Package 'CJAMP'

March 20, 2019

Type Package

Title Copula-Based Joint Analysis of Multiple Phenotypes

Version 0.1.1

Description We provide a computationally efficient and robust implementation of the recently proposed C-JAMP (Copula-based Joint Analysis of Multiple Phenotypes) method (Konigorski et al., 2019, submitted). C-JAMP allows estimating and testing the association of one or multiple predictors on multiple outcomes in a joint model, and is implemented here with a focus on large-scale genome-wide association studies with two phenotypes. The use of copula functions allows modeling a wide range of multivariate dependencies between the phenotypes, and previous results are supporting that C-JAMP can increase the power of association studies to identify associated genetic variants in comparison to existing methods (Konigorski, Yilmaz, Pischon, 2016, <DOI:10.1186/s12919-016-0045-6>; Konigorski, Yilmaz, Bull, 2014, <DOI:10.1186/1753-6561-8-S1-S72>). In addition to the C-JAMP functions, functions are available to generate genetic and phenotypic data, to compute the minor allele frequency (MAF) of genetic markers, and to estimate the phenotypic variance explained by genetic markers.

License GPL-2

Encoding UTF-8

LazyData TRUE

Imports stats, optimx

Suggests MASS, knitr, rmarkdown

VignetteBuilder knitr

RoxygenNote 6.1.1

NeedsCompilation no

Author Stefan Konigorski [aut, cre],
Yildiz E. Yilmaz [ctb, ths]

Maintainer Stefan Konigorski <stefan.konigorski@gmail.com>

Repository CRAN

Date/Publication 2019-03-20 22:23:33 UTC

R topics documented:

`cjamp` *C-JAMP: Copula-based joint analysis of multiple phenotypes.*

Description

Functions to perform C-JAMP: `cjamp` fits a joint model of two phenotypes conditional on one or multiple predictors; `cjamp_loop` uses `cjamp` to fit the same copula model separately for a list of multiple predictors, e.g. for a genetic association study.

Usage

```
cjamp(copula = "Clayton", Y1 = NULL, Y2 = NULL,
  predictors_Y1 = NULL, predictors_Y2 = NULL, scale_var = FALSE,
  optim_method = "BFGS", trace = 0, kkt2tol = 1e-16, SE_est = TRUE,
  pval_est = TRUE, n_iter_max = 10)

cjamp_loop(copula = "Clayton", Y1 = NULL, Y2 = NULL,
  predictors = NULL, covariates_Y1 = NULL, covariates_Y2 = NULL,
  scale_var = FALSE, optim_method = "BFGS", trace = 0,
  kkt2tol = 1e-16, SE_est = TRUE, pval_est = TRUE, n_iter_max = 10)
```

Arguments

`copula`	String indicating whether the joint model will be computed under the Clayton (`"Clayton"`) or 2-parameter copula (`"2param"`) model.
`Y1`	Numeric vector containing the first phenotype.
`Y2`	Numeric vector containing the second phenotype.
`predictors_Y1`	Dataframe containing the predictors of `Y1` in columns (for the `cjamp` function).
`predictors_Y2`	Dataframe containing the predictors of `Y2` in columns (for the `cjamp` function).
`scale_var`	Logical. Indicating whether all predictors will be centered and scaled before the analysis or not (default: FALSE).

optim_method — String passed to the optimx function. It specifies the optimization method in the optimx function that is used for maximizing the log-likelihood function. Recommended is the "BFGS" optimization method (default). For further methods, see the description of the optimx function.

trace — Integer passed to the optimx function. It specifies the tracing information on the progress of the optimization. The value 2 gives full tracing, default value 0 blocks all details. See also the optimx documentation.

kkt2tol — Numeric. Passed to the optimx function, default value is 1E-16. It specifies the tolerance for the eigenvalue ratio in the Karush-Kuhn-Tucker (KKT) test for a positive definite Hessian matrix. See also the optimx documentation.

SE_est — Logical indicator whether standard error estimates are computed for the parameters using the inverse of the observed information matrix (TRUE, default), or whether standard error estimates are not computed (FALSE).

pval_est — Logical indicator whether p-values are computed from hypothesis tests of the absence of effects of each predictor on each phenotype in the marginal models (TRUE, default), or whether p-values are not computed (FALSE). P-values are obtained from large sample Wald-type tests based on the maximum likelihood parameter estimates and the standard error estimates.

n_iter_max — Integer indicating the maximum number of optimization attempts of the log-likelihood function with different starting values, if the optimization doesn't converge (default: 10).

predictors — Dataframe containing the predictors of Y1 and Y2 in columns for which estimates are returned (for the cjamp_loop function).

covariates_Y1 — Dataframe containing the covariates of Y1 in columns for which estimates are not returned (for the cjamp_loop function).

covariates_Y2 — Dataframe containing the covariates of Y2 in columns for which estimates are not returned (for the cjamp_loop function).

Details

Both functions cjamp and cjamp_loop fit a joint copula model of two phenotypes using the Clayton copula or 2-parameter copula, conditional on none or multiple predictors and covariates in the marginal models. The optimx function of the optimx package is used to maximize the log-likelihood (i.e. to minimize the minus log-likelihood function minusloglik) to obtain maximum likelihood coefficient estimates of all parameters. For this, the BFGS optimization method is recommended. Standard error estimates of the coefficient estimates can be obtained by using the observed inverse information matrix. P-values from hypothesis tests of the absence of effects of each predictor on each phenotype in the marginal models can be obtained from large-sample Wald-type tests (see the vignette for details).

It should be noted that cjamp, cjamp_loop and minusloglik) assume quantitative predictors and use an additive model, i.e. for categorical predictors with more than 2 levels, dummy variables have to be created beforehand. Accordingly, if single nucleotide variants (SNVs) are included as predictors, the computation is based on an additive genetic model if SNVs are provided as 0-1-2 genotypes and on a dominant model if SNVs are provided as 0-1 genotypes.

The cjamp function returns point estimates of all parameters, standard error estimates and p-values for all marginal parameters (i.e. all parameters for predictors_Y1, predictors_Y1), the minus log-likelihood value as well as information about the convergence. The cjamp_loop function only returns point estimates, standard error estimates, and p-values for the specified predictors predictors and not the covariates covariates_Y1 and covariates_Y2, in addition to the minus log-likelihood value as well as convergence information.

It is recommended that all variables are centered and scaled before the analysis, which can be done through the scale_var parameter. Otherwise, if the scales of the variables differ, it can lead to convergence problems of the optimization.

Value

An object of class cjamp, for which the summary function summary.cjamp is implemented. The output is a list containing estimates of the copula parameters, marginal parameters, Kendall's tau (as well as the upper and lower tail dependence λ_l, λ_u if the 2-parameter copula model is fitted), the standard error estimates of all parameters, p-values of hypothesis tests of the marginal parameters (i.e. of the absence of predictor effects on the phenotypes), the convergence code of the log-likelihood maximization (from the optimx function, where 0 indicates successful convergence), the KKT conditions 1 and 2 of the convergence, and the maximum log-likelihood value.

Examples

```
# Data generation
set.seed(10)
genodata <- generate_genodata(n_SNV = 20, n_ind = 100)
phenodata <- generate_phenodata_2_copula(genodata = genodata$SNV1,
                                         MAF_cutoff = 1, prop_causal = 1,
                                         tau = 0.2, b1 = 0.3, b2 = 0.3)
predictors <- data.frame(X1 = phenodata$X1, X2 = phenodata$X2,
                         genodata[, 1:3])

## Not run. When executing, the following takes about 2 minutes running time.
## Example 1: Analysis of multiple SNVs as predictors in one model
#cjamp(copula = "Clayton", Y1 = phenodata$Y1, Y2 = phenodata$Y2,
#      predictors_Y1 = predictors, predictors_Y2 = predictors,
#      optim_method = "BFGS", trace = 0, kkt2tol = 1E-16, SE_est = TRUE,
#      pval_est = TRUE, n_iter_max = 10)
#cjamp(copula = "2param", Y1 = phenodata$Y1, Y2 = phenodata$Y2,
#      predictors_Y1 = predictors, predictors_Y2 = predictors,
#      optim_method = "BFGS", trace = 0, kkt2tol = 1E-16, SE_est = TRUE,
#      pval_est = TRUE, n_iter_max = 10)
#
## Example 2: Analysis of multiple SNVs in separate models
#covariates <- data.frame(X1 = phenodata$X1, X2 = phenodata$X2)
#predictors <- genodata
#cjamp_loop(copula = "Clayton", Y1 = phenodata$Y1, Y2 = phenodata$Y2,
#           predictors = predictors, covariates_Y1 = covariates,
#           covariates_Y2 = covariates, optim_method = "BFGS", trace = 0,
#           kkt2tol = 1E-16, SE_est = TRUE, pval_est = TRUE,
#           n_iter_max = 10)
```

compute_expl_var *Phenotypic variance explained by genetic variants.*

Description

Function to estimate the percentage of the variance of a phenotype that can be explained by given single nucleotide variants (SNVs).

Usage

```
compute_expl_var(genodata = NULL, phenodata = NULL,
  type = "Rsquared_unadj", causal_idx = NULL, effect_causal = NULL)
```

Arguments

genodata	Numeric vector or dataframe containing the genetic variant(s) in columns. Must be in allelic coding 0, 1, 2.
phenodata	Numeric vector or dataframe of the phenotype.
type	String (vector) specifying the estimation approach(es) that are computed. Available are the methods "Rsquared_unadj", "Rsquared_adj", "MAF_based", and "MAF_based_Y_adjusted". See below for more details.
causal_idx	Vector with entries TRUE, FALSE specifying which SNVs are causal. Has to be supplied for the approaches MAF_based and MAF_based_Y_adjusted.
effect_causal	Numeric vector containing the effect sizes of the causal SNVs. Has to be supplied for the approaches MAF_based and MAF_based_Y_adjusted.

Details

Four different approaches are available to estimate the percentage of explained phenotypic variance (Laird & Lange, 2011):

(1) "Rsquared_unadj": Unadjusted R^2 from a linear regression of the phenotype conditional on all provided SNVs.

(2) "Rsquared_adj": Adjusted R^2 from a linear regression of the phenotype conditional on all provided SNVs.

(3) "MAF_based": Expected explained phenotypic variance computed based on the MAF and effect size of the provided causal SNVs.

(4) "MAF_based_Y_adjusted": Expected explained phenotypic variance computed based on the MAF and effect size of the causal SNVs, with respect to the empirical phenotypic variance, which is the broad-sense heritability relative to the empirical phenotypic variance.

References:

Laird NM, Lange C (2011). The fundamentals of modern statistical genetics. New York: Springer.

Value

A list containing the estimated percentage of explained phenotypic variance.

Examples

```
set.seed(10)
genodata <- generate_genodata(n_SNV = 20, n_ind = 1000)
phenodata <- generate_phenodata_1_simple(genodata = genodata[,1],
                                         type = "quantitative", b = 0)
compute_expl_var(genodata = genodata, phenodata = phenodata$Y,
                 type = c("Rsquared_unadj", "Rsquared_adj"),
                 causal_idx = NULL, effect_causal = NULL)
```

---

compute_MAF *Compute minor allele frequency of genetic variants.*

Description

Function to compute the minor allele frequency (MAF) of one or more genetic variants.

Usage

```
compute_MAF(genodata)
```

Arguments

genodata	Numeric vector or dataframe containing the genetic variants in columns. Must be in allelic coding 0, 1, 2.

Value

A vector containing the minor allele frequencies of the variants.

Examples

```
# Example of a single variant
set.seed(10)
genodata <- stats::rbinom(2000, 2, 0.3)
compute_MAF(genodata)

# Example of a set of variants
genodata <- generate_genodata()
compute_MAF(genodata)
```

generate_clayton_copula *Generate data from the Clayton copula.*

Description

Function to generate two quantitative phenotypes Y_1, Y_2, from the bivariate Clayton copula with standard normal marginal distributions.

Usage

```
generate_clayton_copula(n = NULL, phi = NULL)
```

Arguments

n	Sample size.
phi	Integer specifying the value of the copula parameter ϕ for the dependence between the two generated phenotypes.

Value

A dataframe containing n observations of Y_1, Y_2.

Examples

```
set.seed(10)
dat1a <- generate_clayton_copula(n = 1000, phi = 0.5)
dat1b <- generate_clayton_copula(n = 1000, phi = 2)
dat1c <- generate_clayton_copula(n = 1000, phi = 8)
par(mfrow = c(3, 1))
plot(dat1a$Y1, dat1a$Y2, main="Clayton copula, tau = 0.2")
plot(dat1b$Y1, dat1b$Y2, main="Clayton copula, tau = 0.5")
plot(dat1c$Y1, dat1c$Y2, main="Clayton copula, tau = 0.8")
```

generate_genodata *Functions to generate genetic data.*

Description

Functions to generate genetic data in the form of single nucleotide variants (SNVs). The function generate_singleton_data generates singletons (i.e. SNVs with one observed minor allele); generate_doubleton_data generates doubletons (i.e. SNVs with two observed minor alleles), and the function generate_genodata generates n_ind observations of n_SNV SNVs with random minor allele frequencies.

Usage

```
generate_genodata(n_SNV = 100, n_ind = 1000)

generate_singleton_data(n_SNV = 100, n_ind = 1000)

generate_doubleton_data(n_SNV = 100, n_ind = 1000)
```

Arguments

n_SNV	Integer specifying the number of SNVs that are generated.
n_ind	Integer specifying the number of observations that are generated.

Value

A dataframe containing n_ind observations of n_SNV SNVs.

Examples

```
set.seed(10)
genodata1 <- generate_singleton_data()
compute_MAF(genodata1)

genodata2 <- generate_doubleton_data()
compute_MAF(genodata2)

genodata3 <- generate_genodata()
compute_MAF(genodata3)
```

---

generate_phenodata *Functions to generate phenotype data.*

Description

Functions to generate standard normal or binary phenotypes based on provided genetic data, for specified effect sizes. The functions generate_phenodata_1_simple and generate_phenodata_1 generate one phenotype Y conditional on single nucleotide variants (SNVs) and two covariates. generate_phenodata_2_bvn as well as generate_phenodata_2_copula generate two phenotypes Y_1, Y_2 with dependence Kendall's tau conditional on the provided SNVs and two covariates.

Usage

```
generate_phenodata_1_simple(genodata = NULL, type = "quantitative",
  b = 0, a = c(0, 0.5, 0.5))

generate_phenodata_1(genodata = NULL, type = "quantitative", b = 0.6,
  a = c(0, 0.5, 0.5), MAF_cutoff = 1, prop_causal = 0.1,
  direction = "a")
```

```
generate_phenodata_2_bvn(genodata = NULL, tau = NULL, b1 = 0,
  b2 = 0, a1 = c(0, 0.5, 0.5), a2 = c(0, 0.5, 0.5))

generate_phenodata_2_copula(genodata = NULL, phi = NULL, tau = 0.5,
  b1 = 0.6, b2 = 0.6, a1 = c(0, 0.5, 0.5), a2 = c(0, 0.5, 0.5),
  MAF_cutoff = 1, prop_causal = 0.1, direction = "a")
```

Arguments

genodata	Numeric input vector or dataframe containing the genetic variant(s) in columns. Must be in allelic coding 0, 1, 2.
type	String with value "quantitative" or "binary" specifying whether normally-distributed or binary phenotypes are generated.
b	Integer or vector specifying the genetic effect size(s) of the provided SNVs (genodata) in the data generation.
a	Numeric vector specifying the effect sizes of the covariates X_1, X_2 in the data generation.
MAF_cutoff	Integer specifying a minor allele frequency cutoff to determine among which SNVs the causal SNVs are sampled for the phenotype generation.
prop_causal	Integer specifying the desired percentage of causal SNVs among all SNVs.
direction	String with value "a", "b", or "c" specifying whether all causal SNVs have a positive effect on the phenotypes ("a"), 20% of the causal SNVs have a negative effect and 80% a positive effect on the phenotypes ("b"), or 50% of the causal SNVs have a negative effect and 50% a positive effect on the phenotypes ("c").
tau	Integer specifying Kendall's tau, which determines the dependence between the two generated phenotypes.
b1	Integer or vector specifying the genetic effect size(s) of the provided SNVs (genodata) on the first phenotype in the data generation.
b2	Integer or vector specifying the genetic effect size(s) of the provided SNVs (genodata) on the second phenotype in the data generation.
a1	Numeric vector specifying the effect sizes of the covariates X_1, X_2 on the first phenotype in the data generation.
a2	Numeric vector specifying the effect sizes of the covariates X_1, X_2 on the second phenotype in the data generation.
phi	Integer specifying the parameter ϕ for the dependence between the two generated phenotypes.

Details

In more detail, the function generate_phenodata_1_simple generates a quantitative or binary phenotype Y with n observations, conditional on the specified SNVs with given effect sizes and conditional on one binary and one standard normally-distributed covariate with specified effect sizes. n is given through the provided SNVs.

generate_phenodata_1 provides an extension of generate_phenodata_1_simple and allows to further select the percentage of causal SNVs, a minor allele frequency cutoff on the causal SNVs, and varying effect directions. n is given through the provided SNVs.

The function generate_phenodata_2_bvn generates two quantitative phenotypes Y_1, Y_2 conditional on one binary and one standard normally-distributed covariate X_1, X_2 from the bivariate normal distribution so that they have have dependence τ given by Kendall's tau.

The function generate_phenodata_2_copula generates two quantitative phenotypes Y_1, Y_2 conditional on one binary and one standard normally-distributed covariate X_1, X_2 from the Clayton copula so that Y_1, Y_2 are marginally normally distributed and have dependence Kendall's tau specified by tau or phi, using the function generate_clayton_copula.

The genetic effect sizes are the specified numeric values b and b1, b2, respectively, in the functions generate_phenodata_1_simple and generate_phenodata_2_bvn. In generate_phenodata_1 and generate_phenodata_2_copula, the genetic effect sizes are computed by multiplying b or b1, b2, respectively, with the absolute value of the log10-transformed minor allele frequencies, so that rarer variants have larger effect sizes.

Value

A dataframe containing n observations of the phenotype Y or phenotypes Y_1, Y_2 and of the covariates X_1, X_2.

Examples

```
# Generate genetic data:
set.seed(10)
genodata <- generate_genodata(n_SNV = 20, n_ind = 1000)
compute_MAF(genodata)

# Generate different phenotype data:
phenodata1 <- generate_phenodata_1_simple(genodata = genodata[,1],
                                          type = "quantitative", b = 0)
phenodata2 <- generate_phenodata_1_simple(genodata = genodata[,1],
                                          type = "quantitative", b = 2)
phenodata3 <- generate_phenodata_1_simple(genodata = genodata,
                                          type = "quantitative", b = 2)
phenodata4 <- generate_phenodata_1_simple(genodata = genodata,
                                          type = "quantitative",
                                          b = seq(0.1, 2, 0.1))
phenodata5 <- generate_phenodata_1_simple(genodata = genodata[,1],
                                          type = "binary", b = 0)
phenodata6 <- generate_phenodata_1(genodata = genodata[,1],
                                   type = "quantitative", b = 0,
                                   MAF_cutoff = 1, prop_causal = 0.1,
                                   direction = "a")
phenodata7 <- generate_phenodata_1(genodata = genodata,
                                   type = "quantitative", b = 0.6,
                                   MAF_cutoff = 0.1, prop_causal = 0.05,
                                   direction = "a")
phenodata8 <- generate_phenodata_1(genodata = genodata,
```

```
                                    type = "quantitative",
                                    b = seq(0.1, 2, 0.1),
                                    MAF_cutoff = 0.1, prop_causal = 0.05,
                                    direction = "a")
phenodata9 <- generate_phenodata_2_bvn(genodata = genodata[,1],
                                       tau = 0.5, b1 = 0, b2 = 0)
phenodata10 <- generate_phenodata_2_bvn(genodata = genodata,
                                        tau = 0.5, b1 = 0, b2 = 0)
phenodata11 <- generate_phenodata_2_bvn(genodata = genodata,
                                        tau = 0.5, b1 = 1,
                                        b2 = seq(0.1,2,0.1))
phenodata12 <- generate_phenodata_2_bvn(genodata = genodata,
                                        tau = 0.5, b1 = 1, b2 = 2)
par(mfrow = c(3, 1))
hist(phenodata12$Y1)
hist(phenodata12$Y2)
plot(phenodata12$Y1, phenodata12$Y2)

phenodata13 <- generate_phenodata_2_copula(genodata = genodata[,1],
                                           MAF_cutoff = 1, prop_causal = 1,
                                           tau = 0.5, b1 = 0, b2 = 0)
phenodata14 <- generate_phenodata_2_copula(genodata = genodata,
                                           MAF_cutoff = 1, prop_causal = 0.5,
                                           tau = 0.5, b1 = 0, b2 = 0)
phenodata15 <- generate_phenodata_2_copula(genodata = genodata,
                                           MAF_cutoff = 1, prop_causal = 0.5,
                                           tau = 0.5, b1 = 0, b2 = 0)
phenodata16 <- generate_phenodata_2_copula(genodata = genodata,
                                           MAF_cutoff = 1, prop_causal = 0.5,
                                           tau = 0.2, b1 = 0.3,
                                           b2 = seq(0.1, 2, 0.1))
phenodata17 <- generate_phenodata_2_copula(genodata = genodata,
                                           MAF_cutoff = 1, prop_causal = 0.5,
                                           tau = 0.2, b1 = 0.3, b2 = 0.3)
par(mfrow = c(3, 1))
hist(phenodata17$Y1)
hist(phenodata17$Y2)
plot(phenodata17$Y1, phenodata17$Y2)
```

`get_estimates_naive` *Naive estimates of the copula and marginal parameters.*

Description

Function to compute naive estimates of the copula parameter(s) and maximum likelihood (ML) estimates of the marginal parameters in a joint copula model of `Y1` and `Y2` given the predictors of `Y1` and `Y2`. The main use of the function is to provide parameter starting values for the optimization of the log-likelihood function of the joint copula model in `cjamp` in order to obtain maximum likelihood estimates in the copula model.

Usage

```
get_estimates_naive(Y1 = NULL, Y2 = NULL, predictors_Y1 = NULL,
  predictors_Y2 = NULL, copula_param = "both")
```

Arguments

`Y1`	Numeric vector containing the first phenotype.
`Y2`	Numeric vector containing the second phenotype.
`predictors_Y1`	Dataframe containing the predictors of `Y1` in columns.
`predictors_Y2`	Dataframe containing the predictors of `Y2` in columns.
`copula_param`	String indicating whether estimates should be computed for ϕ (`"phi"`), for θ (`"theta"`), or both (`"both"`).

Details

The estimates of the copula parameter(s) include estimates of ϕ (if `copula_param == "phi"`), θ (if `copula_param == "theta"`) or both (if `copula_param == "both"`). They are obtained by computing Kendall's tau between `Y1` and `Y2` and using the relationship $\tau = \phi/(\phi+2)$ of the Clayton copula to obtain an estimate of ϕ and $\tau = (\theta - 1)/\theta$ of the Gumbel copula to obtain an estimate of θ.

The ML estimates of the marginal parameters include estimates of the log standard deviations of `Y1`, `Y2` given their predictors $(log(\sigma 1), log(\sigma 2))$ and of the effects of `predictors_Y1` on `Y1` and `predictors_Y2` on `Y2`. The estimates of the marginal effects are obtained from linear regression models of `Y1` given `predictors_Y1` and `Y2` given `predictors_Y2`, respectively. If single nucleotide variants (SNVs) are included as predictors, the genetic effect estimates are obtained from an underlying additive genetic model if SNVs are provided as 0-1-2 genotypes and from an underlying dominant model if SNVs are provided as 0-1 genotypes.

Value

Vector of the numeric estimates of the copula parameters $log(\phi)$ and/or $log(\theta - 1)$, of the marginal parameters $(log(\sigma 1), log(\sigma 2)$, and estimates of the effects of the predictors `predictors_Y1` on `Y1` and `predictors_Y2` on `Y2`).

Examples

```
# Generate genetic data:
set.seed(10)
genodata <- generate_genodata(n_SNV = 20, n_ind = 1000)
```

```
# Generate phenotype data:
phenodata <- generate_phenodata_2_copula(genodata = genodata, MAF_cutoff = 1,
                                         prop_causal = 0.5, tau = 0.2,
                                         b1 = 0.3, b2 = 0.3)
predictors <- data.frame(X1 = phenodata$X1, X2 = phenodata$X2,
                         SNV = genodata$SNV1)

get_estimates_naive(Y1 = phenodata$Y1, Y2 = phenodata$Y2,
                    predictors_Y1 = predictors, predictors_Y2 = predictors,
                    copula_param = "both")
```

`lrt` *Compute likelihood ratio tests.*

Description

Functions to compute likelihood ratio tests of two nested copula models with the same marginal models, and of marginal parameters in the same copula model but with different nested marginal models where one or more parameters are set to 0 (Yilmaz & Lawless, 2011).

Usage

```
lrt_copula(minlogl_null = NULL, minlogl_altern = NULL)

lrt_param(minlogl_null = NULL, minlogl_altern = NULL, df = NULL)
```

Arguments

`minlogl_null`	Minus log-likelihood of the null model, which is nested within the larger alternative model `minlogl_altern`.
`minlogl_altern`	Minus log-likelihood of the alternative model, which contains the null model `minlogl_null`.
`df`	Degrees of freedom in the likelihood ratio test of marginal parameters in the same copula model, i.e. the number of parameters that are in the alternative model but that are not contained in the smaller nested null model.

Details

References:

Yilmaz YE, Lawless JF (2011). Likelihood ratio procedures and tests of fit in parametric and semiparametric copula models with censored data. Lifetime Data Analysis, 17(3): 386-408.

Value

List including the χ^2 value and p-value of the likelihood ratio test.

Examples

```
# Example 1: Test whether 2-parameter copula model has a better
#            model fit compared to Clayton copula (no).
set.seed(10)
genodata <- generate_genodata(n_SNV = 20, n_ind = 100)
phenodata <- generate_phenodata_2_copula(genodata = genodata$SNV1,
                                         MAF_cutoff = 1, prop_causal = 1,
                                         tau = 0.2, b1 = 0.3, b2 = 0.3)
predictors <- data.frame(X1 = phenodata$X1, X2 = phenodata$X2,
                         SNV = genodata$SNV1)
estimates_c <- get_estimates_naive(Y1 = phenodata$Y1, Y2 = phenodata$Y2,
                                   predictors_Y1 = predictors,
                                   predictors_Y2 = predictors,
                                   copula_param = "phi")
minusloglik_Clayton <- minusloglik(Y1 = phenodata$Y1, Y2 = phenodata$Y2,
                                   predictors_Y1 = predictors,
                                   predictors_Y2 = predictors,
                                   parameters = estimates_c, copula = "Clayton")
estimates_2p <- get_estimates_naive(Y1 = phenodata$Y1, Y2 = phenodata$Y2,
                                    predictors_Y1 = predictors,
                                    predictors_Y2 = predictors,
                                    copula_param = "both")
minusloglik_2param <- minusloglik(Y1 = phenodata$Y1, Y2 = phenodata$Y2,
                                  predictors_Y1 = predictors,
                                  predictors_Y2 = predictors,
                                  parameters = estimates_2p, copula = "2param")
lrt_copula(minusloglik_Clayton, minusloglik_2param)

# Example 2: Test marginal parameters (alternative model has better fit).
set.seed(10)
genodata <- generate_genodata(n_SNV = 20, n_ind = 100)
phenodata <- generate_phenodata_2_copula(genodata = genodata$SNV1,
                                         MAF_cutoff = 1, prop_causal = 1,
                                         tau = 0.2, b1 = 2, b2 = 2)
predictors_1 <- data.frame(X1 = phenodata$X1, X2 = phenodata$X2)
estimates_1 <- get_estimates_naive(Y1 = phenodata$Y1, Y2 = phenodata$Y2,
                                   predictors_Y1 = predictors_1,
                                   predictors_Y2 = predictors_1,
                                   copula_param = "phi")
minusloglik_1 <- minusloglik(Y1 = phenodata$Y1, Y2 = phenodata$Y2,
                             predictors_Y1 = predictors_1,
                             predictors_Y2 = predictors_1,
                             parameters = estimates_1, copula = "Clayton")
predictors_2 <- data.frame(X1 = phenodata$X1, X2 = phenodata$X2,
                           SNV = genodata$SNV1)
estimates_2 <- get_estimates_naive(Y1 = phenodata$Y1, Y2 = phenodata$Y2,
                                   predictors_Y1 = predictors_2,
                                   predictors_Y2 = predictors_2,
                                   copula_param = "phi")
minusloglik_2 <- minusloglik(Y1 = phenodata$Y1, Y2 = phenodata$Y2,
                             predictors_Y1 = predictors_2,
```

```
                              predictors_Y2 = predictors_2,
                              parameters = estimates_2, copula = "Clayton")
lrt_param(minusloglik_1, minusloglik_2, df=2)
```

`minusloglik` *Minus log-likelihood of copula models.*

Description

Function to compute the minus log-likelihood of the joint distribution of `Y1` and `Y2` given the predictors `predictors_Y1` of `Y1` and `predictors_Y2` of `Y2` in a copula model for given parameter values `parameters`. Implemented are the Clayton and 2-parameter copula. The function assumes quantitative predictors and uses an additive model, i.e. for categorical predictors with more than 2 levels, dummy variables have to be created beforehand. Accordingly, if single nucleotide variants (SNVs) are included as predictors, the computation is based on an additive genetic model if SNVs are provided as 0-1-2 genotypes and on a dominant model if SNVs are provided as 0-1 genotypes.

Usage

```
minusloglik(copula = "Clayton", Y1 = NULL, Y2 = NULL,
  predictors_Y1 = NULL, predictors_Y2 = NULL, parameters = NULL)
```

Arguments

`copula`	String indicating whether the likelihood should be computed under the Clayton (`"Clayton"`) or 2-parameter copula (`"2param"`) model.
`Y1`	Numeric vector containing the first phenotype.
`Y2`	Numeric vector containing the second phenotype.
`predictors_Y1`	Dataframe containing the predictors of `Y1` in columns.
`predictors_Y2`	Dataframe containing predictors of `Y2` in columns.
`parameters`	Named integer vector containing the parameter estimates of the marginal and copula parameters, for which the log-likelihood will be computed. For details and the necessary format of the vector, see the details below.

Details

The number of predictors is not fixed and also different predictors can be supplied for `Y1` and `Y2`. However, for the functioning of the log-likelihood function, the `parameters` vector has to be supplied in a pre-specified form and formatting:

The vector has to include the copula parameter(s), marginal parameters for `Y1`, and marginal parameters for `Y2` in this order. For example, for the 2-parameter copula with parameters ϕ, θ and marginal models

$$Y_1 = \alpha_0 + \alpha_1 \cdot X_1 + \alpha_2 \cdot X_2 + \epsilon_1,\ \epsilon_1 \sim N(0, \sigma_1^2),$$

$$Y_1 = \beta_0 + \beta_1 \cdot X_1 + \epsilon_2,\ \epsilon_2 \sim N(0, \sigma_2^2),$$

the parameter vector has be

$$(log(\phi), log(\theta - 1), log(\sigma_1), log(\sigma_2), \alpha_0, \alpha_2, \alpha_1, \beta_0, \beta_1)^T.$$

$log(\phi)$ and $log(\theta - 1)$ have to be provided instead of ϕ, θ and $log(\sigma_1)$, $log(\sigma_2)$ instead of σ_1, σ_2 for computational reasons when the log-likelihood function is optimized in `cjamp`. As further necessary format, the vector has to be named and the names of the copula parameter(s) has/have to be `log_phi` and/or `log_theta_minus1`.

Value

Minus log-likelihood value (integer).

Examples

```
# Generate genetic data:
set.seed(10)
genodata <- generate_genodata(n_SNV = 20, n_ind = 1000)

# Generate phenotype data:
phenodata <- generate_phenodata_2_copula(genodata = genodata, MAF_cutoff = 1,
                                         prop_causal = 0.5, tau = 0.2,
                                         b1 = 0.3, b2 = 0.3)

# Example 1: Log-likelihood of null model without covariates & genetic effects
estimates <- get_estimates_naive(Y1 = phenodata$Y1, Y2 = phenodata$Y2,
                                 predictors_Y1 = NULL, predictors_Y2 = NULL,
                                 copula_param = "both")
minusloglik(Y1 = phenodata$Y1, Y2 = phenodata$Y2, predictors_Y1 = NULL,
            predictors_Y2 = NULL, parameters = estimates, copula = "2param")

# Example 2: Log-likelihood of null model with covariates, without genetic effects
predictors <- data.frame(X1 = phenodata$X1, X2 = phenodata$X2)
estimates <- get_estimates_naive(Y1 = phenodata$Y1, Y2 = phenodata$Y2,
                                 predictors_Y1 = predictors,
                                 predictors_Y2 = predictors,
                                 copula_param = "both")
minusloglik(Y1 = phenodata$Y1, Y2 = phenodata$Y2, predictors_Y1 = predictors,
            predictors_Y2 = predictors, parameters = estimates, copula = "2param")

# Example 3: Log-likelihood of model with covariates & genetic effect on Y1 only
predictors_Y1 <- data.frame(X1 = phenodata$X1, X2 = phenodata$X2,
                            SNV = genodata$SNV1)
predictors_Y2 <- data.frame(X1 = phenodata$X1, X2 = phenodata$X2)
estimates <- get_estimates_naive(Y1 = phenodata$Y1, Y2 = phenodata$Y2,
                                 predictors_Y1 = predictors_Y1,
                                 predictors_Y2 = predictors_Y2,
                                 copula_param = "both")
minusloglik(Y1 = phenodata$Y1, Y2 = phenodata$Y2, predictors_Y1 = predictors_Y1,
            predictors_Y2 = predictors_Y2, parameters = estimates,
            copula = "2param")
```

```
# Example 4: Log-likelihood of model with covariates & genetic effect on Y2 only
predictors_Y1 <- data.frame(X1 = phenodata$X1, X2 = phenodata$X2)
predictors_Y2 <- data.frame(X1 = phenodata$X1, X2 = phenodata$X2,
                            SNV = genodata$SNV1)
estimates <- get_estimates_naive(Y1 = phenodata$Y1, Y2 = phenodata$Y2,
                                 predictors_Y1 = predictors_Y1,
                                 predictors_Y2 = predictors_Y2,
                                 copula_param = "both")
minusloglik(Y1 = phenodata$Y1, Y2 = phenodata$Y2, predictors_Y1 = predictors_Y1,
            predictors_Y2 = predictors_Y2, parameters = estimates,
            copula = "2param")

# Example 5: Log-likelihood of model without covariates, with genetic effects
predictors <- data.frame(SNV = genodata$SNV1)
estimates <- get_estimates_naive(Y1 = phenodata$Y1, Y2 = phenodata$Y2,
                                 predictors_Y1 = predictors,
                                 predictors_Y2 = predictors,
                                 copula_param = "both")
minusloglik(Y1 = phenodata$Y1, Y2 = phenodata$Y2, predictors_Y1 = predictors,
            predictors_Y2 = predictors, parameters = estimates, copula = "2param")

# Example 6: Log-likelihood of model with covariates & genetic effects
predictors <- data.frame(X1 = phenodata$X1, X2 = phenodata$X2, SNV = genodata$SNV1)
estimates <- get_estimates_naive(Y1 = phenodata$Y1, Y2 = phenodata$Y2,
                                 predictors_Y1 = predictors,
                                 predictors_Y2 = predictors,
                                 copula_param = "both")
minusloglik(Y1 = phenodata$Y1, Y2 = phenodata$Y2, predictors_Y1 = predictors,
            predictors_Y2 = predictors, parameters = estimates, copula = "2param")

# Example 7: Log-likelihood of model with covariates & multiple genetic effects
predictors <- data.frame(X1 = phenodata$X1, X2 = phenodata$X2, genodata[, 1:5])
estimates <- get_estimates_naive(Y1 = phenodata$Y1, Y2 = phenodata$Y2,
                                 predictors_Y1 = predictors,
                                 predictors_Y2 = predictors,
                                 copula_param = "both")
minusloglik(Y1 = phenodata$Y1, Y2 = phenodata$Y2, predictors_Y1 = predictors,
            predictors_Y2 = predictors, parameters = estimates, copula = "2param")
```

`summary.cjamp` *Summary function.*

Description

Summary function for the `cjamp` and `cjamp_loop` functions.

Usage

```
## S3 method for class 'cjamp'
summary(object = NULL, ...)
```

Arguments

object	cjamp object (output of the cjamp or cjamp_loop function).
...	Additional arguments affecting the summary produced.

Value

Formatted data frame of the results.

Examples

```
## Not run. When executing, the following takes about 2 minutes running time.
## Summary of regular cjamp function
#set.seed(10)
#genodata <- generate_genodata(n_SNV = 20, n_ind = 100)
#phenodata <- generate_phenodata_2_copula(genodata = genodata$SNV1,
#                                         MAF_cutoff = 1, prop_causal = 1,
#                                         tau = 0.2, b1 = 0.3, b2 = 0.3)
#predictors <- data.frame(X1 = phenodata$X1, X2 = phenodata$X2,
#                         genodata[, 1:3])
#results <- cjamp(Y1 = phenodata$Y1, Y2 = phenodata$Y2,
#                 predictors_Y1 = predictors, predictors_Y2 = predictors,
#                 copula = "2param", optim_method = "BFGS", trace = 0,
#                 kkt2tol = 1E-16, SE_est = TRUE, pval_est = TRUE,
#                 n_iter_max = 10)
#summary(results)
#
## Summary of looped cjamp function
#covariates <- data.frame(X1 = phenodata$X1, X2 = phenodata$X2)
#predictors <- genodata
#results <- cjamp_loop(Y1 = phenodata$Y1, Y2 = phenodata$Y2,
#                      covariates_Y1 = covariates,
#                      covariates_Y2 = covariates,
#                      predictors = predictors, copula = "Clayton",
#                      optim_method = "BFGS", trace = 0, kkt2tol = 1E-16,
#                      SE_est = TRUE, pval_est = TRUE, n_iter_max = 10)
#summary(results)
```

Index