Package 'tramicp'

January 31, 2025

Type Package

Title Model-Based Causal Feature Selection for General Response Types

Version 0.1-0

Description Extends invariant causal prediction (Peters et al., 2016, <doi:10.1111/rssb.12167>) to generalized linear and transformation models (Hothorn et al., 2018, <doi:10.1111/sjos.12291>). The methodology is described in Kook et al. (2023, <doi:10.1080/01621459.2024.2395588>).

License GPL-3

Encoding UTF-8

RoxygenNote 7.3.1

Depends R (>= 4.1.0)

Imports tram, mlt, coin, multcomp, survival, variables, basefun, MASS, cotram, dHSIC, ranger, sandwich

Suggests testthat (>= 3.0.0)

Config/testthat/edition 3

URL https://github.com/LucasKook/tramicp

BugReports https://github.com/LucasKook/tramicp/issues

NeedsCompilation no

Author Lucas Kook [aut, cre] (<https://orcid.org/0000-0002-7546-7356>), Sorawit Saengkyongam [ctb], Anton Rask Lundborg [ctb], Torsten Hothorn [ctb], Jonas Peters [ctb]

Maintainer Lucas Kook <lucasheinrich.kook@gmail.com>

Repository CRAN

Date/Publication 2025-01-31 09:50:02 UTC

Contents

`bootstrap_stability` *Bootstrap stability for TRAMICP*

Description

Bootstrap stability for TRAMICP

Usage

```
bootstrap_stability(
  object,
  B = 100,
  size = NULL,
  verbose = FALSE,
  return_all = FALSE
)
```

Arguments

`object`	Object of class `"dICP"`
`B`	Numeric; number of bootstrap iterations
`size`	Numeric; size of bootstrap samples
`verbose`	Logical; print a progress bar (default: `FALSE`)
`return_all`	Logical; return all `"dICP"` objects (default: `FALSE`)

Value

Table of output sets of candidate causal predictors

Examples

```
set.seed(12)
d <- dgp_dicp(n = 1e3, mod = "binary")
res <- glmICP(Y ~ X1 + X2 + X3, data = d, env = ~ E,
    family = "binomial", test = "cor.test")
bootstrap_stability(res, B = 2)
```

dgp_dicp *Simple data-generating process for illustrating tramicp*

Description

Simple data-generating process for illustrating tramicp

Usage

```
dgp_dicp(
  n = 1000,
  K = 6,
  nenv = 2,
  bx3 = stats::rnorm(1),
  ge = stats::rnorm(nenv),
  ae = stats::rnorm(nenv),
  mod = "polr",
  interacting = FALSE,
  rm_censoring = TRUE,
  cfb = c(-3, 1.35),
  cfx = stats::rnorm(2),
  bx2x1 = stats::rnorm(1)
)
```

Arguments

n	Sample size
K	Number of outcome classes or order of Bernstein polynomial
nenv	Number of environments
bx3	Effect of Y on X3
ge	Environment specific effect
ae	Environment specific effect
mod	Type of model
interacting	Toggle baseline interaction with env
rm_censoring	Remove censoring from simulated responses
cfb	Baseline coefs
cfx	Shift coefs
bx2x1	coef from x2 to x1

Details

Simulates from X2 -> X1 -> Y -> X3, with E affecting X1, X2, X3, but not Y.

Value

data.frame with simulated data

dicp *Model-based causal feature selection for general response types*

Description

Function 'dicp()' implements invariant causal prediction (ICP) for transformation and generalized linear models, including binary logistic regression, Weibull regression, the Cox model, linear regression and many others. The aim of ICP is to discover the direct causes of a response given data from heterogeneous experimental settings and a potentially large pool of candidate predictors.

Usage

```
dicp(
  formula,
  data,
  env,
  modFUN,
  verbose = TRUE,
  type = c("residual", "wald", "partial"),
  test = "gcm.test",
  controls = NULL,
  alpha = 0.05,
  baseline_fixed = TRUE,
  greedy = FALSE,
  max_size = NULL,
  mandatory = NULL,
  ...
)
```

Arguments

formula	A formula including response and covariate terms.
data	A data.frame containing response and explanatory variables.
env	A formula specifying the environment variables (see details).
modFUN	Model function from 'tram' (or other packages), e.g., BoxCox, Colr, Polr, Lm, Coxph, Survreg, Lehmann. Standard implementations lm, glm, survreg, coxph, and polr are also supported. See the corresponding alias <model_name>ICP, e.g., PolrICP or ?implemented_model_classes. Models from 'lme4', 'tramME', 'glmnet' and 'mgcv' are also supported.
verbose	Logical, whether output should be verbose (default TRUE).
type	Character, type of invariance ("residual" or "wald"); see Details.
test	Character, specifies the invariance test to be used when type = "residual". The default is "gcm.test". Other implemented tests are "HSIC", "t.test", "var.test", and "combined". Alternatively, a custom function for testing invariance of the form \(r, e, controls) {...} can be supplied, which outputs a list with entry "p.value".

controls	Controls for the used tests and the overall procedure, see dicp_controls.
alpha	Level of invariance test, default 0.05.
baseline_fixed	Fixed baseline transformation, see dicp_controls.
greedy	Logical, whether to perform a greedy version of ICP (default is FALSE).
max_size	Numeric; maximum support size.
mandatory	A formula containing mandatory covariates, i.e., covariates which by domain knowledge are believed to be parents of the response or are in another way required for the environment or model to be valid (for instance, conditionally valid environments or random effects in a mixed model).
...	Further arguments passed to modFUN.

Details

TRAMICP iterates over all subsets of covariates provided in formula and performs an invariance test based on the conditional covariance between score residuals and environments in env (type = "residual") or the Wald statistic testing for the presence of main and interaction effects of the environments (type = "wald"). The algorithm outputs the intersection over all non-rejected sets as an estimate of the causal parents.

Value

Object of class "dICP", containing

- candidate_causal_predictors: Character; intersection of all non-rejected sets,
- set_pvals: Numeric vector; set-specific p-values of the invariance test,
- predictor_pvals: Numeric vector; predictor-specific p-values,
- tests: List of invariance tests.

References

Kook, L., Saengkyongam, S., Lundborg, A. R., Hothorn, T., & Peters, J. (2023). Model-based causal feature selection for general response types. arXiv preprint. doi:10.48550/arXiv.2309.12833

Examples

```
set.seed(12)
d <- dgp_dicp(n = 1e3, mod = "binary")
dicp(Y ~ X1 + X2 + X3, data = d, env = ~ E, modFUN = "glm",
     family = "binomial", type = "wald")
```

dicp_controls *TRAMICP Controls*

Description

TRAMICP Controls

Usage

```
dicp_controls(
  type = "residual",
  test = "gcm.test",
  baseline_fixed = TRUE,
  alpha = 0.05,
  method = "gamma",
  kernel = c("gaussian", "discrete"),
  B = 499,
  vcov = "vcov",
  teststat = "maximum",
  distribution = "asymptotic",
  xtrafo = coin::trafo,
  ytrafo = coin::trafo,
  residuals = "residuals",
  crossfit = getOption("crossfit", default = FALSE),
 stop_if_empty_set_invariant = getOption("stop_if_empty_set_invariant", default = FALSE),
  wald_test_interactions = getOption("wald_test_interactions", default = TRUE)
)
```

Arguments

type	Character, type of invariance ("residual" or "wald"); see Details.
test	Character, specifies the invariance test to be used when type = "residual". The default is "gcm.test". Other implemented tests are "HSIC", "t.test", "var.test", and "combined". Alternatively, a custom function for testing invariance of the form \(r, e, controls) {...} can be supplied, which outputs a list with entry "p.value".
baseline_fixed	Logical; whether or not the baseline transformation is allowed to vary with the environments. Only takes effect when type is "wald".
alpha	Level of invariance test, default 0.05.
method	Only applies if test = "HSIC". See dhsic.test.
kernel	Only applies if test = "HSIC". See dhsic.test.
B	For test = "HSIC", see dhsic.test.
vcov	(Name of) function for computing the variance-covariance matrix of a model.
teststat	Only applies if test = "independence". See independence_test.

distribution — Only applies if test = "independence". See independence_test.

xtrafo — Only applies if test = "independence". See independence_test.

ytrafo — Only applies if test = "independence". See independence_test.

residuals — Character or function; (Name of) function for computing model residuals. The default is stats::residuals with methods dispatch.

crossfit — Logical; toggle for cross fitting when type = "residual".

stop_if_empty_set_invariant — Logical; dicp halts if the empty set is not rejected (the resulting intersection will always be empty). Default is FALSE and can be over-written by setting options(stop_if_empty_set_invariant = TRUE).

wald_test_interactions — Logical; whether to test for interactions between residuals and environments when using type = "wald" (wald_test_interactions = TRUE, the default) or main effects only (wald_test_interactions = FALSE).

Value

List of dicp controls containing the evaluated arguments from above.

---

implemented_model_classes

Aliases for implemented model classes

Description

ICP for Box-Cox-type transformed normal regression, parametric and semiparametric survival models, continuous outcome logistic regression, linear regression, cumulative ordered regression, generalized linear models; and nonparametric ICP via ranger. While TRAMICP based on quantile and survival random forests is also supported, for these methods it comes without theoretical guarantees as of yet.

Usage

```
BoxCoxICP(
  formula,
  data,
  env,
  verbose = TRUE,
  type = "residual",
  test = "gcm.test",
  controls = NULL,
  alpha = 0.05,
  baseline_fixed = TRUE,
  greedy = FALSE,
  max_size = NULL,
```

```
  mandatory = NULL,
  ...
)

SurvregICP(
  formula,
  data,
  env,
  verbose = TRUE,
  type = "residual",
  test = "gcm.test",
  controls = NULL,
  alpha = 0.05,
  baseline_fixed = TRUE,
  greedy = FALSE,
  max_size = NULL,
  mandatory = NULL,
  ...
)

survregICP(
  formula,
  data,
  env,
  verbose = TRUE,
  type = "residual",
  test = "gcm.test",
  controls = NULL,
  alpha = 0.05,
  baseline_fixed = TRUE,
  greedy = FALSE,
  max_size = NULL,
  mandatory = NULL,
  ...
)

coxphICP(
  formula,
  data,
  env,
  verbose = TRUE,
  type = "residual",
  test = "gcm.test",
  controls = NULL,
  alpha = 0.05,
  baseline_fixed = TRUE,
  greedy = FALSE,
  max_size = NULL,
```

```
  mandatory = NULL,
  ...
)

ColrICP(
  formula,
  data,
  env,
  verbose = TRUE,
  type = "residual",
  test = "gcm.test",
  controls = NULL,
  alpha = 0.05,
  baseline_fixed = TRUE,
  greedy = FALSE,
  max_size = NULL,
  mandatory = NULL,
  ...
)

CoxphICP(
  formula,
  data,
  env,
  verbose = TRUE,
  type = "residual",
  test = "gcm.test",
  controls = NULL,
  alpha = 0.05,
  baseline_fixed = TRUE,
  greedy = FALSE,
  max_size = NULL,
  mandatory = NULL,
  ...
)

LehmannICP(
  formula,
  data,
  env,
  verbose = TRUE,
  type = "residual",
  test = "gcm.test",
  controls = NULL,
  alpha = 0.05,
  baseline_fixed = TRUE,
  greedy = FALSE,
  max_size = NULL,
```

```
  mandatory = NULL,
  ...
)

LmICP(
  formula,
  data,
  env,
  verbose = TRUE,
  type = "residual",
  test = "gcm.test",
  controls = NULL,
  alpha = 0.05,
  baseline_fixed = TRUE,
  greedy = FALSE,
  max_size = NULL,
  mandatory = NULL,
  ...
)

lmICP(
  formula,
  data,
  env,
  verbose = TRUE,
  type = "residual",
  test = "gcm.test",
  controls = NULL,
  alpha = 0.05,
  baseline_fixed = TRUE,
  greedy = FALSE,
  max_size = NULL,
  mandatory = NULL,
  ...
)

PolrICP(
  formula,
  data,
  env,
  verbose = TRUE,
  type = "residual",
  test = "gcm.test",
  controls = NULL,
  alpha = 0.05,
  baseline_fixed = TRUE,
  greedy = FALSE,
  max_size = NULL,
```

```
  mandatory = NULL,
  ...
)

polrICP(
  formula,
  data,
  env,
  verbose = TRUE,
  type = "residual",
  test = "gcm.test",
  controls = NULL,
  alpha = 0.05,
  baseline_fixed = TRUE,
  greedy = FALSE,
  max_size = NULL,
  mandatory = NULL,
  ...
)

glmICP(
  formula,
  data,
  env,
  verbose = TRUE,
  type = "residual",
  test = "gcm.test",
  controls = NULL,
  alpha = 0.05,
  baseline_fixed = TRUE,
  greedy = FALSE,
  max_size = NULL,
  mandatory = NULL,
  ...
)

cotramICP(
  formula,
  data,
  env,
  verbose = TRUE,
  type = "residual",
  test = "gcm.test",
  controls = NULL,
  alpha = 0.05,
  baseline_fixed = TRUE,
  greedy = FALSE,
  max_size = NULL,
```

```
  mandatory = NULL,
  ...
)

rangerICP(
  formula,
  data,
  env,
  verbose = TRUE,
  type = "residual",
  test = "gcm.test",
  controls = NULL,
  alpha = 0.05,
  baseline_fixed = TRUE,
  greedy = FALSE,
  max_size = NULL,
  mandatory = NULL,
  ...
)

survforestICP(
  formula,
  data,
  env,
  verbose = TRUE,
  type = "residual",
  test = "gcm.test",
  controls = NULL,
  alpha = 0.05,
  baseline_fixed = TRUE,
  greedy = FALSE,
  max_size = NULL,
  mandatory = NULL,
  ...
)

qrfICP(
  formula,
  data,
  env,
  verbose = TRUE,
  type = "residual",
  test = "gcm.test",
  controls = NULL,
  alpha = 0.05,
  baseline_fixed = TRUE,
  greedy = FALSE,
  max_size = NULL,
```

```
  mandatory = NULL,
  ...
)
```

Arguments

formula	A formula including response and covariate terms.
data	A data.frame containing response and explanatory variables.
env	A formula specifying the environment variables (see details).
verbose	Logical, whether output should be verbose (default TRUE).
type	Character, type of invariance ("residual" or "wald"); see Details.
test	Character, specifies the invariance test to be used when type = "residual". The default is "gcm.test". Other implemented tests are "HSIC", "t.test", "var.test", and "combined". Alternatively, a custom function for testing invariance of the form \(r, e, controls) {...} can be supplied, which outputs a list with entry "p.value".
controls	Controls for the used tests and the overall procedure, see dicp_controls.
alpha	Level of invariance test, default 0.05.
baseline_fixed	Fixed baseline transformation, see dicp_controls.
greedy	Logical, whether to perform a greedy version of ICP (default is FALSE).
max_size	Numeric; maximum support size.
mandatory	A formula containing mandatory covariates, i.e., covariates which by domain knowledge are believed to be parents of the response or are in another way required for the environment or model to be valid (for instance, conditionally valid environments or random effects in a mixed model).
...	Further arguments passed to modFUN.

Value

Object of type "dICP". See dicp

Examples

```
set.seed(123)
d <- dgp_dicp(mod = "boxcox", n = 300)
BoxCoxICP(Y ~ X2, data = d, env = ~ E, type = "wald")


set.seed(123)
d <- dgp_dicp(mod = "weibull", n = 300)
SurvregICP(Y ~ X1 + X2 + X3, data = d, env = ~ E)
### or
library("survival")
d$Y <- Surv(d$Y)
survregICP(Y ~ X1 + X2 + X3, data = d, env = ~ E)
CoxphICP(Y ~ X2, data = d, env = ~ E)
coxphICP(Y ~ X2, data = d, env = ~ E)
```

```
set.seed(123)
d <- dgp_dicp(mod = "colr", n = 300)
ColrICP(Y ~ X1 + X2 + X3, data = d, env = ~ E)
```

```
set.seed(123)
d <- dgp_dicp(mod = "coxph", n = 300)
LehmannICP(Y ~ X2, data = d, env = ~ E)
```

```
set.seed(123)
d <- dgp_dicp(mod = "lm", n = 300)
LmICP(Y ~ X1 + X2 + X3, data = d, env = ~ E)
### or
lmICP(Y ~ X1 + X2 + X3, data = d, env = ~ E)
```

```
set.seed(123)
d <- dgp_dicp(mod = "polr", n = 300)
PolrICP(Y ~ X1 + X2 + X3, data = d, env = ~ E)
### or
PolrICP(Y ~ X1 + X2 + X3, data = d, env = ~ E)
```

```
set.seed(123)
d <- dgp_dicp(mod = "binary", n = 300)
glmICP(Y ~ X1 + X2 + X3, data = d, env = ~ E, family = "binomial")
```

```
set.seed(123)
d <- dgp_dicp(mod = "cotram", n = 300)
cotramICP(Y ~ X2, data = d, env = ~ E)
```

```
set.seed(123)
d <- dgp_dicp(mod = "binary", n = 300)
rangerICP(Y ~ X1 + X2 + X3, data = d, env = ~ E)
```

```
set.seed(12)
d <- dgp_dicp(mod = "coxph", n = 3e2)
d$Y <- survival::Surv(d$Y, sample(0:1, 3e2, TRUE, prob = c(0.1, 0.9)))
survforestICP(Y ~ X1 + X2 + X3, data = d, env = ~ E)
```

```
set.seed(12)
d <- dgp_dicp(mod = "boxcox", n = 3e2)
qrfICP(Y ~ X1 + X2 + X3, data = d, env = ~ E)
```

`invariant_sets` *Return invariant sets*

Description

Return invariant sets

Usage

```
invariant_sets(object, with_pvalues = FALSE)
```

Arguments

`object`	Object of class `"dICP"`.
`with_pvalues`	Logical; whether to also return p-values of invariance tests for the non-rejected sets.

Value

Returns vector of all non-rejected sets. With `with_pvalues = TRUE`, a named vector of p-values is returned. Returns named `numeric(0)` if there are no invariant sets.

`pvalues` *Extract set and predictor p-values from tramicp outputs*

Description

Extract set and predictor p-values from tramicp outputs

Usage

```
pvalues(object, which = c("predictor", "set", "all"))
```

Arguments

`object`	Object of class `'dicp'`
`which`	Which p-values to return, `"predictor"` returns p-values for individual predictors, `"set"` for each subset of the predictors, `"all"` returns a list of both

Details

Predictor p-values are computed from the set p-values as follows: For each predictor j as the largest p-value of all sets not containing j.

Value

Numeric vector (or list in case `which = "all"`) of p-values

Examples

```
set.seed(123)
d <- dgp_dicp(n = 1e3, mod = "polr")
res <- polrICP(Y ~ X1 + X2 + X3, data = d, env = ~ E, type = "wald")
pvalues(res, which = "predictor")
pvalues(res, which = "set")
pvalues(res, which = "all")
```

Index