Package 'fastVoteR'

November 27, 2024

Title Efficient Voting Methods for Committee Selection

Version 0.0.1

Description A fast 'Rcpp'-based implementation of polynomially-computable voting theory methods for committee ranking and scoring. The package includes methods such as Approval Voting (AV), Satisfaction Approval Voting (SAV), sequential Proportional Approval Voting (PAV), and sequential Phragmen's Rule. Weighted variants of these methods are also provided, allowing for differential voter influence.

License LGPL (>= 3)

URL https://bblodfon.github.io/fastVoteR/

Imports checkmate, data.table, Rcpp

Suggests mlr3misc (>= 0.15.1), testthat (>= 3.0.0)

LinkingTo Rcpp

Config/testthat/edition 3

Encoding UTF-8

RoxygenNote 7.3.2

NeedsCompilation yes

Author John Zobolas [cre, aut] (<https://orcid.org/0000-0002-3609-8674>), Anne-Marie George [ctb] (<https://orcid.org/0000-0001-9232-8211>)

Maintainer John Zobolas <bblodfon@gmail.com>

Repository CRAN

Date/Publication 2024-11-27 12:40:02 UTC

Contents

rank_candidates *Rank candidates based on voter preferences*

Description

Calculates a ranking of candidates based on voters' preferences. Approval-Based Committe (ABC) rules are based on Lackner et al. (2023).

Usage

```
rank_candidates(
  voters,
  candidates,
  weights = NULL,
  committee_size = NULL,
  method = "av",
  borda_score = TRUE,
  shuffle_candidates = TRUE
)
```

Arguments

voters	(list) A list of subsets, where each subset contains the candidates approved or selected by a voter.
candidates	(character) A vector of all candidates to be ranked.
weights	(numeric) A numeric vector of weights representing each voter's influence. Larger weight, higher influence. Must have the same length as voters. If NULL (default), all voters are assigned equal weights of 1, representing equal influence.
committee_size	(integer(1)) Number of top candidates to include in the ranking. Default (NULL) includes all candidates. For sequential methods such as "seq_pav" and "seq_phragmen", this parameter can speed up computation by limiting the selection process to only the top N candidates, instead of generating a complete ranking. In other methods (e.g., "sav" or "av"), this parameter simply filters the final output to include only the top N candidates from the complete ranking.
method	(character(1)) The ranking voting method to use. Must be one of: "av", "sav", "seq_pav", "seq_phragmen". See Details.
borda_score	(logical(1)) Whether to calculate and include Borda scores in the output. See Details. Default is TRUE.

`shuffle_candidates`
: (`logical(1)`)
Whether to shuffle the candidates randomly before computing the ranking. Shuffling ensures consistent random tie-breaking across methods and prevents deterministic biases when candidates with equal scores are encountered. Default is `TRUE`. Set to `FALSE` if deterministic ordering of candidates is preferred.

Details

This method implements several consensus-based ranking methods, where voters express preferences for candidates. The input framework considers:

- **Voters**: A list where each element represents the preferences (subsets of candidates) of a single voter.
- **Candidates**: A vector of all possible candidates. This vector is shuffled before processing to enforce random tie-breaking across methods.
- **Weights**: A numeric vector specifying the *influence* of each voter. Equal weights indicate all voters contribute equally; different weights can reflect varying voter importance.

The following methods are supported for ranking candidates:

- `"av"`: **Approval Voting (AV)** ranks candidates based on the number of voters approving them.
- `"sav"`: **Satisfaction Approval Voting (SAV)** ranks candidates by normalizing approval scores based on the size of each voter's approval set. Voters who approve more candidates contribute a lesser score to the individual approved candidates.
- `"seq_pav"`: **Sequential Proportional Approval Voting (PAV)** builds a committee by iteratively maximizing a proportionality-based satisfaction score. The **PAV score** is a metric that calculates the weighted sum of harmonic numbers corresponding to the number of elected candidates supported by each voter, reflecting the overall satisfaction of voters in a committee selection process.
- `"seq_phragmen"`: **Sequential Phragmen's Rule** selects candidates to balance voter representation by distributing "loads" evenly. The rule iteratively selects the candidate that results in the smallest increase in voter load. This approach is suitable for scenarios where a balanced representation is desired, as it seeks to evenly distribute the "burden" of representation among all voters.

All methods have weighted versions which consider voter weights.

To allow for method-agnostic comparisons of rankings, we calculate the **borda scores** for each method as:

$$s_{borda} = (p - i)/(p - 1)$$

where p is the total number of candidates, and i is the candidate's rank.

Value

A data.table::data.table with columns:

- `"candidate"`: Candidate names.
- `"score"`: Scores assigned to each candidate based on the selected method (if applicable).

- `"norm_score"`: Normalized scores (if applicable), scaled to the range $[0, 1]$, which can be loosely interpreted as **selection probabilities** (see Meinshausen et al. (2010) for an example in Machine Learning research where the goal is to perform stable feature selection).
- `"borda_score"`: Borda scores for method-agnostic comparison, ranging in $[0, 1]$, where the top candidate receives a score of 1 and the lowest-ranked candidate receives a score of 0.

Candidates are ordered by decreasing `"score"`, or by `"borda_score"` if the method returns only rankings.

References

Meinshausen, Nicolai, Buhlmann, Peter (2010). "Stability Selection." *Journal of the Royal Statistical Society Series B: Statistical Methodology*, **72**(4), 417–473. ISSN 1369-7412, doi:10.1111/J.14679868.2010.00740.X, 0809.2932.

Lackner, Martin, Skowron, Piotr (2023). *Multi-Winner Voting with Approval Preferences*. Springer Nature. ISBN 9783031090165, doi:10.1007/9783031090165, 2007.01795. "

Examples

```
# 5 candidates
candidates = paste0("V", seq_len(5))

# 4 voters
voters = list(
  c("V3", "V1", "V4"),
  c("V3", "V1"),
  c("V3", "V2"),
  c("V2", "V4")
)

# voter weights
weights = c(1.1, 2.5, 0.8, 0.9)

# Approval voting (all voters equal)
rank_candidates(voters, candidates)

# Approval voting (voters unequal)
rank_candidates(voters, candidates, weights)

# Satisfaction Approval voting (voters unequal, no borda score)
rank_candidates(voters, candidates, weights, method = "sav", borda_score = FALSE)

# Sequential Proportional Approval voting (voters equal, no borda score)
rank_candidates(voters, candidates, method = "seq_pav", borda_score = FALSE)

# Sequential Phragmen's Rule (voters equal)
rank_candidates(voters, candidates, method = "seq_phragmen", borda_score = FALSE)
```

Index