Package ‘cfdecomp’

October 12, 2022

Type Package

Title Counterfactual Decomposition: MC Integration of the G-Formula

Version 0.4.0

Maintainer Maarten Jacob Bijlsma <maarten.bijlsma@gmail.com>

Description Provides a set of functions for counterfactual decomposition (cfdecomp). The functions available in this package decompose differences in an outcome attributable to a mediating variable (or sets of mediating variables) between groups based on counterfactual (causal inference) theory. By using Monte Carlo (MC) integration (simulations based on empirical estimates from multivariable models) we provide added flexibility compared to existing (analytical) approaches, at the cost of computational power or time. The added flexibility means that we can decompose difference between groups in any outcome or and with any mediator (any variable type and distribution). See Sudharsanan & Bijlsma (2019) <doi:10.4054/MPIDR-WP-2019-004> for more information.

Depends R (>= 3.5.0)

License GPL-3

Encoding UTF-8

LazyData true

RoxygenNote 7.1.1

NeedsCompilation no

Author Maarten Jacob Bijlsma [aut, cre],
Nikkil Sudharsanan [aut],
Peng Li [ctb]

Repository CRAN

Date/Publication 2021-08-11 20:40:02 UTC

R topics documented:

`cfd.example.data` *Example Data for the cfdecomp package*

Description

A dataframe with artificially generated data, intended to be used in a demonstration of the functions in this package.

Usage

```
cfd.example.data
```

Format

a dataframe with 6 columns, which are:

SES a factor variable with 3 levels, denoting three groups to be compared. SES stands for socio-economic status.

age a continuous variable going from 40 to 65. Age could stand for age in years.

med.gauss a mediator that is a continuous variable and normally distributed.

med.binom a mediator that is binomial.

med.pois a mediator that is a count variable and poisson distributed.

out.gauss an outcome that is a continuous variable and normally distributed.

id a variable with cluster id information

`cfd.FUN` *Flexible Function Decomposition: decompose any function that returns a vector*

Description

Flexible Function Decomposition: decompose any function that returns a vector

Usage

```
cfd.FUN(
  formula.y,
  formula.m,
  mediator,
  group,
  data,
  family.y = "binomial",
  family.m = "binomial",
  bs.size = 250,
  mc.size = 50,
  FUN.y = mean,
  alpha = 0.05,
  cluster.sample = FALSE,
  cluster.name = NA,
  cluster.mrows = FALSE,
  sample.resid.y = FALSE,
  sample.resid.m = FALSE,
  print.iteration = FALSE,
  ...
)
```

Arguments

`formula.y`	the `formula` for the multivariable model (see `glm`) for the outcome Y.
`formula.m`	the `formula` for the multivariable model (see `glm`) for the mediator M.
`mediator`	the column name of the mediator M.
`group`	column name of a factor variable containing the group identifier.
`data`	a data frame containing the variables in the model.
`family.y`	a description of the error distribution to be used in the model, see `family` for details. For the outcome variable any member of the `glm` family can be used.
`family.m`	a description of the error distribution to be used in the model, see `family` for details. For the mediator, currently `gaussian`, `binomial` and `poisson` are supported.
`bs.size`	the number of bootstrap iterations to be performed.
`mc.size`	the number of Monte Carlo iterations to be performed (more = more MC error reduction).
`FUN.y`	a function to compute the statistics which can be applied to all data subsets, this function should return a vector and should be ran on pred_y (simulated y values in the natural course or counterfactual) and optional additional columns.
`alpha`	the alpha level used to construct confidence intervals (0.05 = 95 percent confidence interval).
`cluster.sample`	set to TRUE if data are clustered in the long format (i.e. multiple rows per individual or other cluster).
`cluster.name`	the name (as a character) of the column containing the cluster identifiers.

`cluster.mrows`	for the mediator model, only allows 1 observation per mediator so that the mediator model is not weighted by number of observations. e.g. set to TRUE if the mediator is time constant in longitudinal analysis of long format data.
`sample.resid.y`	sample.resid if the `outcome` is Gaussian, should the simulation sample from the residuals of the linear regression model of the outcome to approximate the empirical distribution of the outcome in the simulation (Monte Carlo integration) (if so, set to `TRUE`), or should it sample from a Gaussian distribution with the standard deviation of the outcome? If the true distribution of the continuous outcome is not very Gaussian, the former may be preferred.
`sample.resid.m`	sample.resid if the `mediator` is Gaussian, should the simulation sample from the residuals of the linear regression model of the mediator to approximate the empirical distribution of the mediator in the simulation (Monte Carlo integration) (if so, set to `TRUE`), or should it sample from a Gaussian distribution with the standard deviation of the mediator? If the true distribution of the continuous mediator is not very Gaussian, the former may be preferred.
`print.iteration`	print the bootstrap iteration
`...`	further arguments passed to or used by methods.

Value

`out_nc_m` returns the mean level of the mediator under the natural course, which is a value that should be close to the empirically observed value of the mediator for each group. `out_nc_quantile` provides the `alpha/2` and `1-alpha/2` bootstrap quantiles for this mean (AKA bootstrap percentile confidence intervals). `out_nc_y` provides the output of the function fed into FUN.y for each bootstrap iteration, with `out_nc_quantile_y` providing the `alpha/2` and `1-alpha/2` bootstrap quantiles of that output. Similarly, `out_cf_m`, `out_cf_quantile_m`,`out_cf_y`, and `out_cf_quantile_y` provide the corresponding values for the counterfactual scenario where the mediators of the groups are equalized. `mediation` and `mediation_quantile` are not provided for this function, so should be calculated by the user based on the output. `mc_conv_info_m` and `mc_conv_info_y` provide information that can help determine the number of Monte Carlo and Bootstrap iterations needed to achieve stability. See the `Examples` for more information.

Examples

```
set.seed(100)
# the decomposition functions in our package are computationally intensive
# to make the example run quick, I perform it on a subsample (n=250) of the data:
cfd.example.sample <- cfd.example.data[sample(250),]
# define some function (here one that calculates the mean from the data)
# such a function already exists, but this is to demonstrate how to do it for one that
# will be implemented in cfd.FUN:
mean.fun <- function(data,yname) {
x <- data
return(mean(x[,yname],na.rm=TRUE))
}
# test if the function works on normal data:
mean.fun(cfd.example.sample,yname="med.pois")
# then enter it into cfd.FUN and run:
```

```
mean.results <- cfd.FUN(formula.y='out.gauss ~ SES + med.gauss + med.binom + age',
                          formula.m='med.gauss ~ SES + age',
                          mediator='med.gauss',
                          group='SES',
                          data=cfd.example.sample,
                          family.y='gaussian',
                          family.m='gaussian',
                          FUN.y=mean.fun,
                          bs.size=15,
                          mc.size=5,
                          alpha=0.05,
                          print.iteration=TRUE,
                          yname="pred_y")
# more advanced code demonstrating how to do this with a function that calculates
# the age-adjusted rate ratio and life expectancy will hopefully soon be available
# in a publication.
#' @import stats utils
```

`cfd.mean` *Mean Decomposition: parametric version*

Description

Decompose the mean difference in outcome Y between groups.

Usage

```
cfd.mean(
  formula.y,
  formula.m,
  mediator,
  group,
  data,
  family.y = "binomial",
  family.m = "binomial",
  bs.size = 1000,
  mc.size = 50,
  alpha = 0.05,
  cluster.sample = FALSE,
  cluster.name = NA,
  sample.resid = FALSE,
  print.iteration = FALSE
)
```

Arguments

`formula.y`	the `formula` for the multivariable model (see `glm`) for the outcome Y.
`formula.m`	the `formula` for the multivariable model (see `glm`) for the mediator M.

`mediator`	the column name of the mediator M.
`group`	column name of a factor variable containing the group identifier.
`data`	a data frame containing the variables in the model.
`family.y`	a description of the error distribution to be used in the model, see `family` for details. For the outcome variable any member of the `glm` family can be used.
`family.m`	a description of the error distribution to be used in the model, see `family` for details. For the mediator, currently `gaussian`, `binomial` and `poisson` are supported.
`bs.size`	the number of bootstrap iterations to be performed.
`mc.size`	the number of Monte Carlo iterations to be performed (more = more MC error reduction).
`alpha`	the alpha level used to construct confidence intervals (0.05 = 95 percent confidence interval).
`cluster.sample`	set to TRUE if data are clustered in the long format (i.e. multiple rows per individual or other cluster).
`cluster.name`	the name (as a character) of the column containing the cluster identifiers.
`sample.resid`	if the `mediator` is Gaussian, should the simulation sample from the residuals of the linear regression model of the mediator to approximate the empirical distribution of the mediator in the simulation (Monte Carlo integration) (if so, set to `TRUE`), or should it sample from a Gaussian distribution with the standard deviation of the mediator? If the true distribution of the continuous mediator is not very Gaussian, the former may be preferred.
`print.iteration`	print the bootstrap iteration

Value

`out_nc_m` returns the mean level of the mediator under the natural course, which is a value that should be close to the empirically observed value of the mediator for each group. `out_nc_quantile` provides the `alpha/2` and `1-alpha/2` bootstrap quantiles for this mean (AKA bootstrap percentile confidence intervals). `out_nc_y` and `out_nc_quantile_y` provide the corresponding values, but then for the outcome variable Y. Similarly, `out_cf_m`, `out_cf_quantile_m`,`out_cf_y`, and `out_cf_quantile_y` provide the corresponding values for the counterfactual scenario where the mediators of the groups are equalized. `mediation` returns the proportion mediated by setting the intervened on mediator to be equal in level to the reference group and `mediation_quantile` returns the 1-alpha confidence interval. `mc_conv_info_m` and `mc_conv_info_y` provide information that can help determine the number of Monte Carlo and Bootstrap iterations needed to achieve stability. See the `Examples` for more information.

Examples

```
set.seed(100)
# the decomposition functions in our package are computationally intensive
# to make the example run quick, I perform it on a subsample (n=125) of the data:
cfd.example.sample <- cfd.example.data[sample(125),]
mean.results.1 <- cfd.mean(formula.y='out.gauss ~ SES + med.gauss + med.binom + age',
                           formula.m='med.gauss ~ SES + age',
                           mediator='med.gauss',
                           group='SES',
                           data=cfd.example.sample,
                           family.y='gaussian',
                           family.m='gaussian',
                           bs.size=50,
                           mc.size=10,
                           alpha=0.05,
                           cluster.sample=TRUE,
                           cluster.name='id')
# also note that normally we would recommend a bs.size of 250+
# and an mc.size of 50+
# let's interpret the output of this function:
mean(mean.results.1$out_nc_y[,2] - mean.results.1$out_nc_y[,1])
# and after giving the gaussian mediator of SES group 2 the distribution of the one in group 1
# the difference becomes:
mean(mean.results.1$out_cf_y[,2] - mean.results.1$out_nc_y[,1])
# so the % of the outcome Y that is due to differences between the two SES groups
# in the gaussian mediator is
mean(1-(mean.results.1$out_cf_y[,2] - mean.results.1$out_nc_y[,1]) /
(mean.results.1$out_nc_y[,2] - mean.results.1$out_nc_y[,1]))
# we can also get this number, and the one from the comparison of the other SES group
# with group 1, straight from the object
mean.results.1$mediation
# and we can get the 1-alpha CI for each:
mean.results.1$mediation_quantile
# see README.md for a more detailed description of the functions in this package.
```

`cfd.quantile` *Quantile Decomposition: parametric version*

Description

Decompose the difference in a quantile of some outcome Y between groups.

Usage

```
cfd.quantile(
  formula.y,
  formula.m,
  mediator,
  group,
  data,
  family.y = "binomial",
  family.m = "binomial",
  bs.size = 1000,
  mc.size = 50,
  alpha = 0.05,
  probs = 0.5,
  cluster.sample = FALSE,
  cluster.name = NA,
  sample.resid.y = FALSE,
  sample.resid.m = FALSE,
  print.iteration = FALSE
)
```

Arguments

`formula.y`	the `formula` for the multivariable model (see `glm`) for the outcome Y.
`formula.m`	the `formula` for the multivariable model (see `glm`) for the mediator M.
`mediator`	the column name of the mediator M.
`group`	column name of the variable containing the group identifier.
`data`	a data frame containing the variables in the model.
`family.y`	a description of the error distribution to be used in the model, see `family` for details. For the outcome variable any member of the `glm` family can be used.
`family.m`	a description of the error distribution to be used in the model, see `family` for details. For the mediator, currently `gaussian`, `binomial` and `poisson` are supported.
`bs.size`	the number of bootstrap iterations to be performed.
`mc.size`	the number of Monte Carlo iterations to be performed (more = more MC error reduction).
`alpha`	the alpha level used to construct confidence intervals (0.05 = 95 percent confidence interval).
`probs`	the quantiles of interest to be decomposed, should be values between 0 and 1.
`cluster.sample`	set to TRUE if data are clustered in the long format (i.e. multiple rows per individual or other cluster).
`cluster.name`	the name (as a character) of the column containing the cluster identifiers.
`sample.resid.y`	sample.resid if the `outcome` is Gaussian, should the simulation sample from the residuals of the linear regression model of the outcome to approximate the empirical distribution of the outcome in the simulation (Monte Carlo integration) (if so, set to `TRUE`), or should it sample from a Gaussian distribution with the standard deviation of the outcome? If the true distribution of the continuous outcome is not very Gaussian, the former may be preferred.
`sample.resid.m`	sample.resid if the `mediator` is Gaussian, should the simulation sample from the residuals of the linear regression model of the mediator to approximate the empirical distribution of the mediator in the simulation (Monte Carlo integration) (if so, set to `TRUE`), or should it sample from a Gaussian distribution with the standard deviation of the mediator? If the true distribution of the continuous mediator is not very Gaussian, the former may be preferred.
`print.iteration`	print the bootstrap iteration

Value

out_nc_m returns the mean level of the mediator under the natural course, which is a value that should be close to the empirically observed value of the mediator for each group. out_nc_quantile provides the alpha/2 and 1-alpha/2 bootstrap quantiles for this mean (AKA bootstrap percentile confidence intervals). out_nc_y and out_nc_quantile_y provide the corresponding values, but then for the outcome variable Y. Similarly, out_cf_m, out_cf_quantile_m,out_cf_y, and out_cf_quantile_y provide the corresponding values for the counterfactual scenario where the mediators of the groups are equalized. mediation returns the proportion mediated by setting the intervened on mediator to be equal in level to the reference group and mediation_quantile returns the 1-alpha confidence interval.mc_conv_info_m and mc_conv_info_y provide information that can help determine the number of Monte Carlo and Bootstrap iterations needed to achieve stability. See the Examples for more information.

Examples

```
set.seed(100)
# the decomposition functions in our package are computationally intensive
# to make the example run quick, I perform it on a subsample (n=125) of the data:
cfd.example.sample <- cfd.example.data[sample(125),]
quantile.results.1 <- cfd.quantile(formula.y='out.gauss ~ SES + med.gauss + med.binom + age',
                                  formula.m='med.gauss ~ SES + age',
                                  mediator='med.gauss',
                                  group='SES',
                                  data=cfd.example.sample,
                                  family.y='gaussian',
                                  family.m='gaussian',
                                  bs.size=50,
                                  mc.size=10,
                                  alpha=0.05,
                                  probs=0.50)
# also note that normally we would recommend an bs.size of 250+
# and an mc.size of 50+
# let's interpret the output of this function:
# the differences between SES groups 1 and 2 were first:
mean(quantile.results.1$out_nc_y[,2] - quantile.results.1$out_nc_y[,1])
# and after giving the gaussian mediator of SES group 2 the distribution of the one in group 1
# the difference becomes:
mean(quantile.results.1$out_cf_y[,2] - quantile.results.1$out_nc_y[,1])
# so the % of the outcome Y that is due to differences between the two SES groups
# in the gaussian mediator is
mean(1-(quantile.results.1$out_cf_y[,2] - quantile.results.1$out_nc_y[,1]) /
(quantile.results.1$out_nc_y[,2] - quantile.results.1$out_nc_y[,1]))
# we can also get this number, and the one from the comparison of the other SES group
# with group 1, straight from the object
quantile.results.1$mediation
# and we can get the 1-alpha CI for each:
quantile.results.1$mediation_quantile
# see README.md for a more detailed description of the functions in this package.
```

`cfd.semipar.mean` *Mean Decomposition: semiparametric version*

Description

Decompose the mean difference in outcome Y between groups. In this semiparametric version, we do not assume a parametric model for the mediator: instead, we sample from the distribution of the mediator in the reference group; this can be done within strata of one or more third variables.

Usage

```
cfd.semipar.mean(
  formula,
  mediator,
  group,
  strata = NA,
  nbin = 5,
  data,
  family = "binomial",
  bs.size = 1000,
  mc.size = 50,
  alpha = 0.05,
  print.iteration = FALSE
)
```

Arguments

`formula`	the `formula` for the multivariable model (see `glm`) for the outcome Y.
`mediator`	the column name of the mediator M.
`group`	column name of the variable containing the group identifier.
`strata`	the name of a variable containing the strata of a third variable (or set of variables) within which we equalize mediator values. Ideally this is a factor variable.
`nbin`	if a numeric (i.e. non-factor or character) strata variable is defined, how many bins should be made from it within which we equalize the mediator distribution?
`data`	a data frame containing the variables in the model.
`family`	a description of the error distribution to be used in the model, see `family` for details. For the outcome variable any member of the `glm` family can be used.
`bs.size`	the number of bootstrap iterations to be performed.
`mc.size`	the number of Monte Carlo iterations to be performed (more = more MC error reduction).
`alpha`	the alpha level used to construct confidence intervals (0.05 = 95 percent confidence interval).
`print.iteration`	print the bootstrap iteration

Value

`out_nc` returns the mean level of the outcome under the natural course, which is a value that should be close to the empirically observed value of the outcome for each group. `out_nc_quantile` provides the `alpha/2` and `1-alpha/2` bootstrap quantiles for this mean (AKA bootstrap percentile confidence intervals).Similarly, `out_cf`, `out_cf_quantile`,provide the corresponding values for the counterfactual scenario where the mediators of the groups are equalized. `mediation` returns the proportion mediated by setting the intervened on mediator to be equal in level to the reference group and `mediation_quantile` returns the 1-alpha confidence interval.

Examples

```
set.seed(100)
# the decomposition functions in our package are computationally intensive
# to make the example run quick, I perform it on a subsample (n=250) of the data:
cfd.example.sample <- cfd.example.data[sample(250),]
mean.semipar.results.1 <- cfd.semipar.mean(formula='out.gauss ~ SES + med.gauss + med.binom + age',
                                           mediator='med.gauss',
                                           group='SES',
                                           strata='age',
                                           nbin=5,
                                           data=cfd.example.sample,
                                           family='gaussian',
                                           bs.size=50,
                                           mc.size=10,
                                           alpha=0.05)
# also note that normally we would recommend an bs.size of 250+
# and an mc.size of 50+
# see README.md for a more detailed description of the functions in this package.
```

`cfd.semipar.quantile` *Quantile Decomposition: semiparametric version*

Description

Decompose the difference in a quantile of some outcome Y between groups. In this semiparametric version, we do not assume a parametric model for the mediator: instead, we sample from the distribution of the mediator in the reference group; this can be done within strata of one or more third variables.

Usage

```
cfd.semipar.quantile(
  formula,
  mediator,
  group,
  strata = NA,
  nbin = 5,
  data,
  family = "Gaussian",
  bs.size = 1000,
  mc.size = 50,
  alpha = 0.05,
  probs = 0.5,
  print.iteration = FALSE
)
```

Arguments

`formula`	the `formula` for the multivariable model (see `glm`) for the outcome Y.
`mediator`	the column name of the mediator M.
`group`	column name of the variable containing the group identifier.
`strata`	the name of a variable containing the strata of a third variable (or set of variables) within which we equalize mediator values.
`nbin`	if a numeric (i.e. non-factor or character) strata variable is defined, how many bins should be made from it within which we equalize the mediator distribution?
`data`	a data frame containing the variables in the model.
`family`	a description of the error distribution to be used in the model, see `family` for details. For the outcome variable any member of the `glm` family can be used.
`bs.size`	the number of bootstrap iterations to be performed.
`mc.size`	the number of Monte Carlo iterations to be performed (more = more MC error reduction).
`alpha`	the alpha level used to construct confidence intervals (0.05 = 95 percent confidence interval).
`probs`	the quantiles of interest to be decomposed, should be values between 0 and 1.
`print.iteration`	print the bootstrap iteration

Value

`out_nc` returns the mean level of the outcome under the natural course, which is a value that should be close to the empirically observed value of the outcome for each group. `out_nc_quantile` provides the `alpha/2` and `1-alpha/2` bootstrap quantiles for this mean (AKA bootstrap percentile confidence intervals).Similarly, `out_cf`, `out_cf_quantile`,provide the corresponding values for the counterfactual scenario where the mediators of the groups are equalized. `mediation` returns the proportion mediated by setting the intervened on mediator to be equal in level to the reference group and `mediation_quantile` returns the 1-alpha confidence interval.

Examples

```
set.seed(100)
# the decomposition functions in our package are computationally intensive
# to make the example run quick, I perform it on a subsample (n=175) of the data:
cfd.example.sample <- cfd.example.data[sample(175),]
quantile.semipar.results.1 <- cfd.semipar.quantile(formula='out.gauss ~ SES +
                                                  med.gauss + med.binom + age',
                                                  mediator='med.gauss',
                                                  group='SES',
                                                  strata='age',
                                                  nbin=5,
                                                  data=cfd.example.sample,
                                                  family='gaussian',
                                                  bs.size=50,
                                                  mc.size=10,
                                                  alpha=0.05,
                                                  probs=0.50)
# also note that normally we would recommend an bs.size of 250+
# and an mc.size of 50+
# see README.md for a more detailed description of the functions in this package.
```

`cluster.resample` *Cluster Resampling: resampling long format longitudinal or otherwise clustered data*

Description

Cluster Resampling: resampling long format longitudinal or otherwise clustered data

Usage

```
cluster.resample(data, cluster.name, size = NA, newID = FALSE)
```

Arguments

`data`	a data frame containing the variables in the model.
`cluster.name`	the name (as a character) of the column containing the cluster identifiers.
`size`	a non-negative integer giving the number of items to choose, i.e. the number of clusters to resample. If not specified, takes the original data cluster size.
`newID`	if set to TRUE, gives each replicant a new ID, rather than having the old ID appear multiple times. This might be needed if you work with e.g. cluster fixed effects

Value

returns a new dataframe with resampled clusters

Examples

```
table(cfd.example.data$id)
# every ID appears 5 times
cfd.example.sample <- cluster.resample(cfd.example.data, cluster.name='id')
table(cfd.example.sample$id)
# some ID's now don't appear, and some appear more times (multiples of 5)
# the important part is that if a person (id) is resampled, all their rows of data are taken
# i.e. the function resamples clusters, rather than rows.
# this has produced 1 resample, so generally this function would be used
# inside a loop where it is used multiple times
# we use this function inside our decomposition functions when cluster.sample is
# set to TRUE in those functions.
```

`conv.mean` *Running mean function*

Description

A running mean, intended to check when convergence is achieved (useful for setting mcsize or bssize).

Usage

```
conv.mean(x)
```

Arguments

`x`	a variable whose convergence we want to check

Value

returns a running mean of the variable whose convergence we want to check

Index