pref R package manual

Denis Mollison (Heriot-Watt University) - 25 November 2023

Summary

Preferential voting can be used to elect a set of representatives, or just to make a single choice. At present, this package only covers the first of these, using the *Single Transferable Vote* (STV). It provides explanations and graphics designed to show as clearly as possible how the count works - with both text and plots as it progresses, and with web pages that make the results permanently fully accessible. The principles of STV and the outline of the steps required to implement it are described in $7 below.

The core function *stv* uses Meek's algorithm, the conceptually purest and simplest form of STV, and allows voters to express equal preferences (see $7). There is also a function *stv.wig* for the WIG method used for Scottish Council elections since 2007 - and a batch mode (*stv.batch*, not currently documented) to run counts for multiple elections, as used to process the complete 2012, 2017 and 2022 Scottish Council elections.

The required vote data format is as an R list (see $ 4 below). A function *pref.data* is provided to transform some commonly used data formats into this format.

1 Package description

The goal of the function *stv* is to count votes from an STV election, and to provide a clear and full description of the count and the result, using both numbers and graphics. Its core option is Meek STV, which requires electronic counting, but, among other advantages, can handle votes expressing equal preferences for subsets of the candidates.

The function *stv.wig* implements the Weighted Inclusive Gregory method, as used in Scottish council elections; it has the same options as *stv*.

Both these functions require input in the form of an R list *elecdata*, whose only essential element is a matrix of vote data. A function *pref.data* is provided to translate various common vote file formats into the required format. A range of examples from actual elections is included - see $6 below.

The most recent stable version of pref can be installed from CRAN using `install.packages(pref)`, and the development version from Github:

```
library(devtools)
install_github("denismollison/pref")
```

Either way, the package can then be loaded into an R session using `library(pref)`.

2 Usage

First, if vote data are not in the recommended format they should be converted using the function *pref.data* (see $4 below). Then the election count is run using the function *stv*:

```
results=stv(elecdata)
```

(or similarly for *stv.wig*). This will use the default options for *stv*: a brief summary of the result will be printed, while full details and plots with webpages to display them will be stored in an output directory *out_stv* (resp. *out_wig*) within the working directory. Other options (see $3 below) include pausing the election at each stage with fuller details and a plot of the current state of the count displayed, or omitting graphical output altogether; while the output directory can be specified as desired, or set to tempdir() so that it is ephemeral.

3 Functions associated with the count

stv`(elecdata,outdirec="out_stv",verbose=F,plot=T,webdisplay=F)`

The value returned is a list containing five items:

elec - the names of those elected, in order of election
itt - candidates in order of election/exclusion, reported at each stage
votes - a matrix of the votes at each stage and the final keep values

va - a 3D array showing how votes have transferred (from first to current preference) for each stage
keep - the keep values (as %s) at each stage

With the default options, as shown above, the function will make webpages including plots showing each stage of the count and store them in directory "out_stv", but will not display them.

A full expository option is stv(elecdata,verbose=T, webdisplay=T); this displays progress in numbers and a plot at each stage of the count, requiring the user to ask for the next stage when ready, and concludes by displaying the full results as web pages. Note that these webpages include a numerical display of full details of the count and plots of transfers, each as an option that requires clicking a button to display.

Alternatively, combining the options verbose=F and plot=F calculates the detailed result with minimal printout and no graphical output.

Subsidiary functions used in the vote count calculations include *transfer*, which revises the keep values *k* at each stage. Following transfers, the stage is completed by the function *decision*: if any additional candidates have sufficient votes they are deemed elected, otherwise the candidate with fewest votes is excluded. A function *decision_text* expresses the decision in words (a function *plural* helps with grammar).

stv.wig(elecdata,outdirec="out",verbose=F,plot=T,webdisplay=F)

This function implements the 'Weighted Inclusive Gregory' algorithm, which has been used for Scottish Council elections since 2007. This differs from Meek in not allowing transfers to already elected candidates. While that may seem harmless - or even desirable - it introduces discontinuities, and the quota cannot be adjusted: a significant factor in political elections and those with large numbers of candidates. The sole justification for preferring it to Meek is that it can if necessary be counted by hand (which is probably not relevant if you're reading this). The differences in practice between the WIG and Meek algorithms can be explored at www.macs.hw.ac.uk /~denis/stv_elections, which cross-references each Scottish council count (for 2012 and 2017) with its Meek version.

The value returned by the *stv.wig* function is a list containing four items: *elec, itt, votes* and *va*, as for the *stv* function. [*keep* is not relevant for WIG.]

4 Functions associated with data input

The input format required by *stv* is a list *ed* which can have the following elements:

ed$e - election title
ed$s - number of representatives to be elected
ed$c - number of candidates
ed$nv - number of votes
ed$v - matrix of votes (*ed$nv* × *ed$c*)
ed$m - multiplicity for each vote (=1 if just one vote per row)
edn, edf, ed$n2 - name, first name, and abbreviated name for each candidate
edp, edcol - party acronyms and party colours of candidates (if appropriate - otherwise = "")

The first thing to say is that the only essential element here is the vote matrix ed$v. If the candidates' names are not given, they will be assumed to be *dimnames(ed$v)[[2]]*, and if this is empty capital letters A,B,... will be used. The number of representatives to be elected, *ed$s*, is also essential, but if it is not given the user will be asked for it.

The function *pref.data* converts some common preference data formats into the required format. Its defaults -

pref.data(datafile,mult=F,details=T,parties=F,ballot=F, friendly=F)

- correspond to the common .blt format, which is for example used for Scottish council election data; though for those data one needs mult=T, and should also set parties=T.

In more detail, the options are:

mult - if T, the multiplicity of each row of the vote matrix is given as its first element
details - if F, data are simply a vote matrix, with a header of candidate names or identifiers, with the first column containing multiplicities if mult=T
parties - either F, or the name of a file with party acronyms and colours

ballot - if T, *i*th entry in a row is the preference number for candidate *i*
- if F, rows are candidate numbers in order of preference, with bracketing indicating equal preference

friendly - if T, file starts with title, then *ed$s* and *ed$c*, then candidate names
- if F, starts with *ed$c* and *ed$s*, with candidate names and election title at end of file (the common `.blt' format)

Other input functions are *abbrev*, to calculate a suitable abbreviated name, *name2*; *party_colour*, to associate party colours with candidates if appropriate; and *cap_words*, for consistency in name style.

Other data formats

Hopefully, users with data in other formats will not find it too hard to convert their data into an R list with components as for *ed* above. Some points to note are:

ed$nv is the number of lines of vote data: this will be the actual vote total if these are all individual votes; otherwise, the vote total is *sum(ed$m)*.
The variable *ed$n2* is used to avoid names of excessive length, and to distinguish two candidates with the same surname; if these are not a problem for your data, just set *ed$n2 = ed$n*.
If party acronyms and colours are not relevant, just set *ed$p* and *ed$col* to the empty character (*i.e. = rep("",ed$c)*). The program will then use its `rainbow' default to choose colours for the graphics.

STV allowing equal preferences

It should be noted that *elecdata* does not explicitly include information on whether equal preferences are allowed. The numbers in each row of the vote matrix establish that voter's order of preference: if some of the numbers are equal, *stv* treats them as equal preferences. On the other hand, *stv.wig* expects orders of preference to be strict, and will crash if given data including equal preferences.

5 Functions associated with output

The functions *voteplot* and *webpages*, particularly the latter, are unusual uses of R, [I would be interested to hear of others who have used R to write web pages.] Suggestions as to how these functions might be made more elegant will be gratefully received. Note that the plots are saved as jpegs and then displayed using a function *plot_jpeg* that relies on the package *jpeg*. This is to avoid dependence on local R plotting parameters.

In case of problems with live plots and browser opening, ensure that the (default) options plot=T and webdisplay=F are set, allowing you to edit the webpages before trying to display them.

6 Examples

Input function *pref.data*

Yale

An example with input data simply a vote matrix, with first row containing candidate acronyms. [For more details see last of Count function examples.]

```
datafile=system.file("extdata","yale12.dat",package="pref")
y12=pref.data(datafile,details=FALSE)
```

Jedburgh and District 2012

An example converting Scottish Council election data from their original ".blt" format into an R list with full details, including party names and colours.

```
datafile=system.file("extdata","Jedburgh2012.blt",package="pref")
parties12=system.file("extdata","parties_SC2012.txt",package="pref")
jed12=pref.data(datafile,mult=TRUE,parties=parties12)
```

John Muir Trust 2002

An example in ".blt" format with equal preferences indicated using brackets. Note that the file has been reordered in "friendly" format, *i.e.* with the election title and candidate names at the beginning of the file.

```
datafile=system.file("extdata","jmt2002.blt",package="pref")
j02=pref.data(datafile,friendly=TRUE)
```

Count functions *stv, stv.wig*

The first four examples below come from Scottish Council elections; using *stv.wig* reproduces the official result exactly. The full results for 2012-22 can be found at www.macs.hw.ac.uk/~denis /stv_elections. For the elections of 2012 and 2017, both the official count and one using Meek STV are given, with links to switch between the two for ease of comparison.

Helensburgh Central 2012 hc12result=stv.wig(hc12)

A very simple example of an election to choose 4 representatives, where 4 candidates had first preference totals of over 20%, and so were elected at the first stage. Note that in this simple case there is no difference between different STV methods.

Partick East - Kelvindale 2017 p17result=stv.wig(p17)

A more typical STV election.

North West and Central Sutherland 2017
nws17wig=stv.wig(nws17)

An example exposing one of the flaws of WIG STV: none of the elected candidates achieved the quota.

Cumnock and New Cumnock 2017 cnc17wig=stv.wig(cnc17)

An example where WIG and Meek give different results: here one of the two leading parties has fewer first preferences but they are more equally divided between their two candidates; under Meek this matters less because reductions in the quota allow more transfers between the other party's candidates.

John Muir Trust 2002 j02result=stv(j02)

An example from an election allowing equal preferences.

Yale yaleresult=stv(yale) (or =stv.wig(yale))

An example (from a faculty election at Yale university) with a large number of candidates, 44 for 4 places. While the graphics cope, they are difficult to read. Note that this election was held under `Cambridge rules', so neither *stv* nor *stv.wig* reproduces the official outcome. Indeed they both agree in filling the 4th place differently from the official result.

`Yale - last 12' y12result=stv(y12) (or =stv.wig(y12))

A version of the previous example, starting after 32 candidates have been excluded (but noone yet elected), to give clearer plots of the later (and decisive) stages of the election.

7 Background: STV

The Single Transferable Vote is a system designed to elect representatives in such a way that each represents the same number of voters. For its relation to core democratic principles, and comparison with alternative forms of proportional representation, see Mollison (2023).

In STV, each voter provides their order of preference; these preferences are used to transfer unused votes or parts of votes from earlier to later preferences, according to the following algorithm:

(a) Votes are initially assigned to the voter's first choice

(b) The number of votes required to ensure election is calculated; this is called the *quota*. When the total number of votes is v and there are s seats to be filled, the quota is v/(s+1)

(c) Any candidate whose total reaches the quota is elected; if they have more than the quota, the surplus is transferred to their voters' next preferences

(d) If not all seats are filled, the candidate with fewest votes is excluded and all their votes are transferred to their voters' next preferences

Steps (b-d) are repeated, as necessary, until all seats are filled.

In counting the votes the only significant difficulty is in distributing the surpluses of elected candidates, where fairness suggests that equal fractions (typically s_i/v_i) of each vote should be transferred, and that, as a slightly less obvious part of that fairness, transfers should go exactly where the voter has requested even if that candidate has sufficient votes already. The latter feature has the knock-on effect of requiring further transfers, which is why (a) the count requires a computer, and (b) why this feature was not

implemented until computers were available (Meek published his algorithm in 1969/70). Meek's method has a number of other advantages, conceptual and practical, and there is really no good reason why any other method should be used if the vote data can be gathered securely as a computer file.

History of use

The original idea of STV goes back to Thomas Wright Hill (1819), with various improvements (use of preferential voting (Andrae, Hare), quota $\sim v/(s+1)$ (Droop) rather than $\sim v/s$ (Hare), and fractional transfers (Gregory)) introduced between 1855 and 1881, after which there was little change until Meek's reassessment for the computer age nearly 100 years later. STV has been used for political elections in various countries and regions, including Tasmania (since 1909) and Ireland (since its independence in 1921). In the UK it is currently used for the Northern Ireland Assembly and for Scottish Council elections - the latter provide the best source of STV data currently available.

STV is also increasingly used for electing the governing bodies of non-political organisations. STV allowing voters to express equality of preference has been used by the John Muir Trust and the London Mathematical Society (since 1998 and 1999 respectively). The program used for those elections, the first publicly available for STV allowing equal preferences, was written in Pascal by the late David Hill; it was originally available through ERS Services, currently through Civica.

8 Development and extensions

Much of the motivation for developing this software lay in the study of the large sets of data available from Scotland's council elections since 2007, comprising over 1000 individual elections in all. The development therefore included programs to tidy up such data for processing, and a program *stv.batch* to run counts for such large data sets. This software is not currently at a stage suitable for the public domain, but is available from the author on request.

Future planned developments include adapting the web display functions so that they can be used on their own to display already calculated vote results.

The more radical future ambition is to add software for Condorcet's method (`majority rule'), for calculating and analysing results of elections to make a single choice from preference data.

References

Meek, B. L. (1969) `Une nouvelle approche du scrutin transférable', *Mathématiques et sciences humaines* **25**, 13-23.

Meek, B. L. (1970) `Une nouvelle approche du scrutin transférable (fin)'. *Mathématiques et sciences humaines* **29**, 33-39.

Mollison, Denis (2023) `Fair votes in practice', https://arxiv.org/abs/2303.15310.

Feedback

Comments, and expressions of interest in collaboration, will be very welcome.

denis.mollison (at) gmail.com