Package 'QCA'

April 8, 2022

Version 3.16

Date 2022-04-08

Title Qualitative Comparative Analysis

Depends R (>= 3.6.0), admisc (> 0.22)

LazyData yes

Imports methods, shiny, venn

Description An extensive set of functions to perform Qualitative Comparative Analysis:
crisp sets ('csQCA'), temporal ('tQCA'), multi-value ('mvQCA')
and fuzzy sets ('fsQCA'), using a GUI - graphical user interface.
'QCA' is a methodology that bridges the qualitative and quantitative divide
in social science research. It uses a Boolean algorithm, resulting in a
minimal causal configuration that explains a given phenomenon.

License GPL (>= 3)

NeedsCompilation yes

Author Adrian Dusa [aut, cre, cph] (<https://orcid.org/0000-0002-3525-9253>),
Ciprian Paduraru [ctb] (<https://orcid.org/0000-0002-4518-374X>),
jQuery Foundation [cph] (jQuery library and jQuery UI library),
jQuery contributors [ctb, cph] (jQuery library; authors listed in
inst/gui/www/lib/jquery-AUTHORS.txt),
lp_solve [cph] (http://lpsolve.sourceforge.net),
Vasil Dinkov [ctb, cph] (jquery.smartmenus.js library),
Dmitry Baranovskiy [ctb, cph] (raphael.js library),
Emmanuel Quentin [ctb, cph] (raphael.inline_text_editing.js library),
Jimmy Breck-McKye [ctb, cph] (raphael-paragraph.js library),
Alrik Thiem [aut] (from version 1.0-0 up to version 1.1-3)

Maintainer Adrian Dusa <dusa.adrian@unibuc.ro>

Repository CRAN

Date/Publication 2022-04-08 15:12:39 UTC

R topics documented:

`About the QCA package` *QCA: A Package for Qualitative Comparative Analysis*

Description

The package **QCA** contains functions to perform Qualitative Comparative Analysis, complemented with a graphical user interface. It implements the comparative method as first described by Ragin (1987), and extended by Cronqvist and Berg-Schlosser (2009) and Ragin (2000, 2008). QCA is a bridge between the qualitative and quantitative research methodologies, making use of the qualitative procedures in a systematic, algorithmic way (therefore increasing the "confidence" in the results, as understood by quantitative researchers).

The Quine-McCluskey minimization algorithms implemented in this package are mathematically exact, as described by Dusa (2007b), Dusa (2010), Dusa and Thiem (2015) and Dusa (2018). They all return the same, relevant set of prime implicants for *csQCA* (binary crisp sets QCA), *mvQCA* (multi-value QCA) and *fsQCA* (fuzzy-sets QCA).

Between versions 1.0-0 and up to version 1.1-3, the package welcomed a second co-author Alrik Thiem, responsible with the manual testing and documentation of functions, while the main author Adrian Dusa developed this software.

The package gained new functionality and also other types of QCA like *tsQCA* (temporal QCA), see Caren and Panofsky (2005), Ragin and Strand (2008) and more recently also causal chains similar to those from the package **cna** (see Ambuehl et al 2015).

The results of the **QCA** package are consistent with (and sometimes better than) the results of the other software packages for QCA, most notably **fs/QCA** by Ragin and Davey (2014) and **Tosmana** by Cronqvist and Berg-Schlosser (2009). A comparison of several such software is provided by Thiem and Dusa (2013).

From version 2.0, this package uses a new graphical user interface based on the package **shiny**. To avoid developing different interfaces for different operating systems, the current GUI was designed to work in a webpage. It uses a combination of HTML, CSS, jQuery for the user interface, a custom development framework using vector graphics based on the Raphael library, and extensive Javascript code to bind these altogether. A first version of this user interface was presented by Dusa (2007a), but users should be aware the current version is much more advanced. Starting with version 2.5, the user interface gained a web-based command console to offer a complete experience of available functionality.

Version 3.0 brings major improvements and additions, most notably the implementation of a new minimization algorithm called CCubes (Consistency Cubes), that is hundreds of times faster than the previous eQMC. Starting with version 3.5, it is possible to specify conjunctural directional expectations in the minimization function, and from version 3.14 categorical data is also recognized.

Details

Package:	QCA
Type:	Package
Version:	3.16
Date:	2022-04-08
License:	GPL (>= 3)

Author(s)

Authors:
Adrian Dusa
Department of Sociology
University of Bucharest
<dusa.adrian@unibuc.ro>

Maintainer:
Adrian Dusa

References

Ambuehl, M. et al (2015) *A Package for Coincidence Analysis (CNA), R Package Version 2.0 [Computer Program]*, CRAN.

Caren, N.; Panofsky, A. (2005) "TQCA: A Technique for Adding Temporality to Qualitative Comparative Analysis." *Sociological Methods & Research* vol.34, no.2, pp.147-172.

Cronqvist, L. (2016) *Tosmana: Tool for Small-N Analysis, Version 1.522 [Computer Program]*. Trier: University of Trier. url: https://www.tosmana.net/

Dusa, A. (2007a) "User manual for the QCA(GUI) package in R". *Journal of Business Research* vol.60, no.5, pp.576-586, doi: 10.1016/j.jbusres.2007.01.002

Dusa, A. (2007b) *Enhancing Quine-McCluskey*. WP 2007-49, COMPASSS Working Papers series.

Dusa, A. (2010) "A Mathematical Approach to the Boolean Minimization Problem." *Quality & Quantity* vol.44, no.1, pp.99-113, doi: 10.1007/s111350089183x

Dusa, A.; Thiem, A. (2015) "Enhancing the Minimization of Boolean and Multivalue Output Functions With eQMC" *Journal of Mathematical Sociology* vol.39, no.2, pp.92-108, doi: 10.1080/0022250X.2014.897949

Dusa, A. (2018) "Consistency Cubes: A Fast, Efficient Method for Boolean Minimization", R Journal, doi: 10.32614/RJ2018080

Ragin, C.C. (1987) *The Comparative Method: Moving beyond Qualitative and Quantitative Strategies*. Berkeley: University of California Press.

Ragin, C.C. (2000) *Fuzzy-Set Social Science*. Chicago: University of Chicago Press.

Ragin, C.C. (2008) *Redesigning Social Inquiry: Fuzzy Sets and Beyond*. Chicago: University of Chicago Press.

Ragin, C.C.; Strand, S.I. (2008) "Using Qualitative Comparative Analysis to Study Causal Order: Comment on Caren and Panofsky (2005)". *Sociological Methods & Research* vol.36, no.4, pp.431-441.

Ragin, C.C.; Davey, S. (2014) *fs/QCA: Fuzzy-Set/Qualitative Comparative Analysis, Version 2.5 [Computer Program]*. Irvine: Department of Sociology, University of California.

Thiem, A.; Dusa, A. (2013) "Boolean Minimization in Social Science Research: A Review of Current Software for Qualitative Comparative Analysis (QCA)." *Social Science Computer Review* vol.31, no.4, pp.505-521.

`calibrate` *Calibrate raw data to crisp or fuzzy sets*

Description

This function transforms (calibrates) the raw data to either crisp or fuzzy sets values, using both the *direct* and the *indirect* methods of calibration.

Usage

```
calibrate(x, type = "fuzzy", method = "direct", thresholds = NA,
          logistic = TRUE, idm = 0.95, ecdf = FALSE, below = 1, above = 1, ...)
```

Arguments

x	A numerical causal condition.
type	Calibration type, either "crisp" or "fuzzy".
method	Calibration method, either "direct", "indirect" or "TFR".
thresholds	A vector of (named) thresholds.
logistic	Calibrate to fuzzy sets using the logistic function.
idm	The set inclusion degree of membership for the logistic function.
ecdf	Calibrate to fuzzy sets using the empirical cumulative distribution function of the raw data.
below	Numeric (non-negative), determines the shape below crossover.
above	Numeric (non-negative), determines the shape above crossover.
...	Additional parameters, mainly for backwards compatibility.

Details

Calibration is a transformational process from raw numerical data (interval or ratio level of measurement) to set membership scores, based on a certain number of qualitative anchors.

When type = "crisp", the process is similar to recoding the original values to a number of categories defined by the number of thresholds. For one threshold, the calibration produces two categories (intervals): 0 if below, 1 if above. For two thresholds, the calibration produces three categories: 0 if below the first threshold, 1 if in the interval between the thresholds and 2 if above the second threshold etc.

When type = "fuzzy", calibration produces fuzzy set membership scores, using three anchors for the increasing or decreasing *s-shaped* distributions (including the logistic function), and six anchors for the increasing or decreasing *bell-shaped* distributions.

The argument thresholds can be specified either as a simple numeric vector, or as a named numeric vector. If used as a named vector, for the first category of *s-shaped* distributions, the names of the thresholds should be:

"e"	for the full set exclusion
"c"	for the set crossover
"i"	for the full set inclusion

For the second category of *bell-shaped* distributions, the names of the thresholds should be:

"e1"	for the first (left) threshold for full set exclusion
"c1"	for the first (left) threshold for set crossover
"i1"	for the first (left) threshold for full set inclusion
"i2"	for the second (right) threshold for full set inclusion
"c2"	for the second (right) threshold for set crossover
"e2"	for the second (right) threshold for full set exclusion

If used as a simple numerical vector, the order of the values matter.

If `e` $<$ `c` $<$ `i`, then the membership function is increasing from `e` to `i`. If `i` $<$ `c` $<$ `e`, then the membership function is decreasing from `i` to `e`.

Same for the *bell-shaped* distribution, if `e1` $<$ `c1` $<$ `i1` $\leq$ `i2` $<$ `c2` $<$ `e2`, then the membership function is first increasing from `e1` to `i1`, then flat between `i1` and `i2`, and then decreasing from `i2` to `e2`. In contrast, if `i1` $<$ `c1` $<$ `e1` $\leq$ `e2` $<$ `c2` $<$ `i1`, then the membership function is first decreasing from `i1` to `e1`, then flat between `e1` and `e2`, and finally increasing from `e2` to `i2`.

When `logistic = TRUE` (the default), the argument `idm` specifies the inclusion degree of membership for the logistic function. If `logistic = FALSE`, the function returns linear *s-shaped* or *bell-shaped* distributions (curved using the arguments `below` and `above`), unless activating the argument `ecdf`.

If there is no prior knowledge on the shape of the distribution, the argument `ecdf` asks the computer to determine the underlying distribution of the empirical, observed points, and the calibrated measures are found along that distribution.

Both `logistic` and `ecdf` arguments can be used only for *s-shaped* distributions (using 3 thresholds), and they are mutually exclusive.

The parameters `below` and `above` (active only when both `logistic` and `ecdf` are deactivated, establish the degree of concentration and dilation (convex or concave shape) between the threshold and crossover:

`0 < below < 1`	dilates in a concave shape below the crossover
`below = 1`	produces a linear shape (neither convex, nor concave)
`below > 1`	concentrates in a convex shape below the crossover
`0 < above < 1`	dilates in a concave shape above the crossover
`above = 1`	produces a linear shape (neither convex, nor concave)
`above > 1`	concentrates in a convex shape above the crossover

Usually, `below` and `above` have equal values, unless specific reasons exist to make them different.

For the `type = "fuzzy"` it is also possible to use the `"indirect"` method to calibrate the data, using a procedure first introduced by Ragin (2008). The indirect method assumes a vector of thresholds to cut the original data into equal intervals, then it applies a (quasi)binomial logistic regression with a fractional polynomial equation.

The results are also fuzzy between 0 and 1, but the method is entirely different: it has no anchors (specific to the direct method), and it doesn't need to specify a calibration function to calculate the scores with.

The third method applied to fuzzy calibrations is called `"TFR"` and calibrates categorical data (such as Likert type response scales) to fuzzy values using the Totally Fuzzy and Relative method (Chelli and Lemmi, 1995).

Value

A numeric vector of set membership scores, either crisp (starting from 0 with increments of 1), or fuzzy numeric values between 0 and 1.

Author(s)

Adrian Dusa

References

Cheli, B.; Lemmi, A. (1995) "A 'Totally' Fuzzy and Relative Approach to the Multidimensional Analysis of Poverty". In *Economic Notes*, vol.1, pp.115-134.

Dusa, A. (2019) *QCA with R. A Comprehensive Resource*. Springer International Publishing, doi: 10.1007/9783319756684.

Ragin, C. (2008) "Fuzzy Sets: Calibration Versus Measurement." In *The Oxford Handbook of Political Methodology*, edited by Janet Box-Steffensmeier, Henry E. Brady, and David Collier, pp.87-121. Oxford: Oxford University Press.

Examples

```
# generate heights for 100 people
# with an average of 175cm and a standard deviation of 10cm
set.seed(12345)
x <- rnorm(n = 100, mean = 175, sd = 10)


cx <- calibrate(x, type = "crisp", thresholds = 175)
plot(x, cx, main="Binary crisp set using 1 threshold",
     xlab = "Raw data", ylab = "Calibrated data", yaxt="n")
axis(2, at = 0:1)


cx <- calibrate(x, type = "crisp", thresholds = c(170, 180))
plot(x, cx, main="3 value crisp set using 2 thresholds",
     xlab = "Raw data", ylab = "Calibrated data", yaxt="n")
axis(2, at = 0:2)


# calibrate to a increasing, s-shaped fuzzy-set
cx <- calibrate(x, thresholds = "e=165, c=175, i=185")
plot(x, cx, main = "Membership scores in the set of tall people",
     xlab = "Raw data", ylab = "Calibrated data")


# calibrate to an decreasing, s-shaped fuzzy-set
cx <- calibrate(x, thresholds = "i=165, c=175, e=185")
plot(x, cx, main = "Membership scores in the set of short people",
     xlab = "Raw data", ylab = "Calibrated data")


# when not using the logistic function, linear increase
cx <- calibrate(x, thresholds = "e=165, c=175, i=185", logistic = FALSE)
plot(x, cx, main = "Membership scores in the set of tall people",
     xlab = "Raw data", ylab = "Calibrated data")
```

```
# tweaking the parameters "below" and "above" the crossover,
# at value 3.5 approximates a logistic distribution, when e=155 and i=195
cx <- calibrate(x, thresholds = "e=155, c=175, i=195", logistic = FALSE,
      below = 3.5, above = 3.5)
plot(x, cx, main = "Membership scores in the set of tall people",
     xlab = "Raw data", ylab = "Calibrated data")


# calibrate to a bell-shaped fuzzy set
cx <- calibrate(x, thresholds = "e1=155, c1=165, i1=175, i2=175, c2=185, e2=195",
      below = 3, above = 3)
plot(x, cx, main = "Membership scores in the set of average height",
     xlab = "Raw data", ylab = "Calibrated data")


# calibrate to an inverse bell-shaped fuzzy set
cx <- calibrate(x, thresholds = "i1=155, c1=165, e1=175, e2=175, c2=185, i2=195",
      below = 3, above = 3)
plot(x, cx, main = "Membership scores in the set of non-average height",
     xlab = "Raw data", ylab = "Calibrated data")


# the default values of "below" and "above" will produce a triangular shape
cx <- calibrate(x, thresholds = "e1=155, c1=165, i1=175, i2=175, c2=185, e2=195")
plot(x, cx, main = "Membership scores in the set of average height",
     xlab = "Raw data", ylab = "Calibrated data")


# different thresholds to produce a linear trapezoidal shape
cx <- calibrate(x, thresholds = "e1=155, c1=165, i1=172, i2=179, c2=187, e2=195")
plot(x, cx, main = "Membership scores in the set of average height",
     xlab = "Raw data", ylab = "Calibrated data")


# larger values of above and below will increase membership in or out of the set
cx <- calibrate(x, thresholds = "e1=155, c1=165, i1=175, i2=175, c2=185, e2=195",
      below = 10, above = 10)
plot(x, cx, main = "Membership scores in the set of average height",
     xlab = "Raw data", ylab = "Calibrated data")


# while extremely large values will produce virtually crisp results
cx <- calibrate(x, thresholds = "e1=155, c1=165, i1=175, i2=175, c2=185, e2=195",
      below = 10000, above = 10000)
plot(x, cx, main = "Binary crisp scores in the set of average height",
     xlab = "Raw data", ylab = "Calibrated data", yaxt="n")
axis(2, at = 0:1)
abline(v = c(165, 185), col = "red", lty = 2)

# check if crisp
round(cx, 0)
```

```
# using the empirical cumulative distribution function
# require manually setting logistic to FALSE
cx <- calibrate(x, thresholds = "e=155, c=175, i=195", logistic = FALSE,
      ecdf = TRUE)
plot(x, cx, main = "Membership scores in the set of tall people",
     xlab = "Raw data", ylab = "Calibrated data")


## the indirect method, per capita income data from Ragin (2008)
inc <- c(40110, 34400, 25200, 24920, 20060, 17090, 15320, 13680, 11720,
         11290, 10940, 9800, 7470, 4670, 4100, 4070, 3740, 3690, 3590,
         2980, 1000, 650, 450, 110)

cinc <- calibrate(inc, method = "indirect",
        thresholds = "1000, 4000, 5000, 10000, 20000")

plot(inc, cinc, main = "Membership scores in the set of high income",
     xlab = "Raw data", ylab = "Calibrated data")


# calibrating categorical data
set.seed(12345)
values <- sample(1:7, 100, replace = TRUE)

TFR <- calibrate(values, method = "TFR")

table(round(TFR, 3))
```

`causalChain` *Perform CNA - coincidence analysis using QCA*

Description

This function mimics the functionality in the package **cna**, finding all possible necessary and sufficient solutions for all possible outcomes in a specific dataset.

Usage

```
causalChain(data, ordering = NULL, strict = FALSE, pi.cons = 0, pi.depth = 0,
      sol.cons = 0, sol.cov = 1, sol.depth = 0, ...)
```

Arguments

`data`	A data frame containing calibrated causal conditions.
`ordering`	A character string, or a list of character vectors specifying the causal ordering of the causal conditions.

strict	Logical, prevents causal conditions on the same temporal level to act as outcomes for each other.
pi.cons	Numerical fuzzy value between 0 and 1, minimal consistency threshold for a prime implicant to be declared as sufficient.
pi.depth	Integer, a maximum number of causal conditions to be used when searching for conjunctive prime implicants.
sol.cons	Numerical fuzzy value between 0 and 1, minimal consistency threshold for a model to be declared as sufficient.
sol.cov	Numerical fuzzy value between 0 and 1, minimal coverage threshold for a model to be declared as necessary.
sol.depth	Integer, a maximum number of prime implicants to be used when searching for disjunctive models.
...	Other arguments to be passed to functions minimize() and truthTable().

Details

Although claiming to be a novel technique, coincidence analysis is yet another form of Boolean minimization. What it does is very similar and results in the same set of solutions as performing separate QCA analyses where every causal condition from the data is considered an outcome.

This function aims to demonstrate this affirmation and show that results from package **cna** can be obtained with package **QCA**. It is not intended to offer a complete replacement for the function cna(), but only to replicate its so called "asf" - atomic solution formulas.

The three most important arguments from function cna() have direct correspondents in function minimize():

con	corresponds to sol.cons.
con.msc	corresponds to pi.cons.
cov	corresponds to sol.cov.

Two other arguments from function cna() have been directly imported in this function, to complete the list of arguments that generate the same results.

The argument ordering splits the causal conditions in different temporal levels, where prior arguments can act as causal conditions, but not as outcomes for the subsequent temporal conditions. One simple way to split conditions is to use a list object, where different components act as different temporal levels, in the order of their index in the list: conditions from the first component act as the oldest causal factors, while those from the and the last component are part of the most recent temporal level.

Another, perhaps simpler way to express the same thing is to use a single character, where factors on the same level are separated with a comma, and temporal levels are separated by the sign <.

A possible example is: "A, B, C < D, E < F".

Here, there are three temporal levels and conditions A, B and C can act as causal factors for the conditions D, E and F, while the reverse is not possible. Given that D, E and F happen in a subsequent temporal levels, they cannot act as causal conditions for A, B or C. The same thing is valid with D and E, which can act as causal conditions for F, whereas F cannot act as a causal condition for D or E, and certainly not for A, B or C.

The argument `strict` controls whether causal conditions from the same temporal level may be outcomes for each other. If activated, none of A, B and C can act as causal conditions for the other two, and the same thing happens in the next temporal level where neither D nor E can be causally related to each other.

Although the two functions reach the same results, they follow different methods. The input for the minimization behind the function `cna()` is a coincidence list, while in package **QCA** the input for the minimization procedure is a truth table. The difference is subtle but important, with the most important difference that package **cna** is not exhaustive.

To find a set of solutions in a reasonable time, the formal choice in package **cna** is to deliberately stop the search at certain (default) depths of complexity. Users are free to experiment with these depths from the argument `maxstep`, but there is no guarantee the results will be exhaustive.

On the other hand, the function `causalChain()` and generally all related functions from package **QCA** are spending more time to make sure the search is exhaustive. Depths can be set via the arguments `pi.depth` and `sol.depth`, but unlike package **cna** these are not mandatory.

By default, the package **QCA** employes a different search algorithm based on Consistency Cubes (Dusa, 2018), analysing all possible combinations of causal conditions and all possible combinations of their respective levels. The structure of the input dataset (number of causal conditions, number of levels, number of unique rows in the truth table) has a direct implication on the search time, as all of those characteristics become entry parameters when calculating all possible combinations.

Consequently, two kinds of depth arguments are provided:

`pi.depth`	the maximum number of causal conditions needed to conjunctively construct a prime implicant; it is the complexity level where the search can be stopped, as long as the PI chart can be solved.
`sol.depth`	the maximum number of prime implicants needed to disjunctively build a solution model that covers all initial positive output configurations.

These arguments introduce a possible new way of deriving prime implicants and solution models, that can lead to different results (i.e. even more parsimonious) compared to the classical Quine-McCluskey. When either of them is modified from the default value of 0, the minimization method is automatically set to `"CCubes"` and the remainders are automatically included in the minimization.

The higher these depths, the higher the search time. Connversely, the search time can be significantly shorter if these depths are smaller. Irrespective of how large `pi.depth` is, the algorithm will always stop at a maximum complexity level where no new, non-redundant prime implicants are found. The argument `sol.depth` is relevant only when activating the argument `all.sol` to solve the PI chart.

The argument `sol.cons` introduces another method of solving the PI chart. Normally, once the solution models are found among all possible combinations of k prime implicants, consistencies and coverages are subsequently calculated. When `sol.cons` is lower than 1, then models are searched based on their consistencies, which should be at least equal to this threshold.

Exhaustiveness is guaranteed in package **QCA** precisely because it uses a truth table as an input for the minimization procedure. The only exception is the option of finding solutions based on their consistency, with the argument `sol.cons`: for large PI charts, time can quickly increase to infinity, to identify all possible irredundant (disjunctions that are not subsets of previously found) disjunctive models. In such a situation, the number of combinations of all possible numbers of

prime implicants is potentially too large to be solved in a polynomial time and if not otherwise specified in the argument `sol.depth` the function `causalChain()` silently sets a complexity level of 7 prime implicants per model.

When minimizing a dataset instead of a truth table, unless otherwise specified, the argument `incl.cut` is automatically set to the minimum value between `pi.cons` and `sol.cons`, then passed to the function `truthTable()`.

Value

A list of length equal to the number of columns in the `data`. Each component contains the result of the QCA minimization for that specific column acting as an outcome.

Author(s)

Adrian Dusa

See Also

`minimize`, `truthTable`

Examples

```
## Not run:
# The following examples assume the package cna is installed
library(cna)
cna(d.educate, what = "a")

# same results with
cc <- causalChain(d.educate)
cc

# inclusion and coverage scores can be inspected for each outcome
cc$E$IC


# another example, function cna() requires specific complexity depths
cna(d.women, maxstep = c(3, 4, 9), what = "a")

# same results with, no specific depths are required
causalChain(d.women)


# multivalue data require a different function in package cna
mvcna(d.pban, ordering = list(c("C", "F", "T", "V"), "PB"),
      cov = 0.95, maxstep = c(6, 6, 10), what = "a")

# same results again, simpler command
causalChain(d.pban, ordering = "C, F, T, V < PB", sol.cov = 0.95)


# specifying a lower consistency threshold for the solutions
mvcna(d.pban, ordering = list(c("C", "F", "T", "V"), "PB"), con = .93,
```

```
      maxstep = c(6, 6, 10), what = "a")

# same thing with
causalChain(d.pban, ordering = "C, F, T, V < PB", pi.cons = 0.93,
            sol.cons = 0.95)


# setting consistency thresholds for the PIs, solutions and also
# a coverage threshold for the solution (note that an yet another
# function for fuzzy sets is needed in package cna)

dat2 <- d.autonomy[15:30, c("AU","RE", "CN", "DE")]
fscna(dat2, ordering = list("AU"), con = .9, con.msc = .85, cov = .85,
      what = "a")

# again, the same results using the same function:
causalChain(dat2, ordering = "AU", sol.cons = 0.9, pi.cons = 0.85,
            sol.cov = 0.85)

## End(Not run)
```

`complexity` *Number of combinations at a given complexity layer*

Description

This function calculates the number of all possible combinations of conditions (including all levels for each condition), at a given complexity layer.

Usage

```
complexity(n, layers, noflevels)
```

Arguments

`n`	Numeric scalar, the number of input conditions.
`layers`	Numeric vector, the complexity layer(s) with values from 1 to n.
`noflevels`	Numeric vector containing the number of levels for each of the n conditions.

Details

These are the number of combinations which the CCubes algorithm (Dusa, 2018) checks to determine the prime implicants from a minimization process.

In the bottom-up approach, CCubes first checks for single conditions (combinations of both presence and absence, or more levels if multi-value), then all possible combinations of levels for two conditions etc.

The precise equation that partitions the search space into complexity layers is:

$$\sum_{c=1}^{k} \binom{k}{c} \prod_{s=1}^{c} l_s$$

where l stands for the number of levels for each combination of c conditions out of k.

Value

A numeric vector.

Author(s)

Adrian Dusa

References

Dusa, A. (2018) "Consistency Cubes: A Fast, Efficient Method for Boolean Minimization", R Journal, doi: 10.32614/RJ2018080

Examples

```
complexity(3) # all layers from 1 to 3

complexity(5, layers = 2)
```

`findRows` *Find untenable configurations*

Description

This function finds various types of untenable assumptions that are used when excluding certain configurations from the minimization process.

Usage

```
findRows(expression = "", obj, observed = FALSE, type = 1, ...)
```

Arguments

`expression`	String: a QCA expression written in sum of products form.
`obj`	A truth table (an object of class `"QCA_tt"`) or an equivalent numerical matrix.
`observed`	Logical: also return subset relations for observed configurations, when `obj` is a truth table.
`type`	Numeric vector, specifying the type(s) of untenable configurations.
`...`	Additional arguments to be passed to function `truthTable()`, for the negation of the outcome.

Details

The primary purpose is to find untenable assumptions to be excluded from the Boolean minimization process. For this reason, the input is most of the times a truth table, but for demonstration purposes it can also be a simple matrix having column names.

It started as a function to find rows that are subsets of a given SOP expression, and it developed to cover even more untenable asumptions.

Subset rows can be anything, from remainders to the observed configurations: positive output, negative output and contradictions). By default, the function returns only the subset configurations for the remaiders, but activating the argument `observed` adds the corresponding observed configurations to the output.

It might ocasionally find negative output configurations or contradictions, but that doesn't have any side effect because they are going to be excluded from the minimization anyways, unless contradictions are included in the minimization. The only category that really matters if they are identified or not, are the positive output configurations.

The contradictory simplifying assumptions (CSAs) are those which are used for both the presence and the absence of the outcome, while simultaneous subset relations (SSRs) when observed configurations are sufficient for both the presence and the absence of the outcome. CSAs and SSRs are incoherent conterfactuals, part of a category called Untenable Assumptions.

This function takes does what is normally done with a series of commands, in a more integrated and systematic way.

Providing a truth table is sufficient to perform all these tasks, because a truth table already contains all necessary information of how it was produced, most importantly the inclusion cut-off(s). By default, it uses the same options to produce a truth table for the negation of the outcome (if the input truth table was created for its presence, or the other way round), and minimizes both to inspect their simplifying assumptions to detect which are contradictory.

Identical simplifying assumptions that found in both parsimonious solutions are declared as contradictory. Observed configurations that are sufficient for both the presence and the absence of the outcome are incoherent because of the simultaneous subset relations problem.

The following types of untenable assumptions can be searched for:

`0`	all of them
`1`	subsets of a given `expression` (default)
`2`	contradictory simplifying assumptions
`3`	simultaneous subset relations

To find contradictory simplifying assumptions, a truth table for the negated outcome is constructed, using the `incl.cut` argument from the `obj` input object. If the inclusion cut-off has a single value, the same is used for the negated outcome, and if it has two values the second is used.

If very specific cutoff values are needed for the negation of the outcome, these can be provided via the `...` argument, that will be passed to function `truthTable()`.

Value

A numeric vector of row numbers from the truth table.

Author(s)

Adrian Dusa

See Also

truthTable, minimize

Examples

```
# Lipset's binary crisp version
ttLC <- truthTable(LC, "SURV", show.cases = TRUE)

findRows("DEV*~IND*STB", ttLC)

## all subset rows from the truth table, also for observed configurations
findRows("DEV*~IND*STB", ttLC, observed = TRUE)


# Lipset's fuzzy version
ttLF <- truthTable(LF, outcome = "SURV", incl.cut = 0.8)

findRows(obj = ttLF, type = 2) # contradictory simplifying assumptions


# Contradictory simplifying assumptions using different cutoff values
# for the _negation_ of the outcome

findRows(obj = ttLF, type = 2, incl.cut = 0.9, pri.cut = 0.7)
```

---

findTh *Find calibration thresholds*

Description

The purpose of this function is to automatically find calibration thresholds for a numerical causal condition, to be split into separate groups.

Usage

```
findTh(x, n = 1, hclustm = "complete", distm = "euclidean", ...)
```

Arguments

x	A numerical causal condition.
n	The number of thresholds to find.
hclustm	The agglomeration (clustering) method to be used.
distm	The distance measure to be used.
...	Other arguments (mainly for backwards compatibility).

Details

The process of calibration into crisp sets assumes expert knowledge about the best threshold(s) that separate the raw data into the most meaningful groups.

In the absence of such knowledge, an automatic procedure might help grouping the raw data according to statistical clustering techniques.

The number of groups to split depends on the number of thresholds: one thresholds splits into two groups, two thresholds splits into three groups etc.

For more details about how many groups can be formed with how many thresholds, see `cutree()`.

More details about the clustering techniques used in this function are found using `hclust()`, and also more details about different distance measures can be found with `dist()`. This function uses their default values.

Value

A numeric vector of length `n`.

Author(s)

Adrian Dusa

See Also

`cutree`, `hclust`, `dist`

Examples

```
# hypothetical list of country GDPs
gdp <- c(460, 500, 900, 2000, 2100, 2400, 15000, 16000, 20000)


# find one threshold to separate into two groups
findTh(gdp)
# 8700


# find two thresholds to separate into two groups
findTh(gdp, n = 2)
# 8700 18000


# using different clustering methods
findTh(gdp, n = 2, hclustm = "ward.D2", distm = "canberra")
# 1450 8700
```

fuzzyand, fuzzyor *Logical operations*

Description

These functions perform logical operations AND and OR, for binary crisp or fuzzy set membership scores.

Usage

```
fuzzyand(..., na.rm = FALSE)

fuzzyor(..., na.rm = FALSE)
```

Arguments

...	Two or more numerical (calibrated) objects containing membership scores, or a matrix / data frame of calibrated columns.
na.rm	Logical, indicating whether missing values should be removed.

Value

A numerical vector of class "QCA_fuzzy", with a name attribute expression

Author(s)

Adrian Dusa

Examples

```
# -----
# Cebotari & Vink (2013, 2015)

# DEMOC*GEOCON*NATPRIDE
with(CVF, fuzzyand(DEMOC, GEOCON, NATPRIDE))

# same thing with
fuzzyand(CVF[, c(1,3,5)])

# DEMOC*~GEOCON*NATPRIDE
fa <- with(CVF, fuzzyand(DEMOC, 1 - GEOCON, NATPRIDE))
fa

attr(fa, "name")


# ETHFRACT + POLDIS
with(CVF, fuzzyor(ETHFRACT, POLDIS))
```

```
# same thing with
fuzzyor(CVF[, c(2,4)])

# ETHFRACT + ~POLDIS
fo <- with(CVF, fuzzyor(ETHFRACT, 1 - POLDIS))
fo

attr(fo, "name")
```

`generate` *Generate a custom data structure*

Description

This function acts as a DGS - Data Generating Structure for a certain SOP expression.

Usage

```
generate(expression = "", snames = "", noflevels)
```

Arguments

`expression`	String: a SOP - sum of products expression.
`snames`	A string containing the sets' names, separated by commas.
`noflevels`	Numerical vector containing the number of levels for each set.

Details

Using the power of SOP expressions, this function can generate the data for any type of expressions, either Boolean or multi-value.

Causal conditions should always be separated by a product sign "*", unless: - they are single letters, or - the set names are provided, or - the expression is multi-value

All conditions are considered binary crisp, unless the number of levels are provided in conjunction with the set names, in the order of their specification from the `snames` argument.

This is an extension of the function `expand()` from package admisc, the process of data generating process being essentially a Quine expansion to a Disjunctive Normal Form.

Value

A data frame.

Author(s)

Adrian Dusa

Examples

```
generate(D + ~AB + B~C -> Z)

# the positive configurations in their complete DNF expansion:
expanded <- expand(D + ~AB + B~C, snames = c(A, B, C, D))
# ~A~B~CD + ~A~BCD + ~AB~CD + ~ABCD + A~B~CD + A~BCD +
# AB~CD + ABCD + ~AB~C~D + ~ABC~D + AB~C~D

# which has the equivalent simpler, initial expression:
simplify(expanded)
# D + ~AB + B~C

# same structure with different set names
# (note the mandatory use of the product sign *)
generate(Alpha + ~Beta*Gamma + Gamma*~Delta -> Omicron)

# introducing an additional, irrelevant condition
# (note the product sign is not mandatory if providing the set names)
setnames <- "Alpha, Beta, Gamma, Delta, Epsilon"
dat <- generate(Alpha + ~BetaGamma + Gamma~Delta -> Omicron, snames = setnames)

head(dat)

#   Alpha Beta Gamma Delta Epsilon Omicron
# 1     0    0     0     0       0       0
# 2     0    0     0     0       1       0
# 3     0    0     0     1       0       0
# 4     0    0     0     1       1       0
# 5     0    0     1     0       0       1
# 6     0    0     1     0       1       1

minimize(dat, outcome = Omicron)

# M1: Alpha + ~Beta*Gamma + Gamma*~Delta <-> Omicron
```

Implicant matrix functions: allExpressions, createMatrix, getRow

Functions Related to the Implicant Matrix

Description

This is a set of functions dedicated to the implicant matrix, a space where all causal configurations and their minimized solutions are found.

They can produce all possible implicants and prime implicants, or all possible combinations for a specific number of causal conditions and their number of values (either binary or multi-value).

Usage

```
allExpressions(noflevels = NULL, arrange = FALSE, depth, raw = FALSE, ...)

createMatrix(noflevels = NULL, ...)

getRow(row.no = NULL, noflevels = NULL, zerobased = FALSE, ...)
```

Arguments

`noflevels`	The number of levels (values) for each causal condition.
`arrange`	Logical, if `TRUE` the result matrix is arranged for visual inspection.
`depth`	Integer, an upper number of causal conditions to form expressions with.
`raw`	Logical, if `TRUE` it returns the matrix indicating which conditions have been minimized, using `-1`.
`row.no`	A vector, the desired row numbers.
`zerobased`	Logical, if `TRUE` the first row number is zero.
`...`	Other arguments.

Details

A truth table for binary crisp conditions is a matrix with 2^k rows, where k is the number of causal conditions.

For multi-value causal conditions, the same equation can be generalised to:

$v_1 \cdot v_2 \cdot \ldots \cdot v_k$

where v is the number of values (levels) for every causal condition from 1 to k.

Implicant matrices contain all rows from the truth table, plus all of their supersets, (all implicants and prime implicants), including the empty set (Dusa 2007, 2010).

For a binary crisp set procedure, there are $3^k - 1$ possible expressions (groupings), see Ragin (2010). Including the empty set (the situation when all causal conditions have been minimized), the implicant matrix consists of exactly 3^k rows, including the truth table configurations.

In fact, 3^k is also obtained by the product:

$(2+1) \cdot (2+1) \cdot \ldots \cdot (2+1)$

For multi-value causal conditions, the same equation can be generalised to:

$(v_1+1) \cdot (v_2+1) \cdot \ldots \cdot (v_k+1)$

where every number of levels in each causal conditions is incremented with 1, to allow coding the minimization of literals in each (prime) implicant (see examples).

The function `allExpressions()` creates a matrix which contains all possible implicants and prime implicants, displayed in the original values form using the code `-1` to point the minimized literals, while the other functions use the code `0`, all other values being incremented with 1.

Specifying a smaller `depth` automatically activates the argument `arrange`.

When the argument `arrange` is activated, the output is arranged in the increasing order of the number of conditions which form conjunctions, up to the maximum number specified by the argument `depth` (which if `NULL`, it is considered equal to the number of columns in the matrix).

The function `createMatrix()` creates a base matrix for truth tables and implicant matrices.

The function `getRow()` takes the number of a row in the truth table or implicant matrix (in its decimal form), and transforms it into its binary (or multi-base) representation, as a configuration of binary or multi-values for each causal condition.

Note that R is a 1-based language (all numbers start from 1), and similarly positions in vectors and matrices start with 1. For this reason, although (mathematicall) the binary representation of the decimal number 0 (for example, at three causal conditions) is 0 0 0, in R that would be the "first" line in the implicant matrix, therefore 0 0 0 is translated into the number 1, unless the argument `zerobased` is activated.

Value

A matrix with k columns and:

$v_1 \cdot v_2 \cdot \ldots \cdot v_k$ rows if a truth table;

$(v_1 + 1) \cdot (v_2 + 1) \cdot \ldots \cdot (v_k + 1)$ rows if an implicant matrix;

x rows, equal to the length of `row.no`.

Author(s)

Adrian Dusa

References

Dusa, A. (2007b) *Enhancing Quine-McCluskey*. WP 2007-49, COMPASSS Working Papers series.

Dusa, Adrian. 2010. "A Mathematical Approach to the Boolean Minimization Problem." *Quality & Quantity* vol.44, no.1, pp.99-113.

Ragin, Charles C. (2000) *Fuzzy-Set Social Science*. Chicago: University of Chicago Press.

See Also

`expand.grid`

Examples

```
# three binary causal conditions, having two levels each: 0 and 1
noflevels <- c(2, 2, 2)


# for three binary causal conditions
allExpressions(noflevels)

# the same matrix, this time arranged better
# (last rows represent the truth table)
allExpressions(noflevels, arrange = TRUE)

# show only the implicants (excluding the truth table)
allExpressions(noflevels, arrange = TRUE, depth = 2)

# using the raw form
```

```
allExpressions(noflevels, raw = TRUE)


# create a base truth table for 3 binary conditions
createMatrix(noflevels)

# its implicant matrix
createMatrix(noflevels + 1)

# create a base truth table where the second condition has three levels
createMatrix(c(2, 3, 2))



# deriving rows
rows <- c(2, 4, 5, 7, 8, 10, 11, 13, 14, 16, 17)
mat <- getRow(rows, noflevels + 1) # note the +1
rownames(mat) <- rows
colnames(mat) <- c("A", "B", "C")
mat

# implicant matrix     normal values
#
#       A  B  C    |       A  B  C
#    2  0  0  1    |    2  -  -  0       ~C
#    4  0  1  0    |    4  -  0  -       ~B
#    5  0  1  1    |    5  -  0  0       ~B~C
#    7  0  2  0    |    7  -  1  -       B
#    8  0  2  1    |    8  -  1  0       B~C
#   10  1  0  0    |   10  0  -  -       ~A
#   11  1  0  1    |   11  0  -  0       ~A~C
#   13  1  1  0    |   13  0  0  -       ~A~B
#   14  1  1  1    |   14  0  0  0       ~A~B~C
#   16  1  2  0    |   16  0  1  -       ~AB
#   17  1  2  1    |   17  0  1  0       ~AB~C
```

`minimize` *Minimize a truth table*

Description

This function performs the QCA minimization of an input truth table, or if the input is a dataset the minimization it minimizes a set of causal conditions with respect to an outcome. Three minimization methods are available: the classical Quine-McCluskey, the enhanced Quine-McCluskey and the latest Consistency Cubes algorithm that is built for performance.

All algorithms return the same, exact solutions, see Dusa (2018) and Dusa and Thiem (2015).

Usage

```
minimize(input, include = "", dir.exp = NULL, details = FALSE, pi.cons = 0,
        sol.cons = 0, all.sol = FALSE, row.dom = FALSE, min.pin = FALSE,
        max.comb = 0, first.min = FALSE, method = "CCubes", ...)
```

Arguments

input	A truth table object (preferred) or a data frame containing calibrated causal conditions and an outcome.
include	A vector of other output values to include in the minimization process.
dir.exp	Character, a vector of directional expectations to derive the intermediate solution.
details	Logical, print more details about the solution.
pi.cons	Numerical fuzzy value between 0 and 1, minimal consistency threshold for a prime implicant to be declared as sufficient.
sol.cons	Numerical fuzzy value between 0 and 1, minimal consistency threshold for a model to be declared as sufficient.
all.sol	Logical, search for all possible models, including the non-minimal.
row.dom	Logical, perform row dominance in the prime implicants' chart to eliminate redundant prime implicants.
min.pin	Logical, terminate the search at the depth where newly found prime implicants do not contribute to minimally solving the PI chart.
max.comb	Numeric real, to limit the size of the PI chart (see Details).
first.min	Logical, to return only the very first minimal solution (see Details).
method	Minimization method, one of "CCubes" (default), or "QMC" the classical Quine-McCluskey, or "eQMC" the enhanced Quine-McCluskey.
...	Other arguments to be passed to function truthTable().

Details

Most of the times, this function takes a truth table object as the input for the minimization procedure, but the same argument can refer to a data frame containing calibrated columns.

For the later case, the function minimize() originally had some additional formal arguments which were sent to the function truthTable(): outcome, conditions, n.cut, incl.cut, show.cases, use.letters and inf.test.

All of these parameters are still possible with function minimize(), but since they are sent to the truthTable() function anyway, it is unnecessary to duplicate their explanation here. The only situation which does need an additional description relates to the argument outcome, where unlike truthTable() which accepts a single one, the function minimize() accepts multiple outcomes and performs a minimization for each of them (a situation when all columns are considered causal conditions).

The argument include specifies which other truth table rows are included in the minimization process. Most often, the remainders are included but any value accepted in the argument explain is also accepted in the argument include.

The argument exclude is used to exclude truth table rows from the minimization process, from the positive configurations and/or from the remainders. It can be specified as a vector of truth table line numbers, or as a matrix of causal combinations.

The argument dir.exp is used to specify directional expectations, as described by Ragin (2003). They can be specified using SOP (sum of products) expressions, which opens up the possibility to experiment with conjunctural directional expectations. "Don't care" conditions are simply left unspecified.

If at least one of the conditions included in the analysis is multi-value, the entire dir.exp expression should be specified in multi-value notation using squared brackets. If a condition X is crisp or fuzzy, the multi-value notation X[0] is interpreted as its absence, as in the ~X notation.

Activating the details argument has the effect of printing parameters of fit for each prime implicant and each overall model, the essential prime implicants being listed in the top part of the table. It also prints the truth table, in case the argument input has been provided as a data frame instead of a truth table object.

The default method (when all.sol = FALSE), is to find the minimal number (k) of prime implicants needed to cover all initial positive output configurations (minterms), then exhaustively search through all possible disjunctions of k prime implicants which do cover those configurations.

The argument min.pin introduces an additional parameter to control when to stop the search for prime implicants. It is especially useful for very large truth tables and uses an observation by Dusa (2018) that out of the entire set of non redundant prime implicants, only a subset actually contribute to minimally solving the PI chart. The search depth is shortened when the PIs found at the next complexity level do not decrease the minimum k, thus producing absolute minimal models in both number of disjunctive prime implicants, and their depth level.

Once the PI chart is constructed using the prime implicants found in the previous stages, the argument row.dom can be used to further eliminate irrelevant prime implicants when solving the PI chart, applying the principle of row dominance: if a prime implicant A covers the same (intial) positive output configurations as another prime implicant B and in the same time covers other configurations which B does not cover, then B is irrelevant and eliminated.

A large number of causal conditions (i.e. over 15), combined with a large number of cases (i.e. hundreds) usually produce a very large number of prime implicants, resulting in a huge and extremely complex PI chart with sometimes thousands of rows and hundreds of columns.

For such a complex PI chart, even finding a minimum is a formidable task, and exhaustively solving it is very likely impossible in polynomial time. For this reason, after each level of complexity the CCubes algorithm determines if the PI chart is too difficult, by calculating the total number of combinations of minimum k PIs necessary to cover all columns.

The argument max.comb controls this maximum number of combinations. It is a rational number counted in (fractions of) billions, defaulted at zero to signal searching to the maximum possible extent. If the total number of combinations exceeds a positive value of max.comb, the PI chart is determined as too complex, the search is stopped and CCubes attempts to return all possible models using the PIs from the previous levels of complexity, when the PI chart was still not too complex.

In the extreme situation even this is not feasible, the argument first.min controls returning only one (the very first found) minimal model, if at all possible.

Value

An object of class "qca" when using a single outcome, or class "mqca" when using multiple outcomes. These objects are lists having the following components:

tt	The truth table object.
options	Values for the various options used in the function (including defaults).
negatives	The line number(s) of the negative configuration(s).
initials	The initial positive configuration(s).
PIchart	A list containing the PI chart(s).
primes	The prime implicant(s).
solution	A list of solution model(s).
essential	A list of essential PI(s).
pims	A list of PI membership scores.
IC	The matrix containing the inclusion and coverage scores for the model(s).
SA	A list of simplifying assumptions.
i.sol	A list of components specific to intermediate model(s), each having a PI chart, prime implicant membership scores, (non-simplifying) easy counterfactuals and difficult counterfactuals.
complex	Flag solutions from a too complex PI chart
call	The user's command which produced all these objects and result(s).

Author(s)

Adrian Dusa

References

Cebotari, V.; Vink, M.P. (2013) "A Configurational Analysis of Ethnic Protest in Europe". *International Journal of Comparative Sociology* vol.54, no.4, pp.298-324, doi: 10.1177/0020715213508567.

Cebotari, V.; Vink, M.P. (2015) "Replication Data for: A configurational analysis of ethnic protest in Europe", Harvard Dataverse, V2, doi: 10.7910/DVN/PT2IB9.

Cronqvist, L.; Berg-Schlosser, D. (2009) "Multi-Value QCA (mvQCA)", in Rihoux, B.; Ragin, C. (eds.) *Configurational Comparative Methods. Qualitative Comparative Analysis (QCA) and Related Techniques*, SAGE.

Dusa, A.; Thiem, A. (2015) "Enhancing the Minimization of Boolean and Multivalue Output Functions With eQMC" *Journal of Mathematical Sociology* vol.39, no.2, pp.92-108, doi: 10.1080/0022250X.2014.897949.

Dusa, A. (2018) "Consistency Cubes: A Fast, Efficient Method for Boolean Minimization", R Journal vol.10, issue 2, pp. 357-370, doi: 10.32614/RJ2018080

Dusa, A. (2019) *QCA with R. A Comprehensive Resource*. Springer International Publishing, doi: 10.1007/9783319756684.

Ragin, C. (2003) *Recent Advances in Fuzzy-Set Methods and Their Application to Policy Questions*. WP 2003-9, COMPASSS Working Papers series.

Ragin, C. (2009) "Qualitative Comparative Analysis Using Fuzzy-Sets (fsQCA)", in Rihoux, B.; Ragin, C. (eds.) *Configurational Comparative Methods. Qualitative Comparative Analysis (QCA) and Related Techniques*, SAGE.

Ragin, C.C.; Strand, S.I. (2008) "Using Qualitative Comparative Analysis to Study Causal Order: Comment on Caren and Panofsky (2005)." *Sociological Methods & Research* vol.36, no.4, pp.431-441, doi: 10.1177/0049124107313903.

Rihoux, B.; De Meur, G. (2009) "Crisp Sets Qualitative Comparative Analysis (mvQCA)", in Rihoux, B.; Ragin, C. (eds.) *Configurational Comparative Methods. Qualitative Comparative Analysis (QCA) and Related Techniques*, SAGE.

See Also

truthTable, factorize

Examples

```
## Not run:
# -----
# Lipset binary crisp data

# the associated truth table
ttLC <- truthTable(LC, SURV, sort.by = "incl, n", show.cases = TRUE)
ttLC

# conservative solution (Rihoux & De Meur 2009, p.57)
cLC <- minimize(ttLC)
cLC

# view the Venn diagram for the associated truth table
library(venn)
venn(cLC)

# add details and case names
minimize(ttLC, details = TRUE)

# negating the outcome
ttLCn <- truthTable(LC, ~SURV, sort.by = "incl, n", show.cases = TRUE)
minimize(ttLCn)

# parsimonious solution, positive output
pLC <- minimize(ttLC, include = "?", details = TRUE)
pLC

# the associated simplifying assumptions
pLC$SA

# parsimonious solution, negative output
pLCn <- minimize(ttLCn, include = "?", details = TRUE)
pLCn
```

```
# -----
# Lipset multi-value crisp data (Cronqvist & Berg-Schlosser 2009, p.80)

# truth table, conditions all columns from DEV to IND
# note the sequence operator ":"
ttLM <- truthTable(LM, SURV, conditions = DEV:IND,
        sort.by = "incl", show.cases = TRUE)

# conservative solution, positive output
minimize(ttLM, details = TRUE)

# parsimonious solution, positive output
minimize(ttLM, include = "?", details = TRUE)

# negate the outcome
ttLMn <- truthTable(LM, ~SURV, conditions = DEV:IND,
         sort.by = "incl", show.cases = TRUE)

# conservative solution, negative output
minimize(ttLMn, details = TRUE)

# parsimonious solution, positive output
minimize(ttLMn, include = "?", details = TRUE)


# -----
# Lipset fuzzy sets data (Ragin 2009, p.112)

ttLF <- truthTable(LF, SURV, incl.cut = 0.8, sort.by = "incl", show.cases = TRUE)

# conservative solution
minimize(ttLF, details = TRUE)

# parsimonious solution
minimize(ttLF, include = "?", details = TRUE)

# intermediate solution
minimize(ttLF, include = "?", details = TRUE,
         dir.exp = c(DEV, URB, LIT, IND, STB))

# directional expectations can also be specified using a sequence
minimize(ttLF, include = "?", details = TRUE, dir.exp = DEV:STB)


# URB as a don't care condition (left unspecified) and
# conjunctural directional expectations
minimize(ttLF, include = "?", details = TRUE,
         dir.exp = c(DEV, STB, ~LIT*IND))


# -----
# Cebotari & Vink (2013, 2015)
```

```
ttCVF <- truthTable(CVF, outcome = PROTEST, incl.cut = 0.8,
                    sort.by = "incl, n", show.cases = TRUE)

pCVF <- minimize(ttCVF, include = "?", details = TRUE)
pCVF

# inspect the PI chart
pCVF$PIchart

# DEMOC*ETHFRACT*~POLDIS is dominated by DEMOC*ETHFRACT*GEOCON
# using row dominance to solve the PI chart
pCVFrd <- minimize(ttCVF, include = "?", row.dom = TRUE, details = TRUE)

# plot the prime implicants on the outcome
pims <- pCVFrd$pims

par(mfrow = c(2, 2))
for(i in 1:4) {
    XYplot(pims[, i], CVF$PROTEST, cex.axis = 0.6)
}



# -----
# temporal QCA (Ragin & Strand 2008) serving the input as a dataset,
# which will automatically be passed to truthTable() as an intermediary
# step before the minimization

minimize(RS, outcome = REC, details = TRUE)

## End(Not run)
```

`modelFit` *Theory evaluation*

Description

Function to enable theory evaluation, as introduced by Ragin (1987, p.118) and extended Schneider & Wageman (2012, p.295), by producing parameters of fit for all possible intersections between a given theoretical statement (a SOP expresison) and the solutions found by function `minimize()`.

Usage

```
modelFit(model, theory = "")
```

Arguments

`model`	A minimization object of class `"QCA_min"`.
`theory`	Character, a SOP expression.

Details

Following Ragin's (1987) original work, theory evaluation amounts to intersecting a theoretical expectation with a model resulting from a minimization process.

There are in fact four intersections: presence - presence, presence - absence, absence - presence and absence - absence, where by "absence" is actually meant a negation of an expression using the function `negate()`.

When multiple models exist, all of them are automatically detected, negated and intersection with the theory. Intersections and parameters of fit are going to be produced using a single theoretical expression.

Value

A list containing objects of class `"QCA_pof"` with the parameters of fit. For a single theoretical expression and a single model, the object is a simple `"QCA_pof"` object.

Author(s)

Adrian Dusa

References

Ragin, C.C. (1987) *The Comparative Method: Moving beyond Qualitative and Quantitative Strategies*. Berkeley: University of California Press.

Schneider, C.Q.; Wagemann, C. (2012) *Set-Theoretic Methods for the Social Sciences: A Guide to Qualitative Comparative Analysis (QCA)*. Cambridge: Cambridge University Press.

See Also

`intersection`, `negate`, `pof`

Examples

```
# Lipset fuzzy version data

ttLF <- truthTable(LF, outcome = SURV, incl.cut = 0.8)


# parsimonious solution
pLF <- minimize(ttLF, include = "?")

# modelFit(model = pLF, theory = "DEV*STB")


# hypothetical exploration of intermediate solutions
iLF <- minimize(ttLF, include = "?", dir.exp = "1,0,0,0,0")

# modelFit(iLF, "DEV*STB")
```

`Parameters of fit` *Calculate parameters of fit*

Description

These functions returns inclusion (consistency) and coverage, plus PRI for sufficiency and RoN for necessity. The function `pofind()` is a stripped down version of the `pof()` function, to calculate parameters of fit for single conditions.

Usage

```
pof(setms = NULL, outcome = NULL, data = NULL, relation = "necessity",
    categorical = FALSE, inf.test = "", incl.cut = c(0.75, 0.5), add = NULL, ...)

pofind(data = NULL, outcome = "", conditions = "", relation = "necessity",
    categorical = FALSE, ...)
```

Arguments

`setms`	A data frame or a single vector of (calibrated) set memberships, or an expression written in sum of products form.
`outcome`	The name of the outcome column from a calibrated data frame, or the actual numerical column from the data frame, representing the outcome.
`data`	A calibrated data frame.
`conditions`	A single string containing the conditions' (columns) names separated by commas, or a character vector of conditions' names.
`relation`	The set relation to `outcome`, either `"necessity"` or `"sufficiency"`, partial words like `"suf"` being accepted (see examples).
`categorical`	Logical, use category labels if categorical causal conditions.
`inf.test`	Specifies the statistical inference test to be performed (currently only `"binom"`) and the critical significance level. It can be either a vector of length 2, or a single string containing both, separated by a comma.
`incl.cut`	The inclusion cutoff(s): either a single value for the presence of the output, or a vector of length 2, the second for the absence of the output. Used only in conjunction with the argument `inf.test`
`add`	A function, or a list containing functions, to add more parameters of fit.
`...`	Other arguments to be passed to the main function.

Details

The function `pof()` is one of the most flexible functions in the **QCA** package. Depending on particular situations, its arguments can be provided in various formats which are automatically recognized and treated accordingly.

When specified as a data frame, the argument `setms` contains any kind of set membership scores:

- calibrated causal conditions from the original data,

- membership scores from the resulting combinations (component coms) of function superSubset(),

- prime implicant membership scores (component pims) from function minimize(),

- any other, custom created combinations of set memberships.

When specified as a matrix, setms contains the crisp causal combinations similar to those found in the truth table. The number of columns in the matrix should be equal to the number of causal conditions in the original data. If some of them are minimized, they can be replaced by the numerical value -1 (see examples section).

More generally, setms can be a numerical vector of line numbers from the implicant matrix (see function createMatrix()), which are automatically transformed into their corresponding set membership scores.

The argument setms can also be a string expression, written in SOP - sum of products form.

For all other situations when setms is something else than a data frame, it requires the original data to generate the set memberships.

If character, the argument outcome is the name of the column from the original data, to be explained (it is a good practice advice to specify it using upper case letters, although it will nevertheless be converted to upper case, by default).

If the outcome column is multi-value, the argument outcome should use the standard curly-bracket notation X{value}. Multiple values are allowed, separated by a comma (for example X{1,2}). Negation of the outcome can also be performed using the tilde ~ operator, for example ~X{1,2}, which is interpreted as: "all values in X except 1 and 2" and it becomes the new outcome to be explained.

The argument outcome can also be a numerical vector of set membership values, either directly from the original data frame, or a recoded version (if originally multi-value).

The argument inf.test provides the possibility to perform statistical inference tests, comparing the calculated inclusion score with a pair of thresholds (ic1 and ic0) specified in the argument incl.cut. Currently, it can only perform binomial tests ("binom"), which means that data should only be provided as binary crisp (not multivalue, not fuzzy).

If the critical significance level is not provided, the default level of 0.05 is taken.

The resulting object will contain the calculated p-values (pval1 and pval0) from two separate, one-tailed tests with the alternative hypothesis that the true inclusion score is:
- greater than ic1 (the inclusion cut-off for an output value of 1)
- greater than ic0 (the inclusion cut-off for an output value of 0)

It should be noted that statistical tests are performing well only when the number of cases is large, otherwise they are usually not significant.

For the necessity relation, the standard measures of inclusion and coverage are supplemented with the RoN (Relevance of Necessity) measure, as suggested by Schneider & Wagemann's (2012).

The negation of both setms and outcome is accepted and recognized using the Boolean subtraction from 1. If the names of the conditions are provided via an optional (undocumented) argument conditions, the column names of the setms object are negated using the function negate().

The logical argument neg.out is deprecated, but backwards compatible. neg.out = TRUE and a tilde ~ in the outcome name don't cancel each other out, either one (or even both) signaling if the outcome should be negated.

The arguments from function `pofind()` are passed to the main function `pof()` to calculate parameters of fit.

Author(s)

Adrian Dusa

References

Cebotari, V.; Vink, M.P. (2013) "A Configurational Analysis of Ethnic Protest in Europe". *International Journal of Comparative Sociology* vol.54, no.4, pp.298-324, doi: 10.1177/0020715213508567

Schneider, C. and Wagemann, C. (2012) *Set-Theoretic Metods for the Social Sciences. A Guide to Qualitative Comparative Analysis*. Cambridge: Cambridge University Press.

See Also

`minimize`, `superSubset`

Examples

```
## Not run:
# -----
# Cebotari & Vink (2013) fuzzy data

conds <- CVF[, 1:5]
PROTEST <- CVF$PROTEST

# parameters of fit (default is necessity)
pof(conds, PROTEST)

# parameters of fit negating the conditions
pof(1 - conds, PROTEST)

# negating the outcome
pof(conds, 1 - PROTEST)

# parameters of fit for sufficiency
pof(conds, PROTEST, relation = "suf")

# also negating the outcome
pof(conds, 1 - PROTEST, relation = "suf")


# -----
# standard analysis of necessity
# using the "coms" component from superSubset()
nCVF <- superSubset(CVF, outcome = PROTEST, incl.cut = 0.90, cov.cut = 0.6)

# also checking their necessity inclusion score in the negated outcome
pof(nCVF$coms, 1 - PROTEST)
```

```
# -----
# standard analysis of sufficiency
# using the "pims" component from minimize()

# conservative solution
cCVF <- minimize(CVF, outcome = PROTEST, incl.cut = 0.8, details = TRUE)

# verify if their negations are also sufficient for the outcome
pof(1 - cCVF$pims, PROTEST, relation = "sufficiency")


# -----
# using a SOP expression, translated using the function translate()

pof(~NATPRIDE + GEOCON -> PROTEST, data = CVF)

# same for the negation of the outcome
pof(~NATPRIDE + GEOCON -> ~PROTEST, data = CVF)

# necessity is indicated by the reverse arrow
pof(~NATPRIDE + GEOCON <- PROTEST, data = CVF)


# -----
# more parameters of fit, for instance Haesebrouck' consistency

inclH <- function(x, y) {
    sum(fuzzyand(x, y)) /
    sum(fuzzyand(x, y) + sqrt(fuzzyor(x - y, 0) * x))
}

pof(~NATPRIDE + GEOCON -> ~PROTEST, data = CVF, add = inclH)

## End(Not run)
```

`PI chart functions: makeChart, findmin, solveChart`

Create and solve a prime implicants chart

Description

These functions help creating a demo for a prime implicant chart, and also show how to solve it using a minimum number of prime implicants.

Usage

```
makeChart(primes = "", configs = "", snames = "", mv = FALSE, collapse = "*", ...)

findmin(chart, ...)
```

```
solveChart(chart, row.dom = FALSE, all.sol = FALSE, depth = NULL, max.comb = 0,
           first.min = FALSE, ...)
```

Arguments

primes	A string containing prime implicants, separated by commas, or a matrix of implicants.
configs	A string containing causal configurations, separated by commas, or a matrix of causal configurations in the implicants space.
snames	A string containing the sets' names, separated by commas.
mv	Logical, row and column names in multi-value notation.
collapse	Scalar character, how to collapse different parts of the row or column names.
chart	An object of class "QCA_pic" or a logical matrix.
row.dom	Logical, apply row dominance to eliminate redundant prime implicants.
all.sol	Derive all possible solutions, irrespective if the disjunctive number of prime implicants is minimal or not.
depth	A maximum number of prime implicants for any disjunctive solution.
max.comb	Numeric, to stop searching for solutions (see Details).
first.min	Logical, to return only the very first minimal solution (see Details).
...	Other arguments (mainly for backwards compatibility).

Details

A PI chart, in this package, is a logical matrix (with TRUE/FALSE values), containing the prime implicants on the rows and the observed positive output configurations on the columns. Such a chart is produced by makeChart(), and it is useful to visually determine which prime implicants (if any) are essential.

When primes and configs are character, the individual sets are identified using the function translate() from package admisc, using the SOP - Sum Of Products form, which needs the set names in the absence of any other information. If products are formed using the standard * operator, specifying the set names is not mandatory.

When primes and configs are matrices, they have to be specified at implicants level, where the value 0 is interpreted as a minimized literal.

The chart is subsequently processed algorithmically by solveChart() to find the absolute minimal number M of rows (prime implicants) necessary to cover all columns, then searches through all possible combinations of M rows, to find those which actually cover the columns.

The number of all possible combinations of M rows increases exponentially with the number of prime implicants generated by the Quine-McCluskey minimization procedure, and the solving time quickly grows towards infinity for large PI charts.

To solve the chart in a minimal time, the redundant prime implicants need to first be eliminated. This is the purpose of the argument row.dom. When activated, it eliminates the dominated rows (those which cover a smaller number of columns than another, dominant prime implicant).

The identification of the full model space (including the non-minimal solutions) requires the entire PI chart and is guaranteed to consume a lot of time (towards infinity for very large PI charts). This is done by activating the argument `all.sol`, which automatically deactivates the argument `row.dom`.

The argument `depth` is relevant only when the argument `all.sol` is activated, and it is automatically increased if the minimal number of rows `M` needed to cover all columns is larger. By default, it bounds the disjunctive solutions to at most 5 prime implicants, but this number can be increased to widen the search space, with a cost of increasing the search time.

The argument `max.comb` sets a maximum number of combinations to find solutions. It is counted in (fractions of) billions, defaulted at zero to signal searching to the maximum possible extent. If too complex, the search is stopped and the algorithm returns all found solutions up to that point.

For extremly difficult PI charts, the argument `first.min` controls returning only one (the very first found) solution.

Value

For `makeChart`: a logical matrix of class `"QCA_pic"`.

For `findmin`: a numerical scalar.

For `solveChart`: a matrix containing all possible combinations of PI chart rows necessary to cover all its columns.

Author(s)

Adrian Dusa

References

Quine, W.V. (1952) *The Problem of Simplifying Truth Functions*, The American Mathematical Monthly, vol.59, no.8. (Oct., 1952), pp.521-531.

Ragin, Charles C. (1987) *The Comparative Method. Moving beyond qualitative and quantitative strategies*, Berkeley: University of California Press

Examples

```
# non-standard products, it needs the set names
chart <- makeChart("a, b, ~c", "abc, a~b~c, a~bc, ~ab~c")

# same with unquoted expressions
chart <- makeChart(c(a, b, ~c), c(abc, a~b~c, a~bc, ~ab~c))

chart
#      abc   a~b~c a~bc  ~ab~c
# a      x     x     x     -
# b      x     -     -     x
# ~c     -     x     -     x

findmin(chart)
# 2

solveChart(chart)
```

```
# first and second rows (a + b)
# and first and third rows (a + ~c)
# a is an essential prime implicant
#      a + b  a + ~c
#      [,1]   [,2]
# [1,]    1      1
# [2,]    2      3

# using SOP standard product sign
rows <- "EF, ~GH, IJ"
cols <- "~EF*~GH*IJ, EF*GH*~IJ, ~EF*GH*IJ, EF*~GH*~IJ"
chart <- makeChart(rows, cols)
chart
#     ~EF*~GH*IJ EF*GH*~IJ ~EF*GH*IJ EF*~GH*~IJ
# EF      -          x         -         x
# ~GH     x          -         -         x
# IJ      x          -         x         -

solveChart(chart)
# ~GH is redundant
#     EF + IJ
#      [,1]
# [1,]    1
# [2,]    3


# using implicant matrices
primes <- matrix(c(2,2,1,0,2,2,0,2,2,2), nrow = 2)
configs <- matrix(c(2,2,2,1,1,2,2,2,2,1,2,2,2,2,2), nrow = 3)
colnames(primes) <- colnames(configs) <- letters[1:5]

# the prime implicants: a~bce and acde
primes
#      a b c d e
# [1,] 2 1 2 0 2
# [2,] 2 0 2 2 2

# the initial causal combinations: a~bc~de, a~bcde and abcde
configs
#      a b c d e
# [1,] 2 1 2 1 2
# [2,] 2 1 2 2 2
# [3,] 2 2 2 2 2

chartLC <- makeChart(primes, configs, collapse = "")
chartLC
#         a~bc~de a~bcde   abcde
# a~bce      x       x       -
# acde       -       x       x
```

`retention` *Compute the retention probability of a csQCA solution*

Description

This function computes the retention probability for a csQCA solution, under various perturbation scenarios. It only works with bivalent crisp-set data, containing the binary values 0 or 1.

Usage

```
retention(data, outcome = "", conditions = "", incl.cut = 1, n.cut = 1,
         type = "corruption", dependent = TRUE, p.pert = 0.5, n.pert = 1)
```

Arguments

`data`	A dataset of bivalent crisp-set factors.
`outcome`	The name of the outcome.
`conditions`	A string containing the condition variables' names, separated by commas.
`incl.cut`	The minimum sufficiency inclusion score for an output function value of "1".
`n.cut`	The minimum number of cases for a causal combination with a set membership score above 0.5, for an output function value of "0" or "1".
`type`	Simulate corruptions of values in the conditions ("corruption"), or cases deleted entirely ("deletion").
`dependent`	Logical, if `TRUE` indicating DPA - Dependent Perturbations Assumption and if `FALSE` indicating IPA - Independent Perturbations Assumption.
`p.pert`	Probability of perturbation under independent (IPA) assumption.
`n.pert`	Number of perturbations under dependent (DPA) assumption.

Details

The argument `data` requires a suitable data set, in the form of a data frame. with the following structure: values of 0 and 1 for bivalent crisp-set variables.

The argument `outcome` specifies the outcome to be explained, in upper-case notation (e.g. `X`).

The argument `conditions` specifies the names of the condition variables. If omitted, all variables in `data` are used except `outcome`.

The argument `type` controls which type of perturbations should be simulated to calculate the retention probability. When `type = "corruption"`, it simulates changes of values in the conditions (values of 0 become 1, and values of 1 become 0). When `type = "deletion"`, it calculates the probability of retaining the same solution if a number of cases are deleted from the original data.

The argument `dependent` is a logical which choses between two categories of assumptions. If `dependent = TRUE` (the default) it indicates DPA - Dependent Perturbations Assumption, when perturbations depend on each other and are tied to a fixed number of cases, ex-ante (see Thiem, Spohel and Dusa, 2016). If `dependent = FALSE`, it indicates IPA - Independent Perturbations Assumption, when perturbations are assumed to occur independently of each other.

The argument `n.cut` is one of the factors that decide which configurations are coded as logical remainders or not, in conjunction with argument `incl.cut`. Those configurations that contain fewer than `n.cut` cases with membership scores above 0.5 are coded as logical remainders (`OUT = "?"`). If the number of such cases is at least `n.cut`, configurations with an inclusion score of at least `incl.cut` are coded positive (`OUT = "1"`), while configurations with an inclusion score below `incl.cut` are coded negative (`OUT = "0"`).

The argument `p.pert` specifies the probability of perturbation under the IPA - independent perturbations assumption (when `dependent = FALSE`).

The argument `n.pert` specifies the number of perturbations under the DPA - dependent perturbations assumption (when `dependent = TRUE`). At least one perturbation is needed to possibly change a csQCA solution, otherwise the solution will remain the same (retention equal to 100%) if zero perturbations occur under this argument.

Author(s)

Adrian Dusa

References

Thiem, A.; Spoehel, R.; Dusa, A. (2015) "Replication Package for: Enhancing Sensitivity Diagnostics for Qualitative Comparative Analysis: A Combinatorial Approach", Harvard Dataverse, V1. doi: 10.7910/DVN/QE27H9

Thiem, A.; Spoehel, R.; Dusa, A. (2016) "Enhancing Sensitivity Diagnostics for Qualitative Comparative Analysis: A Combinatorial Approach." *Political Analysis* vol.24, no.1, pp.104-120.

Examples

```
# the replication data, see Thiem, Spohel and Dusa (2015)
dat <- data.frame(matrix(c(
    rep(1,25), rep(0,20), rep(c(0,0,1,0,0),3),
    0,0,0,1,0,0,1,0,0,0,0, rep(1,7),0,1),
    nrow = 16, byrow = TRUE, dimnames = list(
    c("AT","DK","FI","NO","SE","AU","CA","FR",
      "US","DE","NL","CH","JP","NZ","IE","BE"),
    c("P", "U", "C", "S", "W"))
))


# calculate the retention probability, for 2.5% probability of data corruption
# under the IPA - independent perturbation assuption
retention(dat, outcome = "W", incl.cut = 1, type = "corruption",
       dependent = FALSE, p.pert = 0.025)

# the probability that a csQCA solution will change
1 - retention(dat, outcome = "W", incl.cut = 1, type = "corruption",
       dependent = FALSE, p.pert = 0.025)
```

runGUI *run the GUI shiny app for the QCA package*

Description

Runs the graphical user interface app based on the **shiny** package.

Usage

```
runGUI(x)
```

Arguments

x	Path to the shiny app.

Details

This function is a wrapper to the `runApp()` function in package **shiny**. If `x` is not provided, it automatically locates the `gui` directory in the path where the QCA package has been installed, and runs it.

The user interface has an interactive R console in the webpage. Commands are parsed and evaluated into a dedicated environment, with efforts to capture errors and warnings.

Author(s)

Adrian Dusa

superSubset, findSubsets, findSupersets
Functions to find subsets or supersets

Description

Functions to find a list of implicants that satisfy some restrictions (see details), or to find the corresponding row numbers in the implicant matrix, for all subsets, or supersets, of a (prime) implicant or an initial causal configuration.

Usage

```
superSubset(data, outcome = "", conditions = "", relation = "necessity",
    incl.cut = 1, cov.cut = 0, ron.cut = 0, pri.cut = 0, depth = NULL,
    use.letters = FALSE, categorical = FALSE, add = NULL, ...)

findSubsets(input, noflevels = NULL, stop = NULL, ...)

findSupersets(input, noflevels = NULL, ...)
```

Arguments

data	A data frame with crisp (binary and multi-value) or fuzzy causal conditions
outcome	The name of the outcome.
conditions	A string containing the conditions' names, separated by commas.
relation	The set relation to outcome, either "necessity", "sufficiency", "necsuf" or "sufnec". Partial words like "suf" are accepted.
incl.cut	The minimal inclusion score of the set relation.
cov.cut	The minimal coverage score of the set relation.
ron.cut	The minimal score for the RoN - relevance of necessity.
pri.cut	The minimal score for the PRI - proportional reduction in inconsistency.
use.letters	Logical, use simple letters instead of original conditions' names.
categorical	Logical, use category labels if categorical causal conditions.
noflevels	A vector containing the number of levels for each causal condition plus 1 (all subsets are located in the higher dimension, implicant matrix)
input	A vector of row numbers where the (prime) implicants are located, or a matrix of configurations (only for supersets).
stop	The maximum line number (subset) to stop at, and return
depth	Integer, an upper number of causal conditions to form expressions with.
add	A function, or a list containing functions, to add more parameters of fit.
...	Other arguments, mainly for backward compatibility.

Details

The function superSubset() finds a list of implicants that satisfy some restrictions referring to the inclusion and coverage with respect to the outcome, under given assumptions of necessity and/or sufficiency.

Ragin (2000) posits that under the necessity relation, instances of the outcome constitute a subset of the instances of the cause(s). Conversely, under the sufficiency relation, instances of the outcome constitute a superset of the instances of the cause(s).

When relation = "necessity" the function finds all implicants which are supersets of the outcome, then eliminates the redundant ones and returns the surviving (minimal) supersets, provided they pass the inclusion and coverage thresholds. If none of the surviving supersets pass these thresholds, the function will find disjunctions of causal conditions, instead of conjunctions.

When relation = "sufficiency" it finds all implicants which are subsets of the outcome, and similarly eliminates the redundant ones and return the surviving (minimal) subsets.

When relation = "necsuf", the relation is interpreted as necessity, and cov.cut is automatically set equal to the inclusion cutoff incl.cut. The same automatic equality is made for relation = "sufnec", when relation is interpreted as sufficiency.

The argument outcome specifies the name of the outcome, and if multi-value the argument can also specify the level to explain, using square brackets notation.

Outcomes can be negated using a tilde operator ~X. The logical argument neg.out is now deprecated, but still backwards compatible. Replaced by the tilde in front of the outcome name, it controls

whether outcome is to be explained or its negation. If outcome is from a multivalent variable, it has the effect that the disjunction of all remaining values becomes the new outcome to be explained. neg.out = TRUE and a tilde ~ in the outcome name don't cancel each other out, either one (or even both) signaling if the outcome should be negated.

If the argument conditions is not specified, all other columns in data are used.

Along with the standard measures of inclusion and coverage, the function also returns PRI for sufficiency and RoN (relevance of necessity, see Schneider & Wagemann, 2012) for the necessity relation.

A subset is a conjunction (an intersection) of causal conditions, with respect to a larger (super)set, which is another (but more parsimonious) conjunction of causal conditions.

All subsets of a given set can be found in the so called "implicant matrix", which is a n^k space, understood as all possible combinations of values in any combination of bases n, each causal condition having three or more levels (Dusa, 2007, 2010).

For every two levels of a binary causal conditions (values 0 and 1), there are three levels in the implicants matrix:

0	to mark a minimized literal
1	to replace the value of 0 in the original binary condition
-1	to replace the value of 1 in the original binary condition

A prime implicant is a superset of an initial combination of causal conditions, and the reverse is also true: the initial combination is a subset of a prime implicant.

Any normal implicant (not prime) is a subset of a prime implicant, and in the same time a superset of some initial causal combinations.

Functions findSubsets() and findSupersets() find:

- all possible such subsets for a given (prime) implicant, or
- all possible supersets of an implicant or initial causal combination

in the implicant matrix.

The argument depth can be used to impose an upper number of causal conditions to form expressions with, it is the complexity level where the search is stopped. Depth is set to a maximum by default, and the algorithm will always stop at the maximum complexity level where no new, non-redundant prime implicants are found. Reducing the depth below that maximum will also reduce computation time.

For examples on how to add more parameters of fit via argument add, see the function pof().

Value

The result of the superSubset() function is an object of class "ss", which is a list with the following components:

incl.cov	A data frame with the parameters of fit.
coms	A data frame with the (m)embersip (s)cores of the resulting (co)mbinations.

For `findSubsets()` and `findSupersets()`, a vector with the row numbers corresponding to all possible subsets, or supersets, of a (prime) implicant.

Author(s)

Adrian Dusa

References

Cebotari, V.; Vink, M.P. (2013) "A Configurational Analysis of Ethnic Protest in Europe". *International Journal of Comparative Sociology* vol.54, no.4, pp.298-324, doi: 10.1177/0020715213508567.

Cebotari, Victor; Vink, Maarten Peter (2015) *Replication Data for: A configurational analysis of ethnic protest in Europe*, Harvard Dataverse, V2, doi: 10.7910/DVN/PT2IB9.

Dusa, A. (2007b) *Enhancing Quine-McCluskey*. WP 2007-49, COMPASSS Working Papers series.

Dusa, Adrian (2010) "A Mathematical Approach to the Boolean Minimization Problem." *Quality & Quantity* vol.44, no.1, pp.99-113, doi: 10.1007/s111350089183x.

Lipset, S. M. (1959) "Some Social Requisites of Democracy: Economic Development and Political Legitimacy", *American Political Science Review* vol.53, pp.69-105.

Schneider, Carsten Q.; Wagemann, Claudius (2012) *Set-Theoretic Methods for the Social Sciences: A Guide to Qualitative Comparative Analysis (QCA)*. Cambridge: Cambridge University Press.

See Also

`createMatrix`, `getRow`

Examples

```
# Lipset binary crisp sets
ssLC <- superSubset(LC, "SURV")

library(venn)
x = list("SURV" = which(LC$SURV == 1),
         "STB" = which(ssLC$coms[, 1] == 1),
         "LIT" = which(ssLC$coms[, 2] == 1))
venn(x, cexil = 0.7)

# Lipset multi-value sets
superSubset(LM, "SURV")

# Cebotari & Vink (2013) fuzzy data
# all necessary combinations with at least 0.9 inclusion and 0.6 coverage cut-offs
ssCVF <- superSubset(CVF, outcome = "PROTEST", incl.cut = 0.90, cov.cut = 0.6)
ssCVF

# the membership scores for the first minimal combination (GEOCON)
ssCVF$coms$GEOCON

# same restrictions, for the negation of the outcome
superSubset(CVF, outcome = "~PROTEST", incl.cut = 0.90, cov.cut = 0.6)
```

```
# to find supersets or supersets, a hypothetical example using
# three binary causal conditions, having two levels each: 0 and 1
noflevels <- c(2, 2, 2)

# second row of the implicant matrix: 0 0 1
# which in the "normal" base is:      - - 0
# the prime implicant being: ~C
(sub <- findSubsets(input = 2, noflevels + 1))
#  5  8 11 14 17 20 23 26


getRow(sub, noflevels + 1)

# implicant matrix   normal values
#      a  b  c    |       a  b  c
#   5  0  1  1    |    5  -  0  0      ~b~c
#   8  0  2  1    |    8  -  1  0      b~c
#  11  1  0  1    |   11  0  -  0      ~a~c
#  14  1  1  1    |   14  0  0  0      ~a~b~c
#  17  1  2  1    |   17  0  1  0      ~ab~c
#  20  2  0  1    |   20  1  -  0      a~c
#  23  2  1  1    |   23  1  0  0      a~b~c
#  26  2  2  1    |   26  1  1  0      ab~c


# stopping at maximum row number 20
findSubsets(input = 2, noflevels + 1, stop = 20)
#  5  8 11 14 17 20


# -----
# for supersets
findSupersets(input = 14, noflevels + 1)
#  2  4  5 10 11 13 14

findSupersets(input = 17, noflevels + 1)
#  2  7  8 10 11 16 17

# input as a matrix
(im <- getRow(c(14, 17), noflevels + 1))

# implicant matrix   normal values
#  14  1  1  1    |   14  0  0  0       ~a~b~c
#  17  1  2  1    |   17  0  1  0       ~ab~c


sup <- findSupersets(input = im, noflevels + 1)
sup
#  2  4  5  7  8 10 11 13 14 16 17


getRow(sup, noflevels + 1)
```

```
# implicant matrix   normal values
#      a  b  c    |       a  b  c
#   2  0  0  1    |    2  -  -  0       ~c
#   4  0  1  0    |    4  -  0  -       ~b
#   5  0  1  1    |    5  -  0  0       ~b~c
#   7  0  2  0    |    7  -  1  -       b
#   8  0  2  1    |    8  -  1  0       b~c
#  10  1  0  0    |   10  0  -  -       ~a
#  11  1  0  1    |   11  0  -  0       ~a~c
#  13  1  1  0    |   13  0  0  -       ~a~b
#  14  1  1  1    |   14  0  0  0       ~a~b~c
#  16  1  2  0    |   16  0  1  -       ~ab
#  17  1  2  1    |   17  0  1  0       ~ab~c
```

`truthTable` *Create a truth table*

Description

Function to create a truth table from all types of calibrated data (binary crisp, multi-value crisp and fuzzy). For fuzzy data, an improved verson of Ragin's (2008) procedure is applied to assign cases to the vector space corners (the truth table rows).

Usage

```
truthTable(data, outcome = "", conditions = "", incl.cut = 1, n.cut = 1, pri.cut = 0,
       exclude = NULL, complete = FALSE, use.letters = FALSE, categorical = FALSE,
           show.cases = FALSE, dcc = FALSE, sort.by = "", inf.test = "", ...)
```

Arguments

`data`	A data frame containing calibrated causal conditions and an outcome.
`outcome`	String, the name of the outcome.
`conditions`	A single string containing the conditions' (columns) names separated by commas, or a character vector of conditions' names.
`incl.cut`	The inclusion cut-off(s): either a single value for the presence of the output, or a vector of length 2, the second for the absence of the output.
`n.cut`	The minimum number of cases under which a truth table row is declared as a remainder.
`pri.cut`	The minimal score for the `PRI` - proportional reduction in inconsistency, under which a truth table row is declared as negative.
`exclude`	A vector of (remainder) row numbers from the truth table, to code as negative output configurations.
`complete`	Logical, print complete truth table.

use.letters	Logical, use letters instead of causal conditions' names.
categorical	Logical, use category labels if categorical causal conditions.
show.cases	Logical, print case names.
dcc	Logical, if show.cases = TRUE, the cases being displayed are the deviant cases consistency in kind.
sort.by	Sort the truth table according to various columns.
inf.test	Specifies the statistical inference test to be performed (currently only "binom") and the critical significance level. It can be either a vector of length 2, or a single string containing both, separated by a comma.
...	Other arguments (mainly for backward compatibility).

Details

The data should always be provided as a data frame, with calibrated columns.

Calibration can be either crisp, with 2 or more values starting from 0, or fuzzy with continous scores from 0 to 1. Raw data containing relative frequencies can also be continous between 0 and 1, but these are not calibrated, fuzzy data.

Some columns can contain the placeholder "-" indicating a "don't care", which is used to indicate the temporal order between other columns in tQCA. These special columns are not causal conditions, hence no parameters of fit will be calculated for them.

The argument outcome specifies the column name to be explained. If the outcome is a multivalue column, it can be specified in curly bracket notation, indicating the value to be explained (the others being automatically converted to zero).

The outcome can be negated using a tilde operator ~X. The logical argument neg.out is now deprecated, but still backwards compatible. Replaced by the tilde in front of the outcome name, it controls whether outcome is to be explained or its negation. Note that using both neg.out = TRUE and a tilde ~ in the outcome name cancel each other out.

If the outcome column is multi-value, the argument outcome should use the standard curly-bracket notation X{value}. Multiple values are allowed, separated by a comma (for example X{1,2}). Negation of the outcome can also be performed using the tilde ~ operator, for example ~X{1,2}, which is interpreted as: "all values in X except 1 and 2" and it becomes the new outcome to be explained.

The argument conditions specifies the causal conditions' names among the other columns in the data. When this argument is not specified, all other columns except for the outcome are taken as causal conditions.

A good practice advice is to specify both outcome and conditions as upper case letters. It is possible, in a next version, to negate outcomes using lower case letters, a situation where it really does matter how the outcome and/or conditions are specified.

The argument incl.cut replaces both (deprecated, but still backwards compatible) former arguments incl.cut1 and incl.cut0. Most of the analyses use the inclusion cutoff for the presence of the output (code "1"). When users need both inclusion cutoffs (see below), incl.cut can be specified as a vector of length 2, in the form: c(ic1, ic0) where:

ic1 is the inclusion cutoff for the presence of the output,

a minimum sufficiency inclusion score above which the output value is coded with "1".

ic0 is the inclusion cutoff for the absence of the output,
a maximum sufficiency inclusion score below which the output value is coded with "0".

If not specifically declared, the argument ic0 is automatically set equal to ic1, but otherwise ic0 should always be lower than ic1.

Using these two cutoffs, as well as pri.cut the observed combinations are coded with:

"1" if they have an inclusion score of at least ic1
and a PRI score of at least pri.cut

"C" if they have an inclusion score below ic1 and at least ic0 (contradiction)

"0" if they have an inclusion score below ic0 or
a PRI score below pri.cut

The argument n.cut specifies the frequency threshold under which a truth table row is coded as a remainder, irrespective of its inclusion score.

When argument show.cases is set to TRUE, the case names will be printed at their corresponding row in the truth table. The resulting object always contains the cases for each causal combination, even if not printed on the screen (the print function can later be used to print them).

The sort.by argument orders all configurations by any of the columns present in the truth table. Typically, sorting occurs by their outcome value, and/or by their inclusion score, and/or by their frequency, in any order.

Sorting decreasingly (the default) or increasingly can be specified adding the signs - or +, next after the column name in argument sort.by (see examples). Note that - is redundant because it is the default anyways.

The order specified in this vector is the order in which the configurations will be sorted. When sorting based on the OUTput column, remainders will always be sorted last.

The argument use.letters controls using the original names of the causal conditions, or replace them by single letters in alphabetical order. If the causal conditions are already named with single letters, the original letters will be used.

The argument inf.test combines the inclusion score with a statistical inference test, in order to assign values in the output column OUT. For the moment, it is only the binomial test, which needs crisp data (it doesn't work with fuzzy sets). Following a similar logic as above, for a given (specified) critical significance level, the output for a truth table row will be coded as:

"1" if the true inclusion score is significanly higher than ic1,

"C" contradiction, if the true inclusion score is not significantly higher than ic1
but significantly higher than ic0,

"0" if the true inclusion score is not significantly higher than ic0.

It should be noted that statistical tests perform well only when the number of cases is large, otherwise they are usually not significant. For a low number of cases, depending on the inclusion cutoff value(s), it will be harder to code a value of `"1"` in the output, and also harder to obtain contradictions if the true inclusion is not signficantly higher than `ic0`.

The argument `complete` controls how to print the table on the screen, either complete (when set to `TRUE`), or just the observed combinations (default). For up to 7 causal conditions, the resulting object will always contain the complete truth table, even if it's not printed on the screen. This is useful for multiple reasons: researchers like to manually change output values in the truth table (sometimes including in this way a remainder, for example), and it is also useful to plot Venn diagrams, each truth table row having a correspondent intersection in the diagram.

Value

An object of class `"tt"`, a list containing the following components:

tt	The truth table itself.
indexes	The line numbers for the observed causal configurations.
noflevels	A vector with the number of values for each causal condition.
initial.data	The initial data.
recoded.data	The crisp version of the `initial.data`, if fuzzy.
cases	The cases for each observed causal configuration.
options	The command options used.
rowsorder	The order of the rows after sorting, if using `sort.by`.
minmat	The membership scores matrix of cases in the observed truth table combinations.

Author(s)

Adrian Dusa

References

Cronqvist, L.; Berg-Schlosser, D. (2009) "Multi-Value QCA (mvQCA)", in Rihoux, B.; Ragin, C. (eds.) *Configurational Comparative Methods. Qualitative Comparative Analysis (QCA) and Related Techniques*, SAGE.

Dusa, A. (2019) *QCA with R. A Comprehensive Resource*. Springer International Publishing, doi: 10.1007/9783319756684.

Lipset, S.M. (1959) "Some Social Requisites of Democracy: Economic Development and Political Legitimacy", *American Political Science Review* vol.53, pp.69-105.

Ragin, C.C. (1987) *The Comparative Method: Moving beyond Qualitative and Quantitative Strategies*. Berkeley: University of California Press.

Ragin, C.C. (2008) *Redesigning Social Inquiry: Fuzzy Sets and Beyond*. Chicago: University of Chicago Press.

Ragin, C.C.; Strand, S.I. (2008) "Using Qualitative Comparative Analysis to Study Causal Order: Comment on Caren and Panofsky (2005)." *Sociological Methods & Research* vol.36, no.4, pp.431-441.

Schneider, C.Q.; Wagemann, C. (2012) *Set-Theoretic Methods for the Social Sciences: A Guide to Qualitative Comparative Analysis (QCA)*. Cambridge: Cambridge University Press.

See Also

minimize

Examples

```
# -----
# Lipset binary crisp data
ttLC <- truthTable(LC, "SURV")

# inspect the truth table
ttLC

# print the cases too, even if not specifically asked for
print(ttLC, show.cases = TRUE)

# the printing function also supports the complete version
print(ttLC, show.cases = TRUE, complete = TRUE)

# formally asking the complete version
truthTable(LC, "SURV", complete = TRUE)

# sorting by multiple columns, decreasing by default
truthTable(LC, "SURV", complete = TRUE, sort.by = "incl, n")

# sort the truth table decreasing for inclusion, and increasing for n
# note that "-" is redundant, sorting is decreasing by default
truthTable(LC, "SURV", complete = TRUE, sort.by = "incl-, n+")


# -----
# Lipset multi-value crisp data (Cronqvist & Berg-Schlosser 2009, p.80)
truthTable(LM, "SURV", sort.by = "incl")

# using a frequency cutoff equal to 2 cases
ttLM <- truthTable(LM, "SURV", n.cut = 2, sort.by = "incl")
ttLM

# the observed combinations coded as remainders
ttLM$removed


# -----
# Cebotari & Vink fuzzy data
ttCVF <- truthTable(CVF, "PROTEST", incl.cut = 0.8, sort.by = "incl")

# view the Venn diagram for this truth table
library(venn)
venn(ttCVF)

# each intersection transparent by its inclusion score
venn(ttCVF, transparency = ttCVF$tt$incl)
```

```
# the truth table negating the outcome
truthTable(CVF, "~PROTEST", incl.cut = 0.8, sort.by = "incl")

# allow contradictions
truthTable(CVF, "PROTEST", incl.cut = c(0.8, 0.75), sort.by = "incl")


# -----
# Ragin and Strand data with temporal QCA
# truth table containing the "-" placeholder as a "don't care"
truthTable(RS, "REC")
```

`Xplot` *Display the distribution of points for a single condition*

Description

This function creates a plot for a single vector of numerical values, arranging them horizontally on the X axis from minimum to maximum.

Usage

```
Xplot(x, jitter = FALSE, at = pretty(x), ...)
```

Arguments

`x`	A numeric vector.
`jitter`	Logical, vertically jitter the points.
`at`	The points at which tick-marks are to be drawn. Non-finite (infinite, NaN or NA) values are omitted. By default, tickmark locations are automatically computed, see the help file for `?pretty`.
`...`	Other graphical parameters from `?par`

Details

This is a special type of (scatter)plot, with points being arranged only on the horizontal axis (it has no vertical axis). Useful when inspecting if points are grouped into naturally occuring clusters, mainly for crisp calibration purposes.

The argument `...` is used to pass arguments to the various graphical parameters from `?par`, and also to the settings from `?jitter`.

The points have a default `cex` (character expansion) value of 1, and a default `pch` value of 1 (empty points), which can be modified accordingly (for instance value 21 for filled points). When `pch = 21`, the color for the margins of the points can be specified via the argument `col`, while the argument `bg` will determine the fill color of the points.

The axis labels have a default `cex.axis` value of 0.8, which affects both the tickmarks labels and the axis labels.

When jittering the points, default values of 0.5 are used for the parameters `factor` and `amount`, on the horizontal axis. More details can be found in the base function `jitter()`.

Although the points are displayed in a single dimension, on the horizontal axis, the R graphical window will still have the default squared shape, with a lot of empty space on the vertical axis. Users are free to create their custom code to determine the size of the graphics window, or simply resize it to a suitable height.

Author(s)

Adrian Dusa

See Also

`par`, `text`, `jitter`

Examples

```
# Lipset's raw data
# plot the DEV (level of developent) causal condition
Xplot(LR$DEV)

# jitter the points vertically
Xplot(LR$DEV, jitter = TRUE)

# clip plotting between the range of min and max
Xplot(LR$DEV, jitter = TRUE, at = range(LR$DEV))
```

XYplot *Create an XY plot*

Description

This function creates an XY plot from the first two columns of a dataframe/matrix, or from two separate vectors of numeric values.

Usage

```
XYplot(x, y, data, relation = "sufficiency", mguides = TRUE,
       jitter = FALSE, clabels, enhance = FALSE, model = FALSE, ...)
```

Arguments

`x`	Character, the name of the column from the `data` for the X axis, or the coordinates of points in the plot (either a matrix/dataframe with at least two columns, or a vector of numerical values for the X axis), or a valid SOP expression.
`y`	Character, the name of the column from the `data` for the Y axis, or the Y coordinates of points in the plot, optional if `x` is a matrix/dataframe.

data	A calibrated dataset, only if x and y are names.
relation	The set relation to Y, either "sufficiency" (default) or "necessity".
mguides	Logical, print the middle guides.
jitter	Logical, jitter the points.
clabels	A vector of case labels with the same length as x and y, or a logical vector of the same length as the number of rows in the data (if provided).
enhance	Logical, if TRUE print the points using different characters for each of the five significant regions for process tracing.
model	Logical, for an enhanced plot specify if the SOP expression in argument x is a solution model.
...	Other graphical parameters from ?par

Details

If x is a dataframe or a matrix, the axes labels will be taken from the column names of x, otherwise they will be inferred from the names of the x and y objects that are passed to this function.

x can also be a string containing either the name of the column for the X axis, or two column names separated by a comma, referring to the X and Y axis respectively. When x contains both X and Y column names, the next argument will be considered as the data.

If data is provided, and the names of the X and Y columns are valid R statements, quoting them is not even necessary and they can be negated using either a tilde "~" or "1 - ".

The numeric values should be restricted between 0 and 1, otherwise an error is generated.

The XY plot will also provide inclusion and coverage scores for a sufficiency (along with PRI) or a necessity relation (along with RoN).

The argument x can also be a SOP - sum of products expression, in which case the relation is determined by the usual forward arrow "=>" for sufficiency and backward arrow "<=" for necessity.

The argument ... is used to pass arguments to the various graphical parameters from ?par, and also to the settings from ?jitter.

The points have a default cex (character expansion) value of 0.8, and a default pch value of 21 (filled points), which can be modified accordingly (for example with value 1 of empty points). When pch = 21, the color for the margins of the points can be specified via the argument col, while the argument bg will determine the fill color of the points.

The axes' labels have a default cex.axis value of 0.8, which affects both the tickmarks labels and the axis labels.

When jittering the points, default values of 0.01 are used for the parameters factor and amount, on both horizontal and vertical axes.

The argument enhance does all the work for the shape of the points and their colors, acording to the five regions specified by Schneider & Rohlfing (2016), who augmented the classical XY plot with process tracing.

The default enhanced XY plot has even more settings when the input SOP expression is a minimization model (different colors, different regions where to place the labels etc.), available by activating the argument model. The model is automatically detected if the input for x is a minimization object.

Value

A list of x and y values, especially useful when the points are jittered.

Author(s)

Adrian Dusa

References

Schneider, C.; Wagemann, C. (2012) *Set-Theoretic Metods for the Social Sciences. A Guide to Qualitative Comparative Analysis*. Cambridge: Cambridge University Press.

Cebotari, V.; Vink, M.P. (2013) "A Configurational Analysis of Ethnic Protest in Europe". *International Journal of Comparative Sociology* vol.54, no.4, pp.298-324.

Schneider, C.; Rohlfing, I. (2016) "Case Studies Nested in Fuzzy-set QCA on Sufficiency. Formalizing Case Selection and Causal Inference". *Sociological Methods and Research* vol.45, no.3, pp.536-568, doi: 10.1177/0049124114532446

See Also

par, text, jitter

Examples

```
# Cebotari & Vink (2013)
# necessity relation between NATPRIDE and PROTEST
XYplot(CVF[, 5:6])

# same using two numeric vectors
XYplot(CVF$NATPRIDE, CVF$PROTEST)

# same using two column names
XYplot(NATPRIDE, PROTEST, data = CVF)

# since they are valid R statements, it works even without quotes
# (this only works in normal R console, not in the GUI version)
XYplot(NATPRIDE, PROTEST, data = CVF)

# negating the X axis, using numeric vectors
XYplot(1 - CVF$NATPRIDE, CVF$PROTEST)

# same thing using quotes
XYplot(1 - NATPRIDE, PROTEST, data = CVF)

# using tilde for negation
XYplot(~NATPRIDE, PROTEST, data = CVF)

# different color for the points
XYplot(~NATPRIDE, PROTEST, data = CVF, col = "blue")
```

```
# using a different character expansion for the axes
XYplot(~NATPRIDE, PROTEST , data = CVF, cex.axis = 0.9)

# custom axis labels
XYplot(~NATPRIDE, PROTEST, data = CVF, xlab = "Negation of NATPRIDE",
       ylab = "Outcome: PROTEST")

# necessity relation
XYplot(~NATPRIDE, PROTEST, data = CVF, relation = "necessity")

# jitter the points
XYplot(~NATPRIDE, PROTEST, data = CVF, jitter = TRUE)

# jitter with more amount
XYplot(~NATPRIDE, PROTEST, data = CVF, jitter = TRUE, amount = 0.02)

# adding labels to points
XYplot(~NATPRIDE, PROTEST, data = CVF, jitter = TRUE, cex = 0.8,
       clabels = rownames(CVF))

# or just the row numbers, since the row names are too long
XYplot(~NATPRIDE, PROTEST, data = CVF, jitter = TRUE, cex = 0.8,
       clabels = seq(nrow(CVF)))


# using a SOP expression (necessity relation)
XYplot(NATPRIDE <- ~PROTEST, data = CVF, jitter = TRUE, cex = 0.8,
       clabels = seq(nrow(CVF)))


#-----
# enhanced XY plot for process tracing
XYplot(~NATPRIDE, PROTEST, data = CVF, enhance = TRUE, jitter = TRUE)


# enhanced XY plot for a solution model
ttCVF <- truthTable(CVF, outcome = PROTEST, incl.cut = 0.85)
pCVF <- minimize(ttCVF, include = "?")
XYplot(pCVF$solution[[1]], PROTEST, data = CVF, enhance = TRUE)


# same plot, using the solution as a SOP expression
XYplot(~NATPRIDE + DEMOC*GEOCON*POLDIS + DEMOC*ETHFRACT*GEOCON,
      PROTEST, data = CVF, enhance = TRUE, model = TRUE)
```

Description

This data set was used by Cebotari and Vink (2013), and it was taken here from the associated replication file Cebotari and Vink (2015).

Usage

```
data(CVR)
data(CVF)
```

Format

A data frame containing 29 cases (ethnic minorities) and the following 6 columns:

DEMOC	Level of democracy: (contextual factor), based on a democracy index ranking countries on a scale from strong autocracies (0) to strong democracies (10). The fuzzy scores were calibrated using an exclusion threshold of 2, a crossover of 7 and an inclusion threshold of 9.5.
ETHFRACT	Degree of ethnic fractionalization: (contextual factor), with raw scores ranging from a homogenous society (0) to a highly fragmented country (1). The fuzzy scores were calibrated using an exclusion threshold of 0, a crossover of 0.495 and an inclusion threshold of 0.8.
GEOCON	Territorial concentration: (group-related factor) with raw data coded as: widely dispersed (0) and primarily urban minorities (1) considered territorially dispersed minorities, and ethnic comunities majoritary in a region (2) and entirely concentrated in one region (3) considered as territorially concentrated minorities. The fuzzy scores were calibrated using an exclusion threshold of 0, a crossover of 1.25 and an inclusion threshold of 3.
POLDIS	Political discrimination: (group-related factor) captures discrimination practices toward minority groups that vary from no discrimination (0) to exclusive and repressive policies toward a minority group (4). The fuzzy scores were calibrated using an exclusion threshold of 0, a crossover of 0.75 and an inclusion threshold of 3.
NATPRIDE	National pride: (group-related factor) with raw scores ranging from 'not at all proud' (0) to 'very proud' (3). The fuzzy scores were calibrated using an exclusion threshold of 0.5, a crossover of 1.5 and an inclusion threshold of 2.5.
PROTEST	Outcome, ethnopolitical protest: measured on a range from 0 to 5 with higher values indicating more intense protest actions. The fuzzy scores were calibrated using an exclusion threshold of 0.5, a crossover of 1.5 and an inclusion threshold of 3.

Details

There are two different versions of the Cebotari and Vink data: CVR contains the raw data, and CVF contains the data calibrated to fuzzy-sets.

References

Cebotari, V.; Vink, M.P. (2013) "A Configurational Analysis of Ethnic Protest in Europe". *International Journal of Comparative Sociology* vol.54, no.4, pp.298-324.

Cebotari, V.; Vink, M.P. (2015) "Replication Data for: A configurational analysis of ethnic protest in Europe", doi: 10.7910/DVN/PT2IB9, Harvard Dataverse, V2

`_Hino` *Time-Difference*

Description

This data set was used by Hino (2009), to demonstrate the Time-Difference QCA.

Usage

```
data(HC)
```

Format

A data frame containing 15 cases (countries) and the following 5 columns:

`FOREIGN`	Percentage of foreign population.
`UNEMP`	Percentage of unemployed population.
`CONV`	Party system convergence.
`PRES80`	Presence of extreme-right parties in 1980s.
`VOTE`	Outcome, vote share of extreme-right parties.

Details

For all columns in the data, a value of 1 means a positive difference between 1990 and 1980, and a value of 0 means negative or zero difference, except for the condition CONV, which is the inverse of the condition DIVERT in the raw data. The condition PRES80 does not have a time difference, it represents a simple presence / absence of extreme-right parties in the 1980s.

References

Hiro, A. (2009) "Time-Series QCA. Studying Temporal Change through Boolean Analysis". *Sociological Theory and Methods*, vol.24, no.2, pp.247-265.

`_Legacy datasets` *Legacy datasets*

Description

The following datasets are no longer part of this package in the formal documentation, but have been added to ensure backwards compatibility with prior publications.

Usage

```
data(d.AS)
data(d.Bas)
data(d.biodiversity)
data(d.BWB)
data(d.CS)
data(d.CZH)
data(d.education)
data(d.Emm)
data(d.graduate)
data(d.health)
data(d.HK)
data(d.HMN)
data(d.homeless)
data(d.jobsecurity)
data(d.Kil)
data(d.Kro)
data(d.napoleon)
data(d.partybans)
data(d.represent)
data(d.RS)
data(d.SA)
data(d.socialsecurity)
data(d.SS)
data(d.stakeholder)
data(d.transport)
data(d.urban)
data(Emme)
data(HarKem)
data(Krook)
data(RagStr)
data(Rokkan)
```

_Lipset *Lipset's indicators for the survival of democracy during the inter-war period.*

Description

This dataset is taken from Lipset (1959), as used by Rihoux and De Meur (2009), Cronqvist and Berg-Schlosser (2009) and Ragin (2009).

Usage

```
data(LR)
data(LC)
data(LM)
data(LF)
```

Format

A data frame containing 18 rows and the following 6 columns:

- `DEV` Level of development: it is the GDP per capita (USD) in the raw data, calibrated in the binary crisp version to 0 if below 550 USD and 1 otherwise. For the multi-value crisp version, two thresholds were used: 550 and 850 USD.
- `URB` Level of urbanization: percent of the population in towns with 20000 or more inhabitants, calibrated in the crisp versions to 0 if below 50% and 1 if above.
- `LIT` Level of literacy: percent of the literate population, calibrated in the crisp versions to 0 if below 75% and 1 if above.
- `IND` Level of industrialization: percent of the industrial labor force, calibrated in the crisp versions to 0 if below 30% and 1 if above.
- `STB` Government stability: a "political-institutional" condition added to the previous four "socioeconomic" ones. The raw data has the number of cabinets which governed in the period under study, calibrated in the crisp versions to 0 if 10 or above and to 1 if below 10.
- `SURV` Outcome: survival of democracy during the inter-war period: calibrated to 0 if negative, and 1 if positive raw data.

Details

There are four different versions of the Lipset data:

- `LR` contains the raw data
- `LC` is the same data calibrated to binary crisp sets
- `LM` is calibrated to multi-value sets
- `LF` is calibrated to fuzzy-sets

References

Lipset, S. M. (1959) "Some Social Requisites of Democracy: Economic Development and Political Legitimacy", *American Political Science Review* vol.53, pp.69-105.

Cronqvist, L.; Berg-Schlosser, D. (2009) "Multi-Value QCA (mvQCA)", in Rihoux, B.; Ragin, C. (eds.) *Configurational Comparative Methods. Qualitative Comparative Analysis (QCA) and Related Techniques*, SAGE.

Rihoux, B.; De Meur, G. (2009) "Crisp Sets Qualitative Comparative Analysis (mvQCA)", in Rihoux, B.; Ragin, C. (eds.) *Configurational Comparative Methods. Qualitative Comparative Analysis (QCA) and Related Techniques*, SAGE.

Ragin, C. (2009) "Qualitative Comparative Analysis Using Fuzzy-Sets (fsQCA)", in Rihoux, B.; Ragin, C. (eds.) *Configurational Comparative Methods. Qualitative Comparative Analysis (QCA) and Related Techniques*, SAGE.

_Nieuwbeerta *Class voting in post-World War era*

Description

This fuzzy dataset is an adaptation from Ragin (2005, 2008), the data itself being attributed to Nieuwbeerta (1995).

Usage

```
data(NF)
```

Format

A data frame containing 12 cases (countries) and the following 5 columns:

- `A` degree of membership in the set of highly *affluent* countries
- `I` degree of membership in the set of countries with substantial levels of income *inequality*
- `M` degree of membership in the set of countries with a high percentage of workers employed in *manufacturing*
- `U` degree of membership in the set of countries with strong *unions*
- `W` outcome: degree of membership in the set of countries with *weak* class voting

Details

All fuzzy sets in this data are constructed on a six-values scale, for demonstrative purposes.

In the original dataset, the outcome `W` is presented as the first column.

References

Nieuwbeerta, P. (1995) *The Democratic Class Struggle in Twenty Countries: 1945:1990*. Amsterdam: Thesis Publishers.

Ragin, C.C. (2005) "From fuzzy sets to crisp truth tables". WP 2004-28, COMPASSS Working Papers series.

Ragin, C.C. (2008) *Redesigning Social Inquiry: Fuzzy Sets and Beyond*. Chicago: University of Chicago Press.

_Ragin and Strand *University recognition of a graduate student union*

Description

Original data used by Caren and Panofsky (2005), and reanalysed by Ragin and Strand (2008).

Usage

```
data(RS)
```

Format

A data frame containing 17 cases and the following 6 columns:

P	Public university
E	Support of elite allies
A	National union affiliation
S	Strike or a strike threat
EBA	**E** happens before **A**
REC	Union recognition

Details

The causal conditions are P, E, A and S. All of them are binary crisp with two values: 0 = No and 1 = Yes.

The column EBA is not a causal condition, specifying in which case the causal condition E happens before the causal condition A. It has two values (0 = No and 1 = Yes) plus the placeholder "-" to signal a "don't care".

The outcome is the union recognition EBA, also binary crisp with two values: 0 = No and 1 = Yes.

Source

Caren, N; Panofsky, A. (2005) "TQCA: A Technique for Adding Temporality to Qualitative Comparative Analysis." *Sociological Methods & Research* vol. 34, no.2, pp.147-172, doi: 10.1177/0049124105277197.

Ragin, C.C.; Strand, S.I. (2008) "Using Qualitative Comparative Analysis to Study Causal Order: Comment on Caren and Panofsky (2005)." *Sociological Methods & Research* vol.36, no.4, pp.431-441, doi: 10.1177/0049124107313903.

Index