method `new()`**:** Create a new SCM object.

Usage:

```
SCM$new(
  name,
  uflist,
  vflist,
  rflist = NULL,
  rprefix = "R_",
  starsuffix = "_md"
)
```

Arguments:

`name` Name.

`uflist` A named list containing the functions for latent variables.

`vflist` A named list containing the functions for observed variables.

`rflist` A named list containing the functions for missingness indicators.

`rprefix` The prefix of the missingness indicators.

`starsuffix` The suffix for variables with missing data.

Returns: A new 'SCM' object.

Examples:

```
backdoor <- SCM$new("backdoor",
 uflist = list(
  uz = function(n) {return(stats::runif(n))},
  ux = function(n) {return(stats::runif(n))},
  uy = function(n) {return(stats::runif(n))}
 ),
 vflist = list(
  z = function(uz) {
    return(as.numeric(uz < 0.4))},
  x = function(ux, z) {
    return(as.numeric(ux < 0.2 + 0.5*z))},
  y = function(uy, z, x) {
    return(as.numeric(uy < 0.1 + 0.4*z + 0.4*x))}
 )
)
```

Method `print()`**:** Print a summmary of the SCM object.

Usage:

```
SCM$print()
```

Examples:

```
backdoor
```

Method `plot()`**:** Plot the DAG of the SCM object.

Usage:

```
SCM$plot(subset = "uvr", method = "igraph", ...)
```

Arguments:

`subset` Variable groups to be plotted: "uvr","vr","uv", or "v".

`method` Plotting method: "qgraph" or "igraph".

`...` other parameters passed to the plotting method

Examples:

```
backdoor$plot()
backdoor$plot("v")
```

Method `intervene()`**:** Apply an intervention to the SCM object.

Usage:

```
SCM$intervene(target, ifunction)
```

Arguments:

`target` Name(s) of the variables in vflist to be intervened.

`ifunction` Either numeric value(s) or new structural function(s) for the target variables.

Examples:

```
# A simple intervention
backdoor_x1 <- backdoor$clone()  # making a copy
backdoor_x1$intervene("x",1) # applying the intervention
backdoor_x1$plot() # to see that arrows incoming to x are cut

# An intervention that redefines a structural equation
backdoor_yz <- backdoor$clone()  # making a copy
backdoor_yz$intervene("y",
   function(uy, z) {return(as.numeric(uy < 0.1 + 0.8*z ))}) # making y a function of z only
backdoor_yz$plot() # to see that arrow x -> y is cut
```

Method `simulate()`**:** Simulate data from the SCM object. Returns simulated data as a data.table and/or creates or updates `simdata` in the SCM object. If `no_missing_data = FALSE`, creates or updates also `simdata_md`

Usage:

```
SCM$simulate(
  n = 1,
  no_missing_data = FALSE,
  seed = NULL,
  fixedvars = NULL,
  store_simdata = TRUE,
  return_simdata = FALSE
)
```

Arguments:

`n` Number of observations to be generated.

`no_missing_data` Logical, should the generation of missing data skipped? (defaults FALSE).

`seed` NULL or a number for `set.seed`.

`fixedvars` List of variables that remain unchanged.

`store_simdata` Logical, should the simulated data to be stored in the SCM object (defaults TRUE)

`return_simdata` Logical, should the simulated data to be returned as the output (defaults FALSE)

Examples:

```
backdoor$simulate(8, return_simdata = TRUE, store_simdata = FALSE)
backdoor$simulate(10)
backdoor$simdata
```

Method `causal.effect()`**:** Is a causal effect identifiable from observational data? Calls the implementation of ID algorithm from package **causaleffect**. See the documentation of `causal.effect` for the details.

Usage:

```
SCM$causal.effect(y, x, ...)
```

Arguments:

`y` A vector of character strings specifying target variable(s).

`x` A vector of character strings specifying intervention variable(s).

`...` Other parameters passed to `causal.effect`.

Returns: An expression for the joint distribution of the set of variables (y) given the intervention on the set of variables (x) conditional on (z) if the effect is identifiable. Otherwise an error is thrown describing the graphical structure that witnesses non-identifiability. @examples backdoor$causal.effect(y = "y", x = "x")

Method `dosearch()`**:** Is a causal effect or other query identifiable from given data sources? Calls `dosearch` from the package **dosearch**. See the documentation of dosearch for the details.

Usage:

```
SCM$dosearch(
  data,
  query,
  transportability = NULL,
  selection_bias = NULL,
  missing_data = NULL,
  control = list()
)
```

Arguments:

`data` Character string specifying the data sources.

`query` Character string specifying the query of interest.

`transportability` Other parameters passed to `dosearch()`.

`selection_bias` Other parameters passed to `dosearch()`.

`missing_data` Other parameters passed to `dosearch()`.

`control` List of control parameters passed to `dosearch()`.

Returns: An object of class `dosearch`.

Examples:

```
backdoor$dosearch(data = "p(x,y,z)", query = "p(y|do(x))")
```

Method `clone()`**:** The objects of this class are cloneable with this method.

Usage:

```
SCM$clone(deep = FALSE)
```

Arguments:

`deep` Whether to make a deep clone.

Examples

```
## ------------------------------------------------
## Method `SCM$new`
## ------------------------------------------------

backdoor <- SCM$new("backdoor",
 uflist = list(
  uz = function(n) {return(stats::runif(n))},
  ux = function(n) {return(stats::runif(n))},
  uy = function(n) {return(stats::runif(n))}
 ),
 vflist = list(
  z = function(uz) {
    return(as.numeric(uz < 0.4))},
  x = function(ux, z) {
    return(as.numeric(ux < 0.2 + 0.5*z))},
  y = function(uy, z, x) {
    return(as.numeric(uy < 0.1 + 0.4*z + 0.4*x))}
 )
)

## ------------------------------------------------
## Method `SCM$print`
## ------------------------------------------------

backdoor

## ------------------------------------------------
## Method `SCM$plot`
## ------------------------------------------------

backdoor$plot()
backdoor$plot("v")

## ------------------------------------------------
## Method `SCM$intervene`
## ------------------------------------------------

# A simple intervention
backdoor_x1 <- backdoor$clone()  # making a copy
backdoor_x1$intervene("x",1) # applying the intervention
backdoor_x1$plot() # to see that arrows incoming to x are cut

# An intervention that redefines a structural equation
backdoor_yz <- backdoor$clone()  # making a copy
backdoor_yz$intervene("y",
   function(uy, z) {return(as.numeric(uy < 0.1 + 0.8*z ))}) # making y a function of z only
backdoor_yz$plot() # to see that arrow x -> y is cut

## ------------------------------------------------
## Method `SCM$simulate`
```

```
## ------------------------------------------------

backdoor$simulate(8, return_simdata = TRUE, store_simdata = FALSE)
backdoor$simulate(10)
backdoor$simdata

## ------------------------------------------------
## Method `SCM$dosearch`
## ------------------------------------------------

backdoor$dosearch(data = "p(x,y,z)", query = "p(y|do(x))")
```

`trapdoor` *SCM "trapdoor" used in the examples.*

Description

Variable z is a trapdoor variable for P(y|do(x))

Usage

```
trapdoor
```

Format

An object of class `SCM` (inherits from `R6`) of length 27.

References

J. Helske, S. Tikka, J. Karvanen (2021). Estimation of causal effects with small data in the presence of trapdoor variables, Journal of the Royal Statistical Society Series A, 184(3), 1030-1051, http://doi.org/10.1111/rssa.12699

Examples

```
trapdoor
trapdoor$plot()
```

Index