Package 'votesys'

April 20, 2018

Type Package

Title Voting Systems, Instant-Runoff Voting, Borda Method, Various Condorcet Methods

Version 0.1.1

Date 2018-04-20

Maintainer Jiang Wu <textidea@sina.com>

Description Various methods to count ballots in voting systems are provided. Functions to check validity of ballots are also provided to ensure flexibility.

License GPL-3

Depends R (>= 3.3.0)

Imports data.table, gtools, Matrix

Encoding UTF-8

LazyLoad true

LazyData true

RoxygenNote 6.0.1

NeedsCompilation no

Author Jiang Wu [aut, cre] (from Capital Normal University)

Repository CRAN

Date/Publication 2018-04-20 09:56:40 UTC

R topics documented:

---

votesys-package *Voting Systems, Instant-Runoff Voting, Borda Method, Various Condorcet Methods*

Description

This package provides different methods for counting ballots, which can be used in election, decision making and evaluation. The basic idea is: different forms of ballots can all be transformed into a score matrix; then the score matrix can be put into different counting methods. The functions in this package provide more flexibility to deal with duplicated values (ties) and missing values. And the comparison of results of different methods is also made easy.

Author(s)

Jiang Wu

Examples

```
# Suppose we have the following ballot data
raw <- list2ballot(
    x = list(
        c('m', 'n', 'c', 'k'), c('n', 'c', 'k', 'm'),
        c('c', 'k', 'n', 'm'), c('k', 'c', 'n', 'm'), c(NA, NA, NA, NA)
    ) ,
    n = c(42, 26, 15, 17, 3)
)

# Step 1: check validity of ballots. Delete
# some of them, if needed.
check_validity <- check_dup_wrong(raw,
    xtype = 3,
    candidate = c("m", "n", "k", "c")
)
raw <- raw[- check_validity$row_all_na]

# Step 2: create a vote object
vote <- create_vote(raw, xtype = 3, candidate = c("m", "n", "k", "c"))
```

```
# Step 3: use one or more methods
y <- plurality_method(vote) # winner is m
y <- irv_method(vote) # winner is k
y <- cdc_simple(vote) # winner is n
y <- cdc_rankedpairs(vote) # winner is n
```

`approval_method` *Approval Method*

Description

In approval method, each voter is required to mention one or more candidates, and the winner is the one who gets the top frequency. For this function, a ballot with candidates more than required and different scores is also valid. For a score matrix, the function will check the positions j, k...which have the lowest scores (in a `vote` object, the lower, the better) in the ith row. However, the function will first check the `approval_able` element of the `vote` object. If it is FALSE, the winner will be NULL.

Usage

```
approval_method(x, min_valid = 1, n)
```

Arguments

`x`	an object of class `vote`.
`min_valid`	default is 1. If the number of valid entries of a ballot is less than this value, the ballot will not be used.
`n`	the number of candidates written down by a voter should not larger than this value.

Value

a list object.

- (1) `call` the function call.
- (2) `method` the counting method.
- (3) `candidate` candidate names.
- (4) `candidate_num` number of candidate.
- (5) `ballot_num` number of ballots in x.
- (6) `valid_ballot_num` number of ballots that are used to compute the result.
- (7) `winner` the winners, may be one, more than one or NULL.
- (8) `n` equal to the argument `n`.
- (9) `other_info` frequencies of candidates mentioned by voters.

Examples

```
raw <- matrix(NA, nrow = 22, ncol = 5)
for (i in 1: 20){
   set.seed(i)
   raw[i, ] <- sample(c(1: 5, NA, NA, NA), 5)
}
raw[21, ] <- c(4, 5, 3, 1, 2)
raw[22, ] <- c(3, 5, 1, 2, 4)
vote <- create_vote(raw, xtype = 1)
y <- approval_method(vote, n = 3)
y <- approval_method(vote, n = 3, min_valid = 5)
y <- approval_method(vote, n = 4, min_valid = 3)
```

---

as_complete *Convert Incomplete ranking/rating matrix into full matrix*

Description

This function deals with incomplete ranking and rating matrix (e. g., created by `create_vote` and stored in `$ballot`), so as to convert it into full ranking and rating. In each row of the score matrix, the smallest value represents the most preferred and the biggest value represents the most hated. For the methods used by this function, see Details. See Examples for how to modify an object of class vote created with incomplete data.

Usage

```
as_complete(x, method = c("valid", "max", "len"), plus = 0, n = NULL)
```

Arguments

`x`	the score matrix, should be a matrix, data.frame, or data.table.
`method`	see Details, default is "valid".
`plus`	see Details, default is 0.
`n`	see Details, default is 0.

Details

Three methods are used and you should choose according to your need.

- (1) "valid": the default method. For the vector `c(3, 1, 2, 2, NA, NA)`, as there should be 6 values but only 4 are given, 4 is the valid number, and the NAs will be converted to 4. However, if the argument `plus` is a value other than 0, than NAs will be equal to the valid number plus that value. For example, if `plus = 10`, the NAs will be 14 (4 + 10).
- (2) "max": the maximum value in each row plus the value given by `plus`. So for `c(3, 1, 2, 2, NA, NA)`, and `plus = 0`, NAs will be 3 (3 + 0).
- (3) "len": In the case of topKlist, interviewees may, for example, choose 4 or 5 items from a 20-item list. When the method is "len", use `n` to indicate the total number of items or any other number. The default value of `n` is `ncol(x)`, which is equivalent to the way `create_vote` used to convert NAs so as to calculate the Condorcet matrix.

Value

Always a matrix. NAs are converted to numbers. However, if all entries in a row of the input data are NAs, then that row will NOT be modified. NOTE: the order of the returned matrix (the 1st row, the 2nd row, the 3rd row, etc) is DIFFERENT from the input data.

Examples

```
raw <- list2ballot(string = c("1: a, b, c", "2: b, c", "3: a, b"))
vote <- create_vote(raw, xtype = 3, candidate = c("a", "b", "c"))
ballot <- as_complete(vote$ballot, method = "max", plus = 5)
ballot <- as_complete(vote$ballot, method = "len", n = 10)
# Now re-create the vote object
vote <- create_vote(ballot, xtype = 1)

m <- matrix(c(
    1, 2, 3, NA, NA, NA,
    1, 1.1, 2.2, 8.8, NA, NA,
    1, 1.1, 2.2, 8.8, NA, NA,
    1, 1.1, 2.2, 8.8, NA, NA,
    1, 1.1, 2.2, 8.8, NA, NA,
    NA, NA, NA, NA, NA, NA,
    3, 2, NA, NA, NA, NA,
    3, 2, NA, NA,NA,NA,
    1, 2, 3, 4, 5, 6), ncol = 6, byrow = TRUE)
colnames(m) <- LETTERS[1: 6]
y <- as_complete(m, method = "valid", plus = 30)
```

---

borda_method *Borda Count Method*

Description

Both ordinary Borda method and modified Borda method are available. In an ordinary Borda system, voters are required to assign score values to candidates. See Details.

Usage

```
borda_method(x, allow_dup = TRUE, min_valid = 1, modified = FALSE)
```

Arguments

x	an object of class vote.
allow_dup	whether ballots with duplicated score values are taken into account. Default is TRUE.
min_valid	default is 1. If the number of valid entries of a ballot is less than this value, the ballot will not be used.
modified	if the modified Borda is to be used. Default is FALSE.

Details

Suppose there are 5 candidates. A voter's 1st choice gets 1 point, the 2nd choice gets 2 points... Candidate with the smallest total score wins. The function does not require voters to assign scores to all candidates, for NAs are automatically assigned the highest (worst) score. Duplicated values (two or more candidates share the same score) are also allowed (note: NAs and ties may not be allowed in real ballots).

In modified Borda, the rule changes. Suppose there are 5 candidates. A voter writes down 5 candidates and his 1st choice gets 5 points. The one who gets the largest total score wins. However, if the voter only write down 2 names, then, his 1st choice gets only 2 points rather than 5 points. Thus the modified Borda encourages voters to write down more names. Besides, in modified Borda, only the ranks of true scores, rather than the true scores themselves, are used. If the raw data is a list each ballot of which contains candidate names, scores can also be extracted, that is, the 1st position is the 1st choice which gets 1 point, the 2nd position, 2 points, and so on.

Value

a list object.

- (1) `call` the function call.
- (2) `method` the counting method.
- (3) `candidate` candidate names.
- (4) `candidate_num` number of candidate.
- (5) `ballot_num` number of ballots in x.
- (6) `valid_ballot_num` number of ballots that are used to compute the result.
- (7) `winner` the winners.
- (8) `modified` whether the modified Borda is used.
- (9) `other_info` a list with 2 elements, if `modified` is FALSE, then `count_min` records the total scores, `count_max` is NULL; if `modified` is TRUE, the vice versa.

Examples

```
raw <- c(
    rep(c('m', 'n', 'c', 'k'), 42),
    rep(c('n', 'c', 'k', 'm'), 26),
    rep(c('c', 'k', 'n', 'm'), 15),
    rep(c('k', 'c', 'n', 'm'), 17)
)
raw <- matrix(raw, ncol = 4, byrow = TRUE)
vote <- create_vote(raw, xtype = 2, candidate = c('m', 'n', 'c', 'k'))
y <- borda_method(vote)

raw <- list(c('a', 'e', 'c', 'd', 'b'), c('b', 'a', 'e'),
    c('c', 'd', 'b'), c('d', 'a', 'e')
)
vote <- create_vote(raw, xtype = 3, candidate = c('a', 'b', 'c', 'd', 'e'))
y <- borda_method(vote, modified = TRUE)
```

cdc_copeland *Copeland Method*

Description

Candidates enter into pairwise comparison. if the number of voters who prefer a is larger than the number of voters who prefer b, then a wins b, a gets 1 point, b gets -1 point. If the numbers are equal, then both of them gets 0 point. Then, sum up each one's comparison points. For example, a wins 3 times, loses 1 time, has equal votes with 2 candidate, his score is 3 * 1 + (-1) * 1 + 0 * 2 = 2. The one gets the most points wins. Essentially, this is a way to solve ties in ordinary Condorcet method. However, there may be 2 or more winners. The other type of Copeland method is to count only the times of wins, that is, the loser in pairwise comparison gets 0 point rather than -1 point.

Usage

```
cdc_copeland(x, allow_dup = TRUE, min_valid = 1, lose = -1)
```

Arguments

x	it accepts the following types of input: 1st, it can be an object of class vote. 2nd, it can be a user-given Condorcet matrix, 3rd, it can be a result of another Condorcet method, which is of class condorcet.
allow_dup	whether ballots with duplicated score values are taken into account. Default is TRUE.
min_valid	default is 1. If the number of valid entries of a ballot is less than this value, it will not be used.
lose	the point the pairwise loser gets, should be -1 (default) or 0.

Value

a condorcet object, which is essentially a list.

- (1) call the function call.
- (2) method the counting method.
- (3) candidate candidate names.
- (4) candidate_num number of candidate.
- (5) ballot_num number of ballots in x. When x is not a vote object, it may be NULL.
- (6) valid_ballot_num number of ballots that are actually used to compute the result. When x is not a vote object, it may be NULL.
- (7) winner the winners.
- (8) input_object the class of x.
- (9) cdc the Condorcet matrix which is actually used.
- (10) dif the score difference matrix. When x is not a vote object, it may be NULL.

- (11) `binary` win and loss recorded with 1 (win), 0 (equal) and -1 (loss).
- (12) `summary_m` times of win (1), equal (0) and loss (-1).
- (13) `other_info` a list with 2 elements, the 1st is the point the loser gets, it is equal to `lose`. The 2nd contains the scores.

References

- Merlin, V. & Saari, D. 1996. The Copeland method: I.: Relationships and the dictionary. Economic Theory, 8(1), 51-76.

Examples

```
raw <- c(
    rep(c('m', 'n', 'c', 'k'), 42), rep(c('n', 'c', 'k', 'm'), 26),
    rep(c('c', 'k', 'n', 'm'), 15), rep(c('k', 'c', 'n', 'm'), 17)
)
raw <- matrix(raw, ncol = 4, byrow = TRUE)
vote <- create_vote(raw, xtype = 2, candidate = c('m', 'n', 'k', 'c'))
win1 <- cdc_simple(vote)
win2 <- cdc_copeland(vote) # winner is n
win2 <- cdc_copeland(win1$cdc)
win3 <- cdc_copeland(win2, lose = 0)
```

---

cdc_dodgson *Dodgson Method*

Description

The original Dodgson method checks the number of votes each candidate has to rob from other candidates; the winner is with the smallest number. However, the function `cdc_dodgson` uses two alternative methods rather than the original Dodgson method. The two methods are Tideman score method and Dodgson Quick method. See Details.

Usage

```
cdc_dodgson(x, allow_dup = TRUE, min_valid = 1, dq_t = "dq")
```

Arguments

`x`	it accepts the following types of input: 1st, it can be an object of class `vote`. 2nd, it can be a user-given Condorcet matrix, 3rd, it can be a result of another Condorcet method, which is of class `condorcet`.
`allow_dup`	whether ballots with duplicated score values are taken into account. Default is TRUE.
`min_valid`	default is 1. If the number of valid entries of a ballot is less than this value, it will not be used.
`dq_t`	the alternative Dodgson methods to be used. Default is "dq", for Dodgson Quick method; it can also be "t", Tideman score method.

Details

Suppose the candidates are A, B, C and D. If A wins B in pairwise comparison or has equal votes with B, then add 0 to A. If C wins A, then add to A adv(C, A), that is, the number of voters that prefer C than A, minus the number of voters that prefer A than A. Again, if D wins A, then add to A that number. Then, we sum up the points belong to A. We do the same thing to B, C and D. The one gets the least points is the winner. This is what we do in Tideman score method. In Dodgson Quick method, we first compute the number of votes, then divide it by 2 and get the ceiling, and sum all of them up.

Value

a `condorcet` object, which is essentially a list.

- (1) `call` the function call.
- (2) `method` the counting method.
- (3) `candidate` candidate names.
- (4) `candidate_num` number of candidate.
- (5) `ballot_num` number of ballots in x. When x is not a `vote` object, it may be NULL.
- (6) `valid_ballot_num` number of ballots that are actually used to compute the result. When x is not a `vote` object, it may be NULL.
- (7) `winner` the winners.
- (8) `input_object` the class of x.
- (9) `cdc` the Condorcet matrix which is actually used.
- (10) `dif` the score difference matrix. When x is not a `vote` object, it may be NULL.
- (11) `binary` win and loss recorded with 1 (win), 0 (equal) and -1 (loss).
- (12) `summary_m` times of win (1), equal (0) and loss (-1).
- (13) `other_info` a list with four elements. The 1st indicates the method used to compute score. The 2nd is the score for pairwise comparison (number of votes one has to rob). The 3rd is Tideman score summary (the smaller the better). The 4th is Dodgson Quick summary (the smaller the better).

References

- McCabe-Dansted, J. & Slinko, A. 2008. Approximability of Dodgson's Rule. Social Choice and Welfare, Feb, 1-26.

Examples

```
raw <- list2ballot(
    x = list(
        c('A', 'B', 'C', 'D', 'E', 'F'),
        c('F', 'A', 'B', 'C', 'D', 'E'),
        c('E', 'D', 'C', 'B', 'F', 'A'),
        c('B', 'A', 'C', 'D', 'E', 'F'),
        c('F', 'E', 'D', 'C', 'B', 'A'),
        c('F', 'B', 'A', 'C', 'D', 'E'),
```

```
        c('E', 'D', 'C', 'A', 'F', 'B'),
        c('E', 'B', 'A', 'C', 'D', 'F'),
        c('F', 'D', 'C', 'A', 'E', 'B'),
        c('D', 'B', 'A', 'C', 'E', 'F'),
        c('F', 'E', 'C', 'A', 'D', 'B')
    ),
    n = c(19, 12, 12, 9, 9, 10, 10 , 10 , 10, 10, 10)
)
vote <- create_vote(raw, xtype = 3, candidate = c('A', 'B', 'C', 'D', 'E', 'F'))
win1 <- cdc_simple(vote) # no winner
win2 <- cdc_dodgson(vote, dq_t = "dq") # A
win2 <- cdc_dodgson(win1, dq_t = "dq") # A
win3 <- cdc_dodgson(vote, dq_t = "t") # B
win3 <- cdc_dodgson(win2, dq_t = "t") # B
```

`cdc_kemenyyoung` *Kemeny-Young Method*

Description

Kemeny-Young method first lists all the permutations of candidates, that is, all possible orders, or possible ordered links. Then, it computes the sums of strength of these links. The top link is the one with the highest strength score, and the winner is the first one in this link. Currently, the maximum candidate number is 8 for speed and memory reasons.

Usage

```
cdc_kemenyyoung(x, allow_dup = TRUE, min_valid = 1, margin = FALSE,
  keep_all_link = FALSE)
```

Arguments

`x`	it accepts the following types of input: 1st, it can be an object of class `vote`. 2nd, it can be a user-given Condorcet matrix, 3rd, it can be a result of another Condorcet method, which is of class `condorcet`.
`allow_dup`	whether ballots with duplicated score values are taken into account. Default is TRUE.
`min_valid`	default is 1. If the number of valid entries of a ballot is less than this value, it will not be used.
`margin`	if it is FALSE (default), the values in Condorcet matrix are used, that is: if A vs. B is 30, B vs. A is 18, then 30 and 18 are used to calculate link strength; if it is TRUE, then 30 - 18 = 12 and -12 are used.
`keep_all_link`	if TRUE, the result will store all the links and their strength. However, it is quite memory-costing, so the default is FALSE.

Value

a condorcet object, which is essentially a list.

- (1) call the function call.
- (2) method the counting method.
- (3) candidate candidate names.
- (4) candidate_num number of candidate.
- (5) ballot_num number of ballots in x. When x is not a vote object, it may be NULL.
- (6) valid_ballot_num number of ballots that are actually used to compute the result. When x is not a vote object, it may be NULL.
- (7) winner the winner.
- (8) input_object the class of x.
- (9) cdc the Condorcet matrix which is actually used.
- (10) dif the score difference matrix. When x is not a vote object, it may be NULL.
- (11) binary win and loss recorded with 1 (win), 0 (equal) and -1 (loss).
- (12) summary_m times of win (1), equal (0) and loss (-1).
- (13) other_info a list with 3 elements. win_link is the link with the highest strength. Note: it is a matrix, maybe with 2 or more rows. win_link_value is the strength of the link. all_link is NULL when keep_all_link is FALSE. if TRUE, it stores all the links and scores sorted by scores in decreasing order (this costs much memory on your computer).

References

- Young, H. & Levenglick, A. 1978. A consistent extension of Condorcet's election principle. Society for Industrial and Applied Mathematics, 35(2), 285-300.

Examples

```
m <- matrix(c(0, 58, 58, 58, 42, 0, 32, 32, 42, 68, 0, 17, 42, 68, 83, 0), nr = 4)
colnames(m) <- c('m', 'n', 'c', 'k')
rownames(m) <- c('m', 'n', 'c', 'k')
y <- cdc_kemenyyoung(m, keep_all_link = TRUE) # winner is n
```

---

cdc_minmax *Minmax Method*

Description

Minmax method (also known as Simpson-Kramer method, successive reversal method) means three different methods. The first is winning votes method. In pairwise comparison, if a wins b, a gets 0 point, the number of points for b is the number of voters who prefer a than b. The second method is to use margins. In pairwise comparison, a gets b - a points and b gets a - b points. The third method is pairwise opposition method. The number of points for a is the number of voters who prefer b than a; the number of points for b is the number of voters who prefer a than b. Although

the point-assigning methods are different for the above three methods, they nonetheless do the same thing: to check to what extent one candidate is defeated by others. So the summarizing method is the same: for each candidate, we extract the maximum target points, and the one with the minimum points wins.

Usage

```
cdc_minmax(x, allow_dup = TRUE, min_valid = 1, variant = 1)
```

Arguments

x	it accepts the following types of input: 1st, it can be an object of class vote. 2nd, it can be a user-given Condorcet matrix, 3rd, it can be a result of another Condorcet method, which is of class condorcet.
allow_dup	whether ballots with duplicated score values are taken into account. Default is TRUE.
min_valid	default is 1. If the number of valid entries of a ballot is less than this value, it will not be used.
variant	should be 1, 2 or 3. 1 (default) for winning votes method, 2 for margins method, 3 for pairwise comparison method.

Value

a condorcet object, which is essentially a list.

- (1) call the function call.
- (2) method the counting method.
- (3) candidate candidate names.
- (4) candidate_num number of candidate.
- (5) ballot_num number of ballots in x. When x is not a vote object, it may be NULL.
- (6) valid_ballot_num number of ballots that are actually used to compute the result. When x is not a vote object, it may be NULL.
- (7) winner the winners.
- (8) input_object the class of x.
- (9) cdc the Condorcet matrix which is actually used.
- (10) dif the score difference matrix. When x is not a vote object, it may be NULL.
- (11) binary win and loss recorded with 1 (win), 0 (equal) and -1 (loss).
- (12) summary_m times of win (1), equal (0) and loss (-1).
- (13) other_info a list of 4 elements. The 1st is the method, which is equal to variant. The 2nd is the winning votes matrix. The 3rd is the margins matrix. The 4th is the pairwise comparison matrix.

References

- https://en.wikipedia.org/wiki/Minimax_Condorcet_method

Examples

```
raw <- c(
    rep(c('m', 'n', 'c', 'k'), 42), rep(c('n', 'c', 'k', 'm'), 26),
    rep(c('c', 'k', 'n', 'm'), 15), rep(c('k', 'c', 'n', 'm'), 17)
)
raw <- matrix(raw, ncol = 4, byrow = TRUE)
vote <- create_vote(raw, xtype = 2, candidate = c('m', 'n', 'k', 'c'))
win1 <- cdc_simple(vote)
win2 <- cdc_minmax(vote) # winner is n
win3 <- cdc_minmax(win1, variant = 2)
win4 <- cdc_minmax(win3$cdc, variant = 3)
```

---

cdc_rankedpairs *Ranked Pairs Method*

Description

It is also called Tideman method. See details.

Usage

```
cdc_rankedpairs(x, allow_dup = TRUE, min_valid = 1)
```

Arguments

x	it accepts the following types of input: 1st, it can be an object of class vote. 2nd, it can be a user-given Condorcet matrix, 3rd, it can be a result of another Condorcet method, which is of class condorcet.
allow_dup	whether ballots with duplicated score values are taken into account. Default is TRUE.
min_valid	default is 1. If the number of valid entries of a ballot is less than this value, it will not be used.

Details

The method first summarizes the result of pairwise comparison, the order used is the order of winning votes from large to small. So if pairwise comparison has ties (that is, the number of voters who prefer a than b is equal to the number of voters who prefer b than a, the method will fail, and the winner will be NULL).

The second step is called tally. If a wins b with 100 votes, b wins c with 80 votes, then we put a-b-100 ahead of b-c-80. Suppose a wins b with 100 votes, a wins c with 100 votes, then we have a tie; so we have to check the relation between b and c. If b wins c, then we put a-c-100 ahead of a-b-100. Suppose a wins b with 100 votes, d wins b with 100 votes, then again we have a tie and have to check the a-d relation. If d wins a, then we put d-b-100 ahead of a-b-100. Suppose a wins b with 100 votes, e wins f with 100 votes, then the ties cannot be solved, so the winner will be NULL.

The third step, after the above mentioned tally, is called lock-in. As the relations have been sorted according to their strength from large to small in the tally step, we now add them one by one. The

rule is: if a relation is contradictory with those already locked in relations, this relation will be discarded.

For example, suppose we have already add relation a > b and b > c, then the two relations are locked in. As a result, we should not add b > a. Also, as a > b and b > c indicate a > c, so we should not add c > a. After this process, we will finally find the winner who defeats all others.

Value

a `condorcet` object, which is essentially a list.

- (1) `call` the function call.
- (2) `method` the counting method.
- (3) `candidate` candidate names.
- (4) `candidate_num` number of candidate.
- (5) `ballot_num` number of ballots in x. When x is not a `vote` object, it may be NULL.
- (6) `valid_ballot_num` the number of ballots that are actually used to compute the result. When x is not a `vote` object, it may be NULL.
- (7) `winner` the winner, may be NULL.
- (8) `input_object` the class of x.
- (9) `cdc` the Condorcet matrix which is actually used.
- (10) `dif` the score difference matrix. When x is not a `vote` object, it may be NULL.
- (11) `binary` win and loss recorded with 1 (win), 0 (equal) and -1 (loss).
- (12) `summary_m` times of win (1), equal (0) and loss (-1).
- (13) `other_info` a list of 3 elements. The 1st is the reason of failure. If winner exists, it will be blank. The 2nd is the tally result (it may contain unsolved ties). The 3rd is the lock-in result; if the method fails, it will be NULL.

References

- Tideman, T. 1987. Independence of clones as a criterion for voting rules. Social Choice and Welfare, 4(3), 185-206.

Examples

```
raw <- rbind(c('m', 'n', 'c', 'k'), c('n', 'c', 'k', 'm'),
    c('c', 'k', 'n', 'm'), c('k', 'c', 'n', 'm'))
raw <- list2ballot(m = raw, n = c(42, 26, 15, 17))
vote <- create_vote(raw, xtype = 2, candidate = c('m', 'n', 'c', 'k'))
y <- cdc_rankedpairs(vote)
```

cdc_schulze *Schulze Method*

Description

Schulze method is essentially a widest path problem. With the Condorcet matrix, we must find the so called the strongest path a > b > c > d, and the winner is a. The strength of a path is the strength of its weakest link.

Usage

```
cdc_schulze(x, allow_dup = TRUE, min_valid = 1)
```

Arguments

x	it accepts the following types of input: 1st, it can be an object of class vote. 2nd, it can be a user-given Condorcet matrix, 3rd, it can be a result of another Condorcet method, which is of class condorcet.
allow_dup	whether ballots with duplicated score values are taken into account. Default is TRUE.
min_valid	default is 1. If the number of valid entries of a ballot is less than this value, it will not be used.

Value

a condorcet object, which is essentially a list.

- (1) call the function call.
- (2) method the counting method.
- (3) candidate candidate names.
- (4) candidate_num number of candidate.
- (5) ballot_num number of ballots in x. When x is not a vote object, it may be NULL.
- (6) valid_ballot_num number of ballots that are actually used to compute the result. When x is not a vote object, it may be NULL.
- (7) winner the winners, may be NULL.
- (8) input_object the class of x.
- (9) cdc the Condorcet matrix which is actually used.
- (10) dif the score difference matrix. When x is not a vote object, it may be NULL.
- (11) binary win and loss recorded with 1 (win), 0 (equal) and -1 (loss).
- (12) summary_m times of win (1), equal (0) and loss (-1).
- (13) other_info a list of 2 elements. The 1st is the strength comparison matrix. The 2nd is the strength comparison matrix in binary mode, 1 for win, 0 for else.

References

- Schulze, M. 2010. A new monotonic, clone-independent, reversal symmetric, and Condorcet-consistent single-winner election method. Social Choice and Welfare, 36(2), 267-303.

Examples

```
raw <- list2ballot(
    x = list(
        c('a', 'c', 'b', 'e', 'd'),
        c('a', 'd', 'e', 'c', 'b'),
        c('b', 'e', 'd', 'a', 'c'),
        c('c', 'a', 'b', 'e', 'd'),
        c('c', 'a', 'e', 'b', 'd'),
        c('c', 'b', 'a', 'd', 'e'),
        c('d', 'c', 'e', 'b', 'a'),
        c('e', 'b', 'a', 'd', 'c')
    ),
    n = c(5, 5, 8, 3, 7, 2, 7, 8)
)
vote <- create_vote(raw, xtype = 3, candidate = c('a', 'b', 'c', 'd', 'e'))
win1 <- cdc_simple(vote) # no winner
win2 <- cdc_schulze(vote) # winner is e
win2 <- cdc_schulze(win1)
```

---

cdc_simple *Ordinary Condorcet Method*

Description

Candidates enter into pairwise comparison. if the number of voters who prefer a is larger than the number of voters who prefer b, then a wins b, a gets 1 point, b gets 0 point. If the numbers are equal, then both of them gets 0 point. Suppose there are n candidates, the one gets n-1 points wins (that is, he wins in all pairwise comparison). There may be no Condorcet winner. If thus, you can try other Condorcet family methods.

Usage

```
cdc_simple(x, allow_dup = TRUE, min_valid = 1)
```

Arguments

x	it accepts the following types of input: 1st, it can be an object of class vote. 2nd, it can be a user-given Condorcet matrix, 3rd, it can be a result of another Condorcet method, which is of class condorcet.
allow_dup	whether ballots with duplicated score values are taken into account. Default is TRUE.
min_valid	default is 1. If the number of valid entries of a ballot is less than this value, it will not be used.

Value

a condorcet object, which is essentially a list.

- (1) call the function call.
- (2) method the counting method.
- (3) candidate candidate names.
- (4) candidate_num number of candidate.
- (5) ballot_num number of ballots in x. When x is not a vote object, it may be NULL.
- (6) valid_ballot_num number of ballots that are actually used to compute the result. When x is not a vote object, it may be NULL.
- (7) winner the winner; may be NULL.
- (8) input_object the class of x.
- (9) cdc the Condorcet matrix which is actually used.
- (10) dif the score difference matrix. When x is not a vote object, it may be NULL.
- (11) binary win and loss recorded with 1 (win), 0 (equal) and -1 (loss).
- (12) summary_m times of win (1), equal (0) and loss (-1).
- (13) other_info currently nothing.

Examples

```
raw <- c(
    rep(c('m', 'n', 'c', 'k'), 42), rep(c('n', 'c', 'k', 'm'), 26),
    rep(c('c', 'k', 'n', 'm'), 15), rep(c('k', 'c', 'n', 'm'), 17)
)
raw <- matrix(raw, ncol = 4, byrow = TRUE)
vote <- create_vote(raw, xtype = 2, candidate = c('m', 'n', 'k', 'c'))
win1 <- cdc_simple(vote) # winner is n
win2 <- cdc_simple(win1$cdc) # use a Condorceit matrix
win2 <- cdc_simple(win1) # use an existent result
```

---

check_dup_wrong *Check Ballots with Duplicated Values, Mistakes, or without Any Valid Entry*

Description

The function simply checks validity of ballots and shows the check result. If you want a one-step clean, set clean to TRUE and a set of cleaned ballots will be returned. Here, duplicated values mean that the voter write the same candidate more than one time, or, when he assigns scores, he assigns the same score to more than one candidates. Mistakes are names that do not appear in the candidate list, or score values that are illegal (e.g., if voters are required to assign 1-5 to candidates, then 6 is an illegal value). Ballots without a valid entry (that is, all entries are NAs) are also to be picked out. Different formats can be input into the function, see Details.

Usage

```
check_dup_wrong(x, xtype = 2, candidate = NULL, vv = NULL, isna = NULL,
  clean = FALSE)
```

Arguments

x	a data.frame, matrix or list of raw ballots. See Details.
xtype	should be 1, 2 (default) or 3, designating the type of x. See Details.
candidate	if xtype is 1, this argument is ignored. If xtype is 2 or3, candidate names must be given as a character or numeric vector. If a name is not given, but is still on a ballot, then the ballot is labelled as wrong.
vv	if xtype is 2 or 3, it is ignored. If xtype is 1, this gives the valid score values for x.
isna	entries which should be taken as NAs. NA in x be taken as missing value, however, you can add more (e.g., you may use 99, 999 as missing values). If x contains characters, this argument should also be provided with a character vector, and if numeric, then numeric vector. Do not add NA to isna, because the default (NULL) means NA is already included.
clean	the default is FALSE, that is, it does not return the cleaned data. If it is TRUE, a set of ballots without duplicated values, without mistakes and with at least one valid value, is returned.

Details

The function accepts the following input:

- (1) when xtype is 1, x must be a matrix. Column names are candidate names (if column names are NULL, they will be created: x1, x2, x3...). Candidate number is the number of columns of the matrix. Entry ij is the numeric score assigned by the ith voter to the jth candidate.
- (2) when xtype is 2, x can be a matrix or data.frame. Candidate number is the length of candidate. Entries are names (character or numeric) of candidates. The i1, i2, i3... entries are the 1st, 2nd, 3rd... preferences of voter i.
- (3) when xtype is 3, x should be a list. Each element of the list is a ballot, a vector contains the names (character or numeric) of candidates. The 1st preference is in the 1st position of the vector, the 2nd preference is in the 2nd position... The number of candidates is the length of candidate; as a result, a ballot with number of names larger than candidate number is labelled as wrong.

Value

a list with 3 or 4 elements: row_with_dup is the rows (not row names) of rows that have duplicated values; row_with_wrong is the rows with illegal names or the lengths of them are larger than candidate number (this could only happen when x is a list). row_all_na is the rows the entries of which are all NAs. For a list, elements with NULL are also taken as all-NA ballots.

Examples

```
raw=list(
    c('a', 'e', 'c', 'd', 'b'),
    c('b', 'a', 'e'),
    c('c', 'd', 'b'),
    c('d', 'a', 'b'),
    c('a', 'a', 'b', 'b', 'b'),
    c(NA, NA, NA, NA),
    v7=NULL,
    v8=c('a', NA, NA, NA, NA, NA, NA),
    v9=rep(" ", 3)
)
y=check_dup_wrong(raw, xtype=3, candidate=letters[1: 5])
y=check_dup_wrong(raw, xtype=3, candidate=letters[1: 4])
```

`create_vote` *Create a vote Object that can be used in counting methods*

Description

Some counting methods in this package only accept `vote` object created by this function. So the first step should always be using this function. The function will return the modified ballots and some other helpful information. See Details and Values.

Usage

```
create_vote(x, xtype = 2, candidate = NULL, isna = NULL)
```

Arguments

`x`	a data.frame, matrix or list of raw ballots. See Details.
`xtype`	should be 1, 2 (default) or 3, designating the type of `x`. See Details.
`candidate`	if `xtype` is 1, this argument is ignored. If `xtype` is 2 or 3, candidate names must be given as a character or numeric vector. If a name is not given, but is still on a ballot, then the name is ignored !
`isna`	entries which should be taken as NAs. `NA` in `x` will always be taken as missing value, however, you can add more (e.g., you may use 99, 999 as missing values). If x contains characters, this argument should also be provided with a character vector, and if numeric, then numeric vector. Do not add `NA` to `isna`, because the default (NULL) means `NA` is already included.

Details

The function accepts the following input:

- (1) when `xtype` is 1, x must be a matrix. Column names are candidate names (if column names are NULL, they will be created: x1, x2, x3...). Candidate number is the number of columns of the matrix. Entry ij is the numeric score assigned by the ith voter to the jth candidate.

- (2) when xtype is 2, x can be a matrix or data.frame. Candidate number is the length of candidate. Entries are names (character or numeric) of candidates. The i1, i2, i3... entries are the 1st, 2nd, 3rd... preferences of voter i.
- (3) when xtype is 3, x should be a list. Each element of the list is a ballot, a vector contains the names (character or numeric) of candidates. The 1st preference is in the 1st position of the vector, the 2nd preference is in the 2nd position... The number of candidates is the length of candidate; as a result, a ballot with number of names larger than candidate number is labelled as wrong.

The function also returns Condorcet matrix. Suppose candidates are i, j, k. The voter likes i best, so he assigns 1 to i. The 2nd choice is j, so he assigns 2 to j, leaving k as NA. Now computing the Condorcet matrix: since i's score is smaller than j' score, we add 1 to the ij cell of the matrix, and add 0 to the ji cell. Candidate k's NA is automatically set to the highest (that is, the worst) score: 3 (since there are 3 candidates); i < k, so we add 1 to the ik cell and add 0 to ki cell. Besides, there is also a score difference matrix: we add 2 - 1 = 1 to the ij cell of score difference matrix, and add 3 - 1 = 2 to the ik cell. If tie appears, both sides acquire 0.

Note the ways we calculate the Condorcet matrix. (1) It allow ties, that is, duplicated score values. (2) NA is deems as the worst, which means: if a voter does not mention a candidate, the candidate will be given the highest (worst) score. (3) Ballots mention only one name are assumed to express preference, since unmentioned candidates are assumed to be equally hated. (4) The Condorcet matrix returned by create_vote uses ballots that may have duplicated values and have only one valid entry. However, Condorcet family methods in this package provide possibility to recalculate the matrix. And, the simplest way to get rid of duplicated values and NAs is to delete some ballots.

Value

an object of class vote is returned, which is essentially a list. It has the following elements.

- (1) call the call.
- (2) ballot the returned ballot. It is always a score matrix. The column names are candidate names; entries are numeric scores assigned by voters. Missing values are all set to NA.
- (3) nas those which are taken as NA in data cleaning.
- (4) candidate candidate names.
- (5) candidate_num number of candidates.
- (6) ballot_num number of ballots.
- (7) ballot_at_least_one number of ballots that mention at least one candidate.
- (8) cdc the Condorcet matrix calculated with ballots that have no NA entries.
- (9) cdc_with_na the Condorcet matrix calculated with ballots that have at least one valid entry.
- (10) dif the score difference matrix calculated with ballots that have no NA entries.
- (11) dif_with_na the score difference matrix calculated with ballots that have at least one valid entry.
- (12) row_with_na rows of ballot with NAs.
- (13) row_non_na for rows with NAs, the number of non-NA entries of them.
- (14) row_with_dup rows of ballot with duplicated score values.
- (15) approval_able if length of row_non_dup is 0, then it is TRUE, else, FALSE. It indicates whether approval method can be used. When xtype is 2 or 3, it is always TRUE.

Examples

```
# xtype is 2
raw <- c(
    rep(c('m', 'n', 'c', 'k'), 42),
    rep(c('n', 'c', 'k', 'm'), 26),
    rep(c('c', 'k', 'n', 'm'), 15),
    rep(c('k', 'c', 'n', 'm'), 17)
)
raw <- matrix(raw, ncol = 4, byrow = TRUE)
vote <- create_vote(raw, xtype = 2, candidate = c('m', 'n', 'k', 'c'))

# xtype is 3
raw <- list(
    c('a', 'e', 'c', 'd', 'b'),
    c('b', 'a', 'e'),
    c('c', 'd', 'b'),
    c('d', 'a', 'b'),
    c('a', 'a', 'b', 'b', 'b'),
    c(NA, NA, NA, NA),
    v7 = NULL,
    v8 = c('a', NA, NA, NA, NA, NA, NA),
    v9 = rep(" ", 3)
)
y <- check_dup_wrong(raw, xtype = 3, candidate = letters[1: 4])
raw2 <- raw[-y$row_with_wrong]
vote <- create_vote(raw2, xtype = 3, candidate = letters[1: 4])

# xtype is 1
raw <- rbind(
    c(1, 2, 5, 3, 3),
    c(2, 1, 1, 3, 5),
    c(1, 2, 5, 3, 4),
    c(1, 2, 5, 3, 4),
    c(NA, NA, NA, NA, NA),
    c(NA, 3, 5, 1, 2),
    c(NA, 999, NA, 1, 5)
)
vote <- create_vote(raw, xtype = 1, isna = 999)
```

---

dowdall_method *Dowdall Method*

Description

This is an alternative Borda method. Voters are required to assign preference scores to every candidate and one score value cannot be shared by two or more candidates. For a voter, his 1st choice gets 1, his 2nd choice gets 1/2, his 3rd choice gets 1/3... The candidate who gets the most points wins. For the function `dowdall_method`, ranks, rather than true values, are used. So 1, 3, 5 are ranked as 1, 2, 3, and the scores are 1/1, 1/2, 1/3.

Usage

```
dowdall_method(x, stop = FALSE)
```

Arguments

x	an object of class vote. The ballots in the object should not have duplicated values and NAs.
stop	default is FALSE, when ballots do have duplicated values or NAs, error will not be raised, but the winner will be NULL. If TRUE, an error will be raised.

Value

a list object.

- (1) call the function call.
- (2) method the counting method.
- (3) candidate candidate names.
- (4) candidate_num number of candidate.
- (5) ballot_num number of ballots in x.
- (6) valid_ballot_num number of ballots that are used to compute the result.
- (7) winner the winners.
- (8) other_info total scores.

References

- https://en.wikipedia.org/wiki/Borda_count

Examples

```
raw <- list2ballot(string =
    c("51: a>c>b>d", "5: c>b>d>a", "23: b>c>d>a", "21: d>c>b>a")
)
vote <- create_vote(raw, xtype = 3, candidate = c("a", "b", "c", "d"))
y1 <- borda_method(vote) # winner is c
y2 <- dowdall_method(vote) # winner is a
```

---

irv_method *Instant-Runoff Voting Method*

Description

Instant-runoff voting (IRV) method is also called alternative voting, transferable voting, ranked-choice voting, single-seat ranked-choice voting, or preferential voting. In the 1st round, the candidate with absolute majority (that is, with more than 50 percent) wins. If no absolute winner exists, the one who gets the least votes is deleted, all other candidates enter into the 2nd round. Again, if no absolute winner exists, let the one with the least votes go and start the 3rd round... Finally, an absolute winner will appear. Ties are solved with different methods in reality; however, this function applies the following rules: (a) if more than one candidate gets the least votes, let all of them go; (b) if all the candidates get the same number of votes in a certain round, then all of them are winners. Note: the function accepts object of class `vote` and the method can only be used when x$approval_able is TRUE, that is, there is no duplicated values in the score matrix; otherwise, the winner will be NULL.

Usage

```
irv_method(x, min_valid = 1)
```

Arguments

`x`	an object of class `vote`.
`min_valid`	default is 1. If the number of valid entries of a ballot is less than this value, the ballot will not be used.

Value

a list object.

- (1) `call` the function call.
- (2) `method` the counting method.
- (3) `candidate` candidate names.
- (4) `candidate_num` number of candidate.
- (5) `ballot_num` number of ballots in x.
- (6) `valid_ballot_num` number of ballots that are used to compute the result.
- (7) `winner` the winners, may be NULL.
- (8) `absolute` whether the winner wins absolute majority in the 1st round.
- (9) `other_info` the IRV may run for 2 or more rounds. So here the summary information of each round is recorded. The length of the list is equal to the number of rounds.

References

- Reilly, B. 2004. The global spread of preferential voting: Australian institutional imperialism? Australian Journal of Political Science, 39(2), 253-266.

Examples

```
raw <- c(
   rep(c('m', 'n', 'c', 'k'), 42), rep(c('n', 'c', 'k', 'm'), 26),
   rep(c('c', 'k', 'n', 'm'), 15), rep(c('k', 'c', 'n', 'm'), 17)
)
raw <- matrix(raw, ncol = 4, byrow = TRUE)
vote <- create_vote(raw, xtype = 2, candidate = c('m', 'n', 'k', 'c'))
y <- irv_method(vote) # winner is k
```

---

list2ballot *Repeat ith element of list x or row of matrix/data.frames for j times*

Description

Suppose you have 3 different unique ballots and the amount of each ballot is 10, 20, 30. Now you want to create raw ballots as a list. Then you can use this function. See examples for usage.

Usage

```
list2ballot(x = NULL, n = rep(1, length(x)), m = NULL, string = NULL)
```

Arguments

x	a list, each element of which should be a vector. Note: only one of x, m and string can be a non-NULL object
n	how many times each element of x or each row of m should be replicated. It should be a numeric vector of non-negative integers and the length of it should be equal to that of x or the row number of m. The default is 1 for each element of x.
m	a matrix or dataframe, the number of rows should be equal to the length of n.
string	default is NULL. If it is not NULL, x, m and n are ignored. It should be a character vector. Each one contains two parts, the 1st is the amount of that ballot, and the 2nd part contains the names. The 1st and 2nd parts, as well as the names, should be split by spaces or punctuations. But no space and punctuation is allowed inside the names ("_" is not taken to be a punctuation). See examples.

Value

a list with replicated vectors, if x is not NULL, or a matrix/data.frame with duplicated rows, if m is not NULL.

Examples

```
# Use x and n
unique_ballot <- list(
    c("A", "B", "C"), c("F", "A", "B"),
    c("E", "D", "C", "B", "F", "A"), c("x","x", "A")
)
r <- c(1, 2, 3, 0)
y <- list2ballot(unique_ballot, r)

# Use string, x and n will be ignored.
# The characters can be written in a very loose way as follows,
# for the function will automatically delete unwanted parts.
# But do make sure there is no space or punctuation
# inside the names.
unique_ballot <- c(
"2, Bob, Mike Jane",  "3: barack_obama;;Bob>Jane",
"0 Smith   Jane", "  1 Mike???!!!"
)
y <- list2ballot(string = unique_ballot)
# Use a matrix.
m <- matrix(c(1, 2, 3, 3, 1, 2), nrow = 2, byrow = TRUE)
colnames(m) <- c("p1", "p2", "p3")
r <- c(3, 5)
y <- list2ballot(m = m, n = r)
```

`plurality_method` *Plurality Method to Find Absolute or Relative Majority*

Description

Although with plurality method each voter is required to mention only one candidate, a ballot with more than one candidate and different scores is also valid. For a score matrix, the function will check the position j which has the lowest score (in a `vote` object, the lower, the better) in the ith row. Duplicated values may or may not be a problem. For instance, `c(2, 3, 3)` is valid, for the lowest value is 2 and it is in the 1st position. However, `c(2, 2, 3)` is a problem, for the 1st and 2nd positions all have the lowest value 2. If this problem exists, the winner returned by this function will be NULL.

Usage

```
plurality_method(x, allow_dup = TRUE, min_valid = 1)
```

Arguments

`x`	an object of class `vote`.
`allow_dup`	whether ballots with duplicated score values are taken into account. Default is TRUE.
`min_valid`	default is 1. If the number of valid entries of a ballot is less than this value, the ballot will not be used.

Value

a list object.

- (1) `call` the function call.
- (2) `method` the counting method.
- (3) `candidate` candidate names.
- (4) `candidate_num` number of candidate.
- (5) `ballot_num` number of ballots in x.
- (6) `valid_ballot_num` number of ballots that are used to compute the result.
- (7) `winner` the winners, may be one, more than one or NULL.
- (8) `absolute` whether the winner is of absolute majority.
- (9) `other_info` a list with 2 elements, the 1st is the frequencies of candidates mentioned as 1st choice; the second element is the percentage. If winner is NULL, these two are NULL.

Examples

```
raw <- rbind(
    c(1, 2, 5, 3, 3), c(1, 2, 5, 3, 4), c(1, 2, 5, 3, 4),
    c(NA, NA, NA, NA, NA), c(NA, 3, 5, 1, 2),
    c(NA, NA, NA, 2, 3), c(NA, NA, 1, 2, 3),
    c(NA, NA, NA, NA, 2), c(NA, NA, NA, 2, 2),
    c(NA, NA, 1, 1, 2), c(1, 1, 5, 5, NA)
)
vote <- create_vote(raw, xtype = 1)
y <- plurality_method(vote, allow_dup = FALSE)
y <- plurality_method(vote, allow_dup=FALSE, min_valid = 3)
```

---

star_rating *User Preference Aggregation*

Description

The function uses a simple method to calculate the aggregation scores of user ratings, which is described in Langville, A. and Meyer, C. (2012: 128). Input data can be stored in a sparse matrix. Suppose there are 100 films and users are required to assign scores. However, each user only watched several of them. Thus, when comparing two films A and B, the method only takes account ratings from those who watched both A and B.

Usage

```
star_rating(x, show_name = FALSE, check_na = TRUE)
```

Arguments

x	a numeric matrix, or, dgCMatrix and dgeMatrix matrix created with the Matrix package. 0 in the data means no score is given; and valid score values should all be larger than 0. The function will do NOTHING to check the validity. Besides, NA also means no score is given. If your data has NA, set check_na to TRUE so as to convert NA to 0.
show_name	the default is FALSE, that is to say, the function does not store and show candidate names in the result and you cannot see them. However, you can set it to TRUE.
check_na	if it is TRUE, the function will check NAs and convert them to 0s. If NAs do exist and check_na is FALSE, error will be raised.

Value

a list object.

- (1) call the function call.
- (2) method the counting method.
- (3) candidate candidate names. If show_name is FALSE, this will be NULL.
- (4) candidate_num number of candidate.
- (5) ballot_num number of ballots in x.
- (6) valid_ballot_num number of ballots that are used to compute the result.
- (7) winner the winner. If show_name is FALSE, this only shows the number in 1: ncol(x).
- (8) winner_score the winner's score, which is the highest score.
- (9) other_info scores of all the candidates.

References

- Langville, A. and Meyer, C. 2012. Who's #1? The Science of Rating and Ranking. Princeton University Press, p. 128.

Examples

```
# Example from Langville and Meyer, 2012: 128.
# 4 films are rated by 10 users; 0 means no score.
raw <- c(4, 3, 1, 2, 0, 2, 0, 3, 0, 2, 2, 1, 0, 4, 3, 3, 4,
    1, 3, 0, 2, 0, 2, 2, 2, 0, 1, 1, 2, 2, 0, 2, 0, 0, 5, 0, 3,
    0, 5, 4
)
m <- matrix(raw, ncol = 4)
colnames(m) <- paste("film", 1: 4, sep = "")
y <- star_rating(m, show_name = TRUE) # winner is film4
```

Index