Package 'fastVoteR'

May 15, 2026

Title Efficient Voting Methods for Committee Selection

Version 0.0.3

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 MIT + file LICENSE

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

Imports checkmate, Rcpp

Suggests testthat (>= 3.0.0)

LinkingTo Rcpp

Config/testthat/edition 3

Encoding UTF-8

RoxygenNote 7.3.3

NeedsCompilation yes

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

Maintainer John Zobolas <bblodfon@gmail.com>

Repository CRAN

Date/Publication 2026-05-15 20:20:13 UTC

Contents

`fastVoteR-package` *fastVoteR: Efficient Voting Methods for Committee Selection*

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.

Author(s)

Maintainer: John Zobolas <bblodfon@gmail.com> (ORCID)

Other contributors:

- Anne-Marie George (ORCID) [contributor]

See Also

Useful links:

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

`av` *Approval Voting*

Description

Approval Voting (AV) ranks candidates based on the number of voters approving them.

This function uses an internal C++ implementation for efficient computation. For equal weights, a faster R implementation is used.

Usage

```
av(
  voters,
  candidates,
  weights = NULL,
  committee_size = NULL,
  borda_score = TRUE,
  check = FALSE
)
```

Arguments

`voters`	(`list()`) A list of subsets (`character` vectors), where each subset contains the candidates approved or selected by a voter.
`candidates`	(`character()`) A vector of all candidates to be ranked.
`weights`	(`numeric()|NULL`) A numeric vector of non-negative 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)|NULL`) Number of top-ranked candidates to return. Default (`NULL`) returns all candidates.
`borda_score`	(`logical(1)`) Whether to include a `borda_score` column in the output, which provides a normalized score based on the candidate's rank. If `TRUE` (default), the `borda_score` is calculated as $(p - i)/(p - 1)$, where p is the total number of candidates and i is the candidate's rank.
`check`	(`logical(1)`) Whether to run additional voter-integrity checks. When `TRUE`, each voter must approve at least one candidate, approvals must be unique per voter, and all approved candidates must appear in `candidates`. Use `FALSE` to skip these checks when inputs are known to be valid.

Value

A `data.frame` with columns:

- `"candidate"`: Candidate names.
- `"score"`: Approval scores.
- `"norm_score"`: Normalized scores, scaled to the range $[0, 1]$.
- `"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, based on the total number of candidates.

Candidates are ordered by decreasing `"score"`.

See Also

Other voting methods: `sav()`, `seq_pav()`, `seq_phragmen()`

`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). For an example use in Machine Learning for ensemble feature selection, see Zobolas et al. (2026).

Usage

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

Arguments

`voters`	(`list()`) A list of subsets (`character` vectors), where each subset contains the candidates approved or selected by a voter.
`candidates`	(`character()`) A vector of all candidates to be ranked.
`weights`	(`numeric()|NULL`) A numeric vector of non-negative 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)|NULL`) Number of top-ranked candidates to return. Default (`NULL`) returns all candidates.
`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 include a `borda_score` column in the output, which provides a normalized score based on the candidate's rank. If `TRUE` (default), the `borda_score` is calculated as $(p - i)/(p - 1)$, where p is the total number of candidates and i is the candidate's rank.

`shuffle_candidates`
: (`logical(1)`)
Whether to randomly shuffle candidates before ranking. This provides random tie-breaking and avoids deterministic bias when scores are equal. Default is `TRUE`.

`check`
: (`logical(1)`)
Whether to run additional voter-integrity checks. When `TRUE`, each voter must approve at least one candidate, approvals must be unique per voter, and all approved candidates must appear in `candidates`. Use `FALSE` to skip these checks when inputs are known to be valid.

Details

We implement several consensus-based ranking methods, where voters express preferences for candidates. The framework has three components:

- **Voters**: A list where each element is the set of approved candidates for a single voter.
- **Candidates**: A character vector of all possible candidates. This vector can be shuffled to randomize tie-breaking across methods.
- **Weights**: A numeric vector giving each voter's influence. Equal weights mean equal influence; differing weights reflect varying importance.

This function is a thin wrapper that dispatches to method-specific implementations. Supported methods include:

1. Approval Voting (`method = "av"`), calls `av()`
2. Satisfaction Approval Voting (`method = "sav"`), calls `sav()`
3. Sequential Proportional Approval Voting (`method = "seq_pav"`), calls `seq_pav()`
4. Sequential Phragmen's Rule (`method = "seq_phragmen"`), calls `seq_phragmen()`

All methods support voter weights.

For method-agnostic comparisons, we compute **borda scores**, which map each candidate's rank to a normalized scale across methods that may otherwise return different score scales or only ordinal rankings.

For sequential methods such as `"seq_pav"` and `"seq_phragmen"`, the `committee_size` parameter can speed up computation by selecting only the top N candidates instead of producing a full ranking. For non-sequential methods (e.g., `"sav"` or `"av"`), it simply truncates the final ranking to the top N candidates.

Value

A `data.frame` 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

Lackner M, Skowron P (2023). *Multi-Winner Voting with Approval Preferences*. Springer Nature, 121 p. doi:10.1007/9783031090165.

Zobolas J, George A, Lopez A, Fischer S, Becker M, Aittokallio T (2026). "Prognostic biomarker discovery in pancreatic cancer through hybrid ensemble feature selection and multi-omics data." *BioData Mining*. doi:10.1186/s13040026005460.

Meinshausen N, Buhlmann P (2010). "Stability Selection." *Journal of the Royal Statistical Society Series B: Statistical Methodology*, 72(4), 417-473. doi:10.1111/J.14679868.2010.00740.X.

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)
```

sav *Satisfaction Approval Voting*

Description

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.

This function uses an internal C++ implementation for efficient computation.

Usage

```
sav(
  voters,
  candidates,
  weights = NULL,
  committee_size = NULL,
  borda_score = TRUE,
  check = FALSE
)
```

Arguments

`voters`	(`list()`) A list of subsets (`character` vectors), where each subset contains the candidates approved or selected by a voter.
`candidates`	(`character()`) A vector of all candidates to be ranked.
`weights`	(`numeric()|NULL`) A numeric vector of non-negative 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)|NULL`) Number of top-ranked candidates to return. Default (`NULL`) returns all candidates.
`borda_score`	(`logical(1)`) Whether to include a `borda_score` column in the output, which provides a normalized score based on the candidate's rank. If `TRUE` (default), the `borda_score` is calculated as $(p - i)/(p - 1)$, where p is the total number of candidates and i is the candidate's rank.
`check`	(`logical(1)`) Whether to run additional voter-integrity checks. When `TRUE`, each voter must approve at least one candidate, approvals must be unique per voter, and all approved candidates must appear in `candidates`. Use `FALSE` to skip these checks when inputs are known to be valid.

Value

A `data.frame` with columns:

- `"candidate"`: Candidate names.
- `"score"`: Satisfaction scores.
- `"norm_score"`: Normalized scores, scaled to the range $[0, 1]$.
- `"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, based on the total number of candidates.

Candidates are ordered by decreasing `"score"`.

See Also

Other voting methods: `av()`, `seq_pav()`, `seq_phragmen()`

`seq_pav` *Sequential Proportional Approval Voting*

Description

Sequential Proportional Approval Voting (SeqPAV) is a multi-winner method that builds a committee by iteratively maximizing a proportionality-based satisfaction score. After each selection, the weights of voters who approved the chosen candidate are reduced, which promotes proportional representation. The **PAV score** is computed as the weighted sum of harmonic numbers based on how many elected candidates each voter supports. The process continues until the specified committee size is reached or all candidates are ranked.

This function uses an internal C++ implementation for efficient computation.

Usage

```
seq_pav(
  voters,
  candidates,
  weights = NULL,
  committee_size = NULL,
  borda_score = TRUE,
  check = FALSE
)
```

Arguments

`voters` (`list()`)
A list of subsets (`character` vectors), where each subset contains the candidates approved or selected by a voter.

`candidates` (`character()`)
A vector of all candidates to be ranked.

`weights` (`numeric()|NULL`)
A numeric vector of non-negative 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)|NULL`)
Number of top-ranked candidates to return. Default (`NULL`) returns all candidates.

`borda_score` (`logical(1)`)
Whether to include a `borda_score` column in the output, which provides a normalized score based on the candidate's rank. If `TRUE` (default), the `borda_score` is calculated as $(p - i)/(p - 1)$, where p is the total number of candidates and i is the candidate's rank.

`check` (`logical(1)`)
Whether to run additional voter-integrity checks. When `TRUE`, each voter must approve at least one candidate, approvals must be unique per voter, and all approved candidates must appear in `candidates`. Use `FALSE` to skip these checks when inputs are known to be valid.

Value

A `data.frame` with columns:

- `"candidate"`: Candidate names.
- `"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, based on the total number of candidates.

Candidates are ordered by the sequence in which they were selected.

See Also

Other voting methods: `av()`, `sav()`, `seq_phragmen()`

`seq_phragmen` *Sequential Phragmen's Rule*

Description

Sequential Phragmen's Rule is a multi-winner method that builds a committee by distributing representation *loads* across voters as evenly as possible. At each step, it selects the candidate that yields the smallest increase in voter load, then updates loads for voters who approved the chosen candidate. The process continues until the committee is filled or all candidates are ranked.

This function uses an internal C++ implementation for efficient computation.

Usage

```
seq_phragmen(
  voters,
  candidates,
  weights = NULL,
  committee_size = NULL,
  borda_score = TRUE,
  check = FALSE
)
```

Arguments

voters	(list()) A list of subsets (character vectors), where each subset contains the candidates approved or selected by a voter.
candidates	(character()) A vector of all candidates to be ranked.
weights	(numeric()|NULL) A numeric vector of non-negative 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)|NULL) Number of top-ranked candidates to return. Default (NULL) returns all candidates.
borda_score	(logical(1)) Whether to include a borda_score column in the output, which provides a normalized score based on the candidate's rank. If TRUE (default), the borda_score is calculated as $(p - i)/(p - 1)$, where p is the total number of candidates and i is the candidate's rank.
check	(logical(1)) Whether to run additional voter-integrity checks. When TRUE, each voter must approve at least one candidate, approvals must be unique per voter, and all approved candidates must appear in candidates. Use FALSE to skip these checks when inputs are known to be valid.

Value

A data.frame with columns:

- "candidate": Candidate names.
- "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, based on the total number of candidates.

Candidates are ordered by the sequence in which they were selected.

See Also

Other voting methods: `av()`, `sav()`, `seq_pav()`

Index