Package 'CCI'

August 19, 2025

Type Package

Title Computational Test for Conditional Independence

Version 0.2.1

Date 2025-07-08

Description Tool for performing computational testing for conditional independence between variables in a dataset. 'CCI' implements permutation in combination with Monte Carlo Cross-Validation in generating null distributions and test statistics. For more details see Computational Test for Conditional Independence (2024) <doi:10.3390/a17080323>.

Imports ggplot2, dplyr, caret, xgboost, ranger, stats, dagitty, data.table, e1071, rlang, progress

Suggests testthat, knitr, rmarkdown

License GPL (>= 2)

URL https://github.com/khliland/CCI

BugReports https://github.com/khliland/CCI/issues

Encoding UTF-8

RoxygenNote 7.3.2

NeedsCompilation no

Author Christian B. H. Thorjussen [aut, cre] (ORCID: <https://orcid.org/0009-0005-5006-6491>), Kristian Hovde Liland [aut] (ORCID: <https://orcid.org/0000-0001-6468-9423>)

Maintainer Christian B. H. Thorjussen <christianbern@gmail.com>

Repository CRAN

Date/Publication 2025-08-19 14:50:25 UTC

Contents

add_interaction_terms *Creates interaction terms for specified variables in a data frame Interaction terms are named as* <var1>_int_<var2> *(e.g.,* Z1_int_Z2 *for the product of* Z1 *and* Z2*).*

Description

Creates interaction terms for specified variables in a data frame Interaction terms are named as <var1>_int_<var2> (e.g., Z1_int_Z2 for the product of Z1 and Z2).

Usage

```
add_interaction_terms(data, Z)
```

Arguments

data	Data frame. The data frame containing the variables for which interaction terms are to be created.
Z	Character vector. The names of the variables for which interaction terms are to be created.

Value

A list with two components:

- data: The modified data frame with added interaction terms.
- new_terms: A character vector of the names of the added interaction terms (e.g., Z1_int_2).

Examples

```
data_generator <-  function(N){
Z1 <- rnorm(N,0,1)
Z2 <- rnorm(N,0,1)
X <- rnorm(N, Z1 + Z2, 1)
Y <- rnorm(N, Z1 + Z2, 1)
df <- data.frame(Z1, Z2, X, Y)
return(df)
}
dat <- data_generator(250)
interaction_terms <- add_interaction_terms(data = dat, Z = c("Z1", "Z2"))
head(interaction_terms$data$Z1_int_Z2)
```

add_poly_terms *Creates polynomial terms for specified variables in a data frame Polynomial terms are named as* <variable>_d_<degree> *(e.g.,* Z1_d_2 *for the square of* Z1*).*

Description

Creates polynomial terms for specified variables in a data frame Polynomial terms are named as <variable>_d_<degree> (e.g., Z1_d_2 for the square of Z1).

Usage

```
add_poly_terms(data, Z, degree = 3, poly = TRUE)
```

Arguments

data	Data frame. The data frame containing the variables for which polynomial terms are to be created.
Z	Character vector. The names of the variables for which polynomial terms are to be created.
degree	Integer. The maximum degree of polynomial terms to be created. Default is 3.
poly	Logical. If TRUE, polynomial terms will be created. If FALSE, no polynomial terms will be created. Default is TRUE.

Value

A list with two components:

- data: The modified data frame with added polynomial terms.
- new_terms: A character vector of the names of the added polynomial terms (e.g., Z1_d_2).

#'

Examples

```
set.seed(123)
data_generator <-  function(N){
Z1 <- rnorm(N,0,1)
Z2 <- rnorm(N,0,1)
X <- rnorm(N, Z1 + Z2, 1)
Y <- rnorm(N, Z1 + Z2, 1)
df <- data.frame(Z1, Z2, X, Y)
return(df)
}
dat <- data_generator(250)
poly_terms <- add_poly_terms(data = dat, Z = c("Z1", "Z2"), degree = 3, poly = TRUE)
print(poly_terms$new_terms)
```

BinaryData *Generate Binary Data*

Description

Creates binary data based on a nonlinear interaction of Z1 and Z2.

Usage

```
BinaryData(N, threshold = 0)
```

Arguments

N	Integer. Sample size.
threshold	Numeric. Threshold for binary classification. Default is 0.

Value

A data frame with columns Z1, Z2, X, and Y.

Examples

```
head(BinaryData(100))
```

BivMultinominal *Generate Bivariate Multinomial Categorical Data*

Description

Creates a multinomial dataset where the probabilities are nonlinear functions of Z1 and Z2.

Usage

```
BivMultinominal(N, zeta = 1.5)
```

Arguments

N	Integer. Sample size.
zeta	Numeric. Strength of interaction. Default is 1.5.

Value

A data frame with columns Z1, Z2, X, and Y (both factors).

BivNonLinearCategorization *Generate Bivariate Nonlinear Categorical Data*

Description

Generates categorical variables X and Y based on nonlinear combinations of Z1 and Z2.

Usage

```
BivNonLinearCategorization(N)
```

Arguments

N	Integer. Sample size.

Value

A data frame with columns Z1, Z2, X, and Y.

build_formula *Build an expanded formula with poly and interaction terms*

Description

Build an expanded formula with poly and interaction terms

Usage

```
build_formula(formula, poly_terms = NULL, interaction_terms = NULL)
```

Arguments

formula	A base formula in the format Y ~ X | Z1 + Z2
poly_terms	Character vector of polynomial term names
interaction_terms	Character vector of interaction term names

Value

A formula object combining all terms

Examples

```
poly_terms <- c("Z1_d_2", "Z2_d_2")
interaction_terms <- c("Z1_int_Z2")
formula <- Y ~ X | Z1 + Z2
final_formula <- build_formula(formula, poly_terms, interaction_terms)
print(final_formula)
```

`CCI.direction` *Choose Direction for testing for the CCI test*

Description

This function selects the best direction for the CCI test based on cross validation. For the condition Y || X | Z, the function return the recommended formula either Y ~ X | Z or X ~ Y | Z .

Usage

```
CCI.direction(
  formula,
  data,
  method = "rf",
  folds = 4,
  nrounds = 600,
  max_depth = 6,
  eta = 0.3,
  gamma = 0,
  colsample_bytree = 1,
  min_child_weight = 1,
  subsample = 1,
  poly = TRUE,
  degree = 3,
  interaction = TRUE,
  verbose = FALSE,
  ...
)
```

Arguments

`formula`	A formula object specifying the model to be fitted.
`data`	A data frame containing the variables specified in the formula.
`method`	A character string specifying the method to be used for model fitting. Options include "rf" (random forest), "xgboost" (XGBoost), "nnet" (neural network), "gpr" (Gaussian process regression), and "svm" (support vector machine).
`folds`	An integer specifying the number of folds for cross-validation. Default is 4.
`nrounds`	Integer. The number of rounds (trees) for methods like xgboost, ranger, and lightgbm. Default is 600.

max_depth	Integer. The maximum depth of the trees for methods like xgboost. Default is 6.
eta	Numeric. The learning rate for methods like xgboost. Default is 0.3.
gamma	Numeric. The minimum loss reduction required to make a further partition on a leaf node of the tree for methods like xgboost. Default is 0.
colsample_bytree	Numeric. The subsample ratio of columns when constructing each tree for methods like xgboost. Default is 1.
min_child_weight	Numeric. The minimum sum of instance weight (hessian) needed in a child for methods like xgboost. Default is 1.
subsample	Numeric. The proportion of the data to be used for subsampling. Default is 1 (no subsampling).
poly	Logical. If TRUE, polynomial terms of the conditioning variables are included in the model. Default is TRUE.
degree	Integer. The degree of polynomial terms to include if poly is TRUE. Default is 3.
interaction	Logical. If TRUE, interaction terms of the conditioning variables are included in the model. Default is TRUE.
verbose	Logical. If TRUE, prints additional information during the execution. Default is FALSE.
...	Additional arguments to be passed to the model fitting function.

Value

A formula object specifying the selected model direction.

`CCI.pretuner` *CCI tuner function for CCI test*

Description

The `CCI.tuner` function performs a grid search over parameters for a conditional independence test using machine learning model supported by CCI.test. The tuner use the caret package for tuning.

Usage

```
CCI.pretuner(
  formula,
  data,
  method = "rf",
  metric = "RMSE",
  validation_method = "cv",
  folds = 4,
  training_share = 0.7,
```

```
  tune_length = 4,
  random_grid = TRUE,
  samples = 35,
  poly = TRUE,
  degree = 3,
  interaction = TRUE,
  verboseIter = FALSE,
  include_explanatory = FALSE,
  verbose = FALSE,
  parallel = FALSE,
  mtry = 1:10,
  nrounds = c(100, 200, 300, 400, 500, 600, 700, 800, 900, 1000),
  eta = seq(0.01, 0.3, by = 0.05),
  max_depth = 2:6,
  gamma = c(0, 1, 2, 3),
  colsample_bytree = c(0.8, 0.9, 1),
  min_child_weight = c(1, 3),
  subsample = 1,
  sigma = seq(0.1, 2, by = 0.3),
  C = seq(0.1, 2, by = 0.5),
  ...
)
```

Arguments

formula	Model formula specifying the relationship between dependent and independent variables.
data	A data frame containing the variables specified in the formula.
method	Character. Specifies the machine learning method to use. Supported methods are random forest "rf", extreme gradient boosting "xgboost" and Support Vector Machine "svm".
metric	Character. The performance metric to optimize during tuning. Default is "RMSE".
validation_method	Character. Specifies the resampling method. Default is "cv".
folds	Integer. The number of folds for cross-validation during the tuning process. Default is 10.
training_share	Numeric. For leave-group out cross-validation: the training percentage. Default is 0.7.
tune_length	Integer. The number of parameter combinations to try during the tuning process. Default is 10.
random_grid	Logical. If TRUE, a random grid search is performed. If FALSE, a full grid search is performed. Default is TRUE.
samples	Integer. The number of random samples to take from the grid. Default is 30.
poly	Logical. If TRUE, polynomial terms of the conditional variables are included in the model. Default is TRUE.

degree	Integer. The degree of polynomial terms to include if poly is TRUE. Default is 3.
interaction	Logical. If TRUE, interaction terms of the conditional variables are included in the model. Default is TRUE.
verboseIter	Logical. If TRUE, the function will print the tuning process. Default is FALSE.
include_explanatory	Logical. If TRUE, given the condition Y || X | Z, the function will include explanatory variable X in the model for Y. Default is FALSE
verbose	Logical. If TRUE, the function will print the tuning process. Default is FALSE..
parallel	Logical. If TRUE, the function will use parallel processing. Default is TRUE.
mtry	Integer. The number of variables randomly sampled as candidates at each split for random forest. Default is 1:5.
nrounds	Integer. The number of rounds (trees) for methods such as xgboost and random forest. Default is seq(50, 200, by = 25).
eta	Numeric. The learning rate for xgboost. Default is seq(0.01, 0.3, by = 0.05).
max_depth	Integer. The maximum depth of the tree for xgboost. Default is 1:6.
gamma	Numeric. The minimum loss reduction required to make a further partition on a leaf node for xgboost. Default is seq(0, 5, by = 1).
colsample_bytree	Numeric. The subsample ratio of columns when constructing each tree for xgboost. Default is seq(0.5, 1, by = 0.1).
min_child_weight	Integer. The minimum sum of instance weight (hessian) needed in a child for xgboost. Default is 1:5.
subsample	Numeric. The subsample ratio of the training. Default is 1.
sigma	Numeric. The standard deviation of the Gaussian kernel for Gaussian Process Regression. Default is seq(0.1, 2, by = 0.3).
C	Numeric. The regularization parameter for Support Vector Machine. Default is seq(0.1, 2, by = 0.5).
...	Additional arguments to pass to the CCI.tuner function.

Value

A list containing:

- best_param: A data frame with the best parameters.
- tuning_result: A data frame with all tested parameter combinations and their performance metrics.
- warnings: A character vector of warnings issued during tuning.

See Also

CCI.test perm.test, print.summary.CCI, plot.CCI, QQplot

Examples

```
set.seed(123)
data <- data.frame(x1 = rnorm(100), x2 = rnorm(100), x3 = rnorm(100), y = rnorm(100))
# Tune random forest parameters
result <- CCI.pretuner(formula = y ~ x1 | x2 + x3,
data = data,
samples = 5,
folds = 3,
method = "rf")
```

`CCI.test` *Computational test for conditional independence based on ML and Monte Carlo Cross Validation*

Description

The `CCI.test` function performs a conditional independence test using a specified machine learning model or a custom model provided by the user. It calculates the test statistic, generates a null distribution via permutations, computes p-values, and optionally generates a plot of the null distribution with the observed test statistic. The 'CCI.test' function serves as a wrapper around the 'perm.test' function

Usage

```
CCI.test(
  formula = NULL,
  data,
  plot = TRUE,
  p = 0.5,
  nperm = 60,
  nrounds = 600,
  dag = NULL,
  dag_n = 1,
  metric = "Auto",
  method = "rf",
  choose_direction = FALSE,
  print_result = TRUE,
  parametric = FALSE,
  poly = TRUE,
  degree = 3,
  subsample = 1,
  min_child_weight = 1,
  colsample_bytree = 1,
  eta = 0.3,
  gamma = 0,
  max_depth = 6,
  num_class = NULL,
```

```
  interaction = TRUE,
  metricfunc = NULL,
  mlfunc = NULL,
  tail = NA,
  tune = FALSE,
  samples = 35,
  folds = 5,
  tune_length = 10,
  seed = NA,
  random_grid = TRUE,
  nthread = 1,
  verbose = FALSE,
  progress = TRUE,
  ...
)
```

Arguments

formula	Model formula or a DAGitty object specifying the relationship between dependent and independent variables.
data	A data frame containing the variables specified in the formula.
plot	Logical, indicating if a plot of the null distribution with the test statistic should be generated. Default is TRUE.
p	Numeric. Proportion of data used for training the model. Default is 0.5.
nperm	Integer. The number of permutations to perform. Default is 600.
nrounds	Integer. The number of rounds (trees) for methods 'xgboost' and 'rf' Default is 600.
dag	An optional DAGitty object for specifying a Directed Acyclic Graph (DAG) to use for conditional independence testing. Default is NA.
dag_n	Integer. If a DAGitty object is provided, specifies which conditional independence test to perform. Default is 1.
metric	Character. Specifies the type of data: "Auto", "RMSE" or "Kappa". Default is "Auto".
method	Character. Specifies the machine learning method to use. Supported methods include generlaized linear models "lm", random forest "rf", and extreme gradient boosting "xgboost", etc. Default is "rf".#'
choose_direction	Logical. If TRUE, the function will choose the best direction for testing. Default is FALSE.
print_result	Logical. If TRUE, the function will print the result of the test. Default is TRUE.
parametric	Logical, indicating whether to compute a parametric p-value instead of the empirical p-value. A parametric p-value assumes that the null distribution is gaussian. Default is FALSE.
poly	Logical. If TRUE, polynomial terms of the conditional variables are included in the model. Default is TRUE.

degree	Integer. The degree of polynomial terms to include if poly is TRUE. Default is 3.
subsample	Numeric. The proportion of data to use for subsampling. Default is 1 (no subsampling).
min_child_weight	Numeric. The minimum sum of instance weight (hessian) needed in a child for methods like xgboost. Default is 1.
colsample_bytree	Numeric. The subsample ratio of columns when constructing each tree for methods like xgboost. Default is 1.
eta	Numeric. The learning rate for methods like xgboost. Default is 0.3.
gamma	Numeric. The minimum loss reduction required to make a further partition on a leaf node of the tree for methods like xgboost. Default is 0.
max_depth	Integer. The maximum depth of the trees for methods like xgboost. Default is 6.
num_class	Integer. The number of classes for categorical data (used in xgboost). Default is NULL.
interaction	Logical. If TRUE, interaction terms of the conditional variables are included in the model. Default is TRUE.
metricfunc	Optional the user can pass a custom function for calculating a performance metric based on the model's predictions. Default is NULL.
mlfunc	Optional the user can pass a custom machine learning wrapper function to use instead of the predefined methods. Default is NULL.
tail	Character. Specifies whether to calculate left-tailed or right-tailed p-values, depending on the performance metric used. Only applicable if using `metricfunc` or `mlfunc`. Default is NA.
tune	Logical. If TRUE, the function will perform hyperparameter tuning for the specified machine learning method. Default is FALSE.
samples	Integer. The number of samples to use for tuning. Default is 35.
folds	Integer. The number of folds for cross-validation during the tuning process. Default is 5.
tune_length	Integer. The number of parameter combinations to try during the tuning process. Default is 10.
seed	Integer. The seed for tuning. Default is NA.
random_grid	Logical. If TRUE, a random grid search is performed. If FALSE, a full grid search is performed. Default is TRUE.
nthread	Integer. The number of threads to use for parallel processing. Default is 1.
verbose	Logical. If TRUE, additional information is printed during the execution of the function. Default is FALSE.
progress	Logical. If TRUE, a progress bar is displayed during the permutation process. Default is TRUE.
...	Additional arguments to pass to the `perm.test` function.

Value

Invisibly returns the result of `perm.test`, which is an object of class 'CCI' containing the null distribution, observed test statistic, p-values, the machine learning model used, and the data.

See Also

`perm.test`, `print.summary.CCI`, `plot.CCI`, `CCI.pretuner`, `QQplot`

Examples

```
set.seed(123)
data <- data.frame(x1 = stats::rnorm(100), x2 = stats::rnorm(100), y = stats::rnorm(100))
result <- CCI.test(y ~ x1 | x2, data = data, nperm = 25, interaction = FALSE)
summary(result)
```

---

check_formula *Check the formula statement*

Description

This function verifies that all variables specified in the formula are present in the provided data frame. If any variables are missing, the function will stop and return an error message listing the missing variables.

Usage

```
check_formula(formula, data)
```

Arguments

`formula`	Formula. The model formula that specifies the relationship between the dependent and independent variables.
`data`	Data frame. The data frame in which to check for the presence of variables specified in the formula.

Value

Invisibly returns `NULL` if all variables are present. Stops with an error if any variables are missing.

clean_formula *Clean and Reformat Formula String*

Description

This function processes and reformats formula string to ensure it is in the correct format for conditional independence testing. The function checks if the formula uses the '+' operator for additive models and transforms it into a format that includes a conditioning variable separated by '|'.

Usage

```
clean_formula(formula)
```

Arguments

formula	Formula. The model formula that specifies the relationship between the dependent and independent variables, and potentially the conditioning variables. The formula is expected to follow the format `Y ~ X + Z1 + Z2` or `Y ~ X | Z1 + Z2`.

Value

A reformatted formula in the correct format for conditional independence testing. The returned formula will either retain the original format or be transformed to include conditioning variables.

Examples

```
clean_formula(y ~ x | z + v)
clean_formula(y ~ x + z + v)
# Error: The formula is not of the right format
try(clean_formula(y ~ x))
```

ComplexCategorization *Generate Complex Categorical Data*

Description

A more intricate categorization based on combinations of Z1 and Z2.

Usage

```
ComplexCategorization(N)
```

Arguments

N	Integer. Sample size.

Value

A data frame with columns Z1, Z2, X, and Y.

Examples

```
head(ComplexCategorization(100))
```

ExpLogData *Generate Categorical Data Based on Exponential and Logarithmic Functions*

Description

Categorizes based on thresholds of exponential and logarithmic transformations of Z1 and Z2.

Usage

```
ExpLogData(N)
```

Arguments

N	Integer. Sample size.

Value

A data frame with columns Z1, Z2, X, and Y.

ExpLogThreshold *Generate Exponential and Logarithmic Data*

Description

Generates data with exponential and logarithmic dependencies based on Z1 and Z2.

Usage

```
ExpLogThreshold(N)
```

Arguments

N	Integer. Sample size.

Value

A data frame with columns Z1, Z2, X, and Y.

Examples

```
head(ExpLogThreshold(100))
```

ExponentialNoise *Generate Data with Exponential Noise*

Description

Adds exponential noise to a nonlinear combination of Z1 and Z2.

Usage

```
ExponentialNoise(N, rate_param = 1)
```

Arguments

N	Integer. Sample size.
rate_param	Numeric. Rate parameter for the exponential distribution. Default is 1.

Value

A data frame with columns Z1, Z2, X, and Y.

Examples

```
head(ExponentialNoise(100))
```

---

get_pvalues *P-value Calculation Based on Null Distribution and Test Statistic*

Description

This function calculates p-values based on the comparison of a test statistic against a null distribution. It can perform either empirical or parametric p-value calculations and supports both left-tailed and right-tailed tests.

Usage

```
get_pvalues(
  dist,
  test_statistic,
  parametric = FALSE,
  tail = c("left", "right")
)
```

Arguments

dist	Numeric vector. Represents the null distribution of the test statistic.
test_statistic	Numeric. The observed test statistic for which the p-value is to be calculated.
parametric	Logical. If TRUE, calculates parametric p-values assuming the null distribution is normal. If FALSE, calculates empirical p-values. Default is FALSE.
tail	Character. Specifies whether to calculate left-tailed or right-tailed p-values. Must be either "left" or "right". Default is "left".

Value

Numeric. The calculated p-value.

Examples

```
set.seed(123)
null_dist <- rnorm(1000)
observed_stat <- 1.5
p_value <- get_pvalues(null_dist, observed_stat, parametric = FALSE, tail = "right")
print(p_value)
```

---

get_tuned_params *Get the best parameters after tuning with CCI.tuner*

Description

Get the best parameters after tuning with CCI.tuner

Usage

```
get_tuned_params(tuned_model)
```

Arguments

tuned_model	A model object returned from the CCI.pretuner function. This object contains the tuned parameters and other relevant information.

Value

A named list of tuned parameters specific to the model method (e.g., `mtry` for random forest, `eta`, `max_depth` for xgboost). Returns `NULL` for unsupported methods.

GridPartition *Generate Grid Partitioned Data*

Description

Generates data with a grid partitioning effect based on Z1 and Z2.

Usage

```
GridPartition(N)
```

Arguments

N	Integer. Sample size.

Value

A data frame with columns Z1, Z2, X, and Y.

Examples

```
head(GridPartition(100))
```

HardCase *Generate Hard Case Data with Two Z Variables*

Description

Generates data with a hard case scenario where X and Y are influenced by two Z variables in a nonlinear manner.

Usage

```
HardCase(N)
```

Arguments

N	Integer. Sample size.

Value

A data frame with columns X, Y, Z1, and Z2.

Examples

```
head(HardCase(100))
```

InteractiondData *Generate Categorical Data Based on Interactions*

Description

Creates categorical X and Y variables based on the interaction of signs and sums of Z1 and Z2.

Usage

```
InteractiondData(N)
```

Arguments

N	Integer. Sample size.

Value

A data frame with columns Z1, Z2, X, and Y.

NonLinearCategorization *Generate Nonlinear Categorical Data (Univariate)*

Description

Generates a dataset with a single Z influencing categorical X and Y.

Usage

```
NonLinearCategorization(N, d = 0)
```

Arguments

N	Integer. Sample size.
d	Numeric. Dependency strength. Default is 0.

Value

A data frame with columns Z, X, and Y.

NonLinearData *Generate Nonlinear Categorical Data (Bivariate)*

Description

Creates categorical X and Y variables based on sinusoidal and cosine functions of Z1 and Z2.

Usage

```
NonLinearData(N)
```

Arguments

N	Integer. Sample size.

Value

A data frame with columns Z1, Z2, X, and Y.

NonLinNormal *Generate Nonlinear Normal Data*

Description

Creates nonlinear continuous data based on an exponential interaction of Z1 and Z2.

Usage

```
NonLinNormal(N)
```

Arguments

N	Integer. Sample size.

Value

A data frame with columns Z1, Z2, X, and Y.

Examples

```
head(NonLinNormal(N = 100))
```

NonLinNormalZs *Generate High-dimensional Nonlinear Normal Data*

Description

Creates a Z-dimensional nonlinear dataset with complex dependencies between features and targets.

Usage

```
NonLinNormalZs(N, d = 0, Zs = 20)
```

Arguments

N	Integer. Sample size.
d	Numeric. Dependency strength. Default is 0.
Zs	Integer. Number of Z variables. Default is 10.

Value

A data frame with columns Z1-Z10, X, and Y.

Examples

```
head(NonLinNormalZs(N = 100, Zs  = 20))
```

NormalData *Generate Normal Data for Conditional Independence Testing*

Description

This function generates continuous data where X and Y are both functions of Z1 and Z2 with added normal noise.

Usage

```
NormalData(N)
```

Arguments

N	Integer. Sample size.

Value

A data frame with columns Z1, Z2, X, and Y.

perm.test *Permutation Test for Conditional Independence*

Description

Permutation Test for Conditional Independence

Usage

```
perm.test(
  formula,
  data,
  p = 0.7,
  nperm = 600,
  subsample = 1,
  metric = "RMSE",
  method = "rf",
  nrounds = 120,
  parametric = FALSE,
  poly = TRUE,
  interaction = TRUE,
  degree = 3,
  tail = NA,
  metricfunc = NULL,
  mlfunc = NULL,
  nthread = 1,
  dag = NA,
  dag_n = NA,
  num_class = NULL,
  progress = TRUE,
  ...
)
```

Arguments

formula	Model formula or DAGitty object specifying the relationship between dependent and independent variables.
data	A data frame containing the variables specified in the formula.
p	Proportion of data to use for training the model. Default is 0.825.
nperm	Number of permutations to perform. Default is 500.
subsample	The proportion of the data to be used. Default is 1 (no subsampling).
metric	Type of metric: "RMSE", "Kappa" or "Custom". Default is 'RMSE'.
method	The machine learning method to use. Supported methods include "rf", "xgboost", etc. Default is "rf".

nrounds	Number of rounds (trees) for methods such as xgboost and random forest. Default is 120.
parametric	Logical. If TRUE, a parametric p-value is calculated in addition to the empirical p-value. Default is FALSE.
poly	Logical. If TRUE, polynomial terms of the conditional variables are included in the model. Default is TRUE.
interaction	Logical. If TRUE, interaction terms of the conditional variables are included in the model. Default is TRUE.
degree	The degree of polynomial terms to include if poly is TRUE. Default is 3.
tail	Specifies whether the test is one-tailed ("left" or "right") or two-tailed. Default is NA.
metricfunc	An optional custom function to calculate the performance metric based on the model's predictions. Default is NULL.
mlfunc	An optional custom machine learning function to use instead of the predefined methods. Default is NULL.
nthread	Integer. The number of threads to use for parallel processing. Default is 1.
dag	A DAGitty object specifying the directed acyclic graph for the variables. Default is NA.
dag_n	A character string specifying the name of the node in the DAGitty object to be used for conditional independence testing. Default is NA.
num_class	Integer. The number of classes for categorical data (used in xgboost). Default is NULL.
progress	Logical. If TRUE, a progress bar is displayed during the permutation process. Default is TRUE.
...	Additional arguments to pass to the machine learning model fitting function.

Value

An object of class 'CCI' containing the null distribution, observed test statistic, p-values, the machine learning model used, and the data.

See Also

print.CCI, summary.CCI, plot.CCI, QQplot

Examples

```
set.seed(123)
dat <- data.frame(x1 = rnorm(100),
x2 = rnorm(100),
x3 = rnorm(100),
x4 = rnorm(100),
y = rnorm(100))
perm.test(y ~ x1 | x2 + x3 + x4, data = dat, nperm = 25)
```

plot.CCI *Plot for CCI testing*

Description

Plot for CCI testing

Usage

```
## S3 method for class 'CCI'
plot(
  x,
  fill_color = "lightblue",
  axis.text.x = 13,
  axis.text.y = 13,
  strip.text.x = 13,
  strip.text.y = 13,
  legend.text = 13,
  legend.title = 13,
  ...
)
```

Arguments

x	Object of class 'CCI'
fill_color	Color for the histogram fill
axis.text.x	Size of x-axis text
axis.text.y	Size of y-axis text
strip.text.x	Size of x-axis strip text
strip.text.y	Size of y-axis strip text
legend.text	Size of legend text
legend.title	Size of legend title
...	Additional arguments to ggplot2

Value

A plot of the null distribution and the test statistic in ggplot2 format.

See Also

print.CCI, summary.CCI, plot.CCI, perm.test

Examples

```
dat <- data.frame(x1 = rnorm(100), x2 = rnorm(100), y = rnorm(100))
cci <- CCI.test(y ~ x1 + x2, data = dat, interaction = FALSE)
plot(cci)
```

PoissonNoise *Generate Data with Poisson Noise*

Description

Adds Poisson noise to a nonlinear combination of Z1 and Z2.

Usage

```
PoissonNoise(N, lambda = 1)
```

Arguments

N	Integer. Sample size.
lambda	Numeric. Rate parameter for the Poisson distribution. Default is 1.

Value

A data frame with columns Z1, Z2, X, and Y.

Examples

```
head(PoissonNoise(100))
```

PolyData *Generate Categorical Polynomial Data*

Description

Generates X and Y categories based on polynomial combinations of Z1 and Z2.

Usage

```
PolyData(N)
```

Arguments

N	Integer. Sample size.

Value

A data frame with columns Z1, Z2, X, and Y.

PolyDecision *Generate Polynomial Decision Boundary Data*

Description

Generates data with a polynomial decision boundary based on Z1 and Z2.

Usage

```
PolyDecision(N)
```

Arguments

N	Integer. Sample size.

Value

A data frame with columns Z1, Z2, X, and Y.

Examples

```
head(PolyDecision(100))
```

print.summary.CCI *Print and summary methods for the CCI class*

Description

Print and summary methods for the CCI class

Usage

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

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

Arguments

x	Object of class 'CCI'
...	Additional arguments to print/summary
object	Object of class 'CCI'

Value

The print methods have no return value, the summary methods return an object of class 'summary.CCI'.

See Also

perm.test, plot.CCI, QQplot

`QQplot` *QQ-plot for multiple testing in CCI*

Description

QQ-plot for multiple testing in CCI

Usage

```
QQplot(
  object,
  axis.text.x = 17,
  axis.text.y = 17,
  strip.text.x = 17,
  strip.text.y = 17,
  legend.text = 17,
  legend.title = 17,
  ...
)
```

Arguments

`object`	Object of class 'CCI'
`axis.text.x`	Size of x-axis text
`axis.text.y`	Size of y-axis text
`strip.text.x`	Size of x-axis strip text
`strip.text.y`	Size of y-axis strip text
`legend.text`	Size of legend text
`legend.title`	Size of legend title
`...`	Additional arguments to pass to the `test.gen` function.

Value

A QQ-plot of the p-values in ggplot2 format.

See Also

print.CCI, summary.CCI, plot.CCI, perm.test

Examples

```
dat <- data.frame(x1 = rnorm(100), x2 = rnorm(100), y = rnorm(100))
cci <- CCI.test(y ~ x1 | x2,
data = dat,
nperm = 25,
interaction = FALSE)
QQplot(cci)
```

QuadThresh *Generate Quadratic Threshold Data*

Description

Generates data with a quadratic threshold effect based on Z1 and Z2.

Usage

```
QuadThresh(N)
```

Arguments

N	Integer. Sample size.

Value

A data frame with columns Z1, Z2, X, and Y.

Examples

```
head(QuadThresh(100))
```

SinCosThreshold *Generate Sinusoidal and Cosine Data*

Description

Generates data with sinusoidal and cosine dependencies based on Z1 and Z2.

Usage

```
SinCosThreshold(N)
```

Arguments

N	Integer. Sample size.

Value

A data frame with columns Z1, Z2, X, and Y.

Examples

```
head(SinCosThreshold(100))
```

SineGaussian *Generate Sine-Gaussian Data (Univariate)*

Description

This function generates data with a nonlinear sinusoidal dependency based on a Gaussian density envelope.

Usage

```
SineGaussian(N, a = 1, d = 0)
```

Arguments

N	Integer. Sample size.
a	Numeric. Frequency parameter of the sine function. Default is 1.
d	Numeric. Strength of dependency between X and Y. Default is 0.

Value

A data frame with columns Z, X, and Y.

SineGaussianBiv *Generate Sine-Gaussian Data (Bivariate)*

Description

This function generates bivariate data with nonlinear dependencies based on a Gaussian density envelope and sinusoidal functions.

Usage

```
SineGaussianBiv(N, a = 1, d = 0)
```

Arguments

N	Integer. Sample size.
a	Numeric. Frequency parameter for the sine function. Default is 1.
d	Numeric. Strength of dependency between X and Y. Default is 0.

Value

A data frame with columns Z1, Z2, X, and Y.

SineGaussianNoise *Generate Sine-Gaussian Data (Bivariate)*

Description

This function generates bivariate data with nonlinear dependencies based on a Gaussian density envelope and sinusoidal functions.

Usage

```
SineGaussianNoise(N, a = 1, d = 0)
```

Arguments

N	Integer. Sample size.
a	Numeric. Frequency parameter for the sine function. Default is 1.
d	Numeric. Strength of dependency between X and Y. Default is 0.

Value

A data frame with columns Z1, Z2, X, and Y.

---

test.gen *Generate the Test Statistic or Null Distribution Using Permutation*

Description

This function generates the test statistic or a null distribution through permutation for conditional independence testing. It supports various machine learning methods, including random forests, extreme gradient boosting, and allows for custom metric functions and model fitting functions.

Usage

```
test.gen(
  formula,
  data,
  method = "rf",
  metric,
  nperm = 60,
  subsample = 1,
  p = 0.8,
  poly = TRUE,
```

```
  interaction = TRUE,
  degree = 3,
  nrounds = 600,
  nthread = 1,
  permutation = FALSE,
  metricfunc = NULL,
  mlfunc = NULL,
  num_class = NULL,
  progress = TRUE,
  ...
)
```

Arguments

formula	Formula specifying the relationship between dependent and independent variables.
data	Data frame. The data containing the variables used.
method	Character. The modeling method to be used. Options include "xgboost" for gradient boosting, or "rf" for random forests or '"svm" for Support Vector Machine.
metric	Character. The type of metric: can be "RMSE", "Kappa" or "Custom. Default is 'RMSE'
nperm	Integer. The number of generated Monte Carlo samples. Default is 60.
subsample	Numeric. The proportion of the data to be used for subsampling. Default is 1 (no subsampling).
p	Numeric. The proportion of the data to be used for training. The remaining data will be used for testing. Default is 0.8.
poly	Logical. Whether to include polynomial terms of the conditioning variables. Default is TRUE.
interaction	Logical. Whether to include interaction terms of the conditioning variables. Default is TRUE.
degree	Integer. The degree of polynomial terms to be included if poly is TRUE. Default is 3.
nrounds	Integer. The number of rounds (trees) for methods like xgboost, ranger, and lightgbm. Default is 500.
nthread	Integer. The number of threads to use for parallel processing. Default is 1.
permutation	Logical. Whether to perform permutation to generate a null distribution. Default is FALSE.
metricfunc	Function. A custom metric function provided by the user. The function must take arguments: data, model, test_indices, and test_matrix, and return a single value performance metric. Default is NULL.
mlfunc	Function. A custom machine learning function provided by the user. The function must have the arguments: formula, data, train_indices, test_indices, and . . . , and return a single value performance metric. Default is NULL.
num_class	Integer. The number of classes for categorical data (used in xgboost and lightgbm). Default is NULL.

progress	Function. A logical value indicating whether to show a progress bar during the permutation process. Default is TRUE.
...	Additional arguments to pass to the machine learning wrapper functions xgboost_wrapper, ranger_wrapper, lightgbm_wrapper, or to a custom-built wrapper function.

Value

A list containing the test distribution.

Examples

```
set.seed(123)
data <- data.frame(x1 = rnorm(100),
x2 = rnorm(100),
x3 = rnorm(100),
x4 = rnorm(100),
y = rnorm(100))
result <- test.gen(formula = y ~ x1 | x2 + x3 + x4,
                   metric = "RMSE",
                   data = data)
hist(result$distribution)
```

TrigData *Generate Categorical Trigonometric Data*

Description

Uses sine and cosine functions of Z1 and Z2 to generate categorical outcomes.

Usage

```
TrigData(N)
```

Arguments

N	Integer. Sample size.

Value

A data frame with columns Z1, Z2, X, and Y.

UniformNoise *Generate Data with Uniform Noise*

Description

Adds uniform noise to a nonlinear combination of Z1 and Z2.

Usage

```
UniformNoise(N)
```

Arguments

N	Integer. Sample size.

Value

A data frame with columns Z1, Z2, X, and Y.

Examples

```
head(UniformNoise(100))
```

wrapper_ranger *Random Forest wrapper for CCI*

Description

Random Forest wrapper for CCI

Usage

```
wrapper_ranger(
  formula,
  data,
  train_indices,
  test_indices,
  metric,
  metricfunc = NULL,
  nthread = 1,
  ...
)
```

Arguments

formula	Model formula specifying the dependent and independent variables.
data	Data frame containing the dataset to be used for training and testing the model.
train_indices	A vector of indices specifying the rows in data to be used as the training set.
test_indices	A vector of indices specifying the rows in data to be used as the test set.
metric	Character string indicating the type of performance metric. Can be "RMSE" for regression, "Kappa" for binary classification, or multiclass classification.
metricfunc	Optional user-defined function to calculate a custom performance metric. This function should take the arguments data, model, and test_indices, and return a numeric value representing the performance metric.
nthread	Integer. The number of threads to use for parallel processing. Default is 1.
...	Additional arguments passed to the ranger function.

Value

A numeric value representing the performance metric of the model on the test set.

---

wrapper_svm *SVM wrapper for CCI*

Description

SVM wrapper for CCI

Usage

```
wrapper_svm(
  formula,
  data,
  train_indices,
  test_indices,
  metric,
  metricfunc = NULL,
  ...
)
```

Arguments

formula	Model formula
data	Data frame
train_indices	Indices for training data
test_indices	Indices for testing data
metric	Type of metric ("RMSE" or "Kappa")
metricfunc	Optional user-defined function to calculate a custom performance metric.
...	Additional arguments passed to e1071::svm

Value

Performance metric (RMSE for continuous, Kappa for classification)

`wrapper_xgboost` *Extreme Gradient Boosting wrapper for CCI*

Description

Extreme Gradient Boosting wrapper for CCI

Usage

```
wrapper_xgboost(
  formula,
  data,
  train_indices,
  test_indices,
  metric,
  nrounds = 500,
  metricfunc = NULL,
  nthread = 1,
  num_class = NULL,
  subsample = 1,
  ...
)
```

Arguments

`formula`	Model formula
`data`	Data frame
`train_indices`	Indices for training data
`test_indices`	Indices for training data
`metric`	Type of performance metric
`nrounds`	Number of boosting rounds
`metricfunc`	A user specific metric function which have the arguments data, model test_indices and test_matrix and returns a numeric value
`nthread`	Integer. Number of threads to use for parallel computation during model training in XGBoost. Default is 1.
`num_class`	Number of categorical classes
`subsample`	Proportion of the data to be used. Default is 1 (no subsampling).
`...`	Additional arguments passed to xgb.train

Value

Performance metric

Index