stv R package manual

Denis Mollison (Heriot-Watt University) - March 2022

Summary
Usage
Background: STV
Functions associated with the count
Functions associated with data input
Functions associated with output
Examples
References

Summary

This package counts votes for elections using the *Single Transferable Vote* (STV), and provides explanations and graphics designed to show as clearly as possible how the count works - both as it progresses, and as web pages that make the results permanently fully accessible. There are just two main functions, *stv* and *stv.data*.

The core function *stv* uses Meek's algorithm, the conceptually purest and simplest form of STV, and allows voters to express equal preference. Options are included for other commonly used STV variants, as is a batch mode to run counts for multiple elections. *[These options are under development. For examples of what can be done see www.macs.hw.ac.uk/∼denis/stv_elections.]*

The preferred data format is as a list, but a function *stv.data* is provided to transform some commonly used data formats into this preferred format.

Usage

The package can be installed using

ilibrary(devtools)   (if not installed, first do install.packages("devtools"))
install_github("denismollison/stv")

An election is then run using the function stv: if the data are in the recommended format as a list *elecdata*, the results are found using

library(stv)
results=stv(elecdata)

This will use the defaults 'verbose" and 'plot'; in verbose mode, the program pauses at each stage wit a prompt of 'next?"; to continue the user should press the 'return' key. On completion, a webpage with a slideshow of the election will appear. These outputs can be suppressed by adding 'verbose=F' and/or 'plot=F' to the function call. For more details, see *stv* below.

Background: STV

The Single Transferable Vote algorithm can be summarised as:

(a) votes are initially assigned to the voter's first choice;
(b) calculate the number of votes a candidate needs to be elected (the *quota*);
(c) if a candidate has more votes than needed, pass on the excess by transferring the same proportion of each vote to that voter's next choice;
(d) if not all seats are filled, exclude the candidate with fewest votes, transferring the whole of each vote to the voter's next choice.

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 (including the use of preferential voting, and of fractional transfers, and establishing the quota necessary for election $(v_i > sum(v)/(s+1)))$ 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 is used by the John Muir Trust and the London Mathematical Society (since 1998 and 1999 respectively.

Functions associated with the count

stv(elecdata,outdirec=tempdir(),verbose=T,plot=T)

The value returned is a list containing six items:

elec - the (abbreviated) names of those elected, in order of election
votes - a matrix of the votes at each stage and the final keep values
keep - the keep values (as %s) at each stage
va - a 3D array showing how votes have transferred (from 1st pref to current) for each stage
itt - candidates in order of election/exclusion, reported at each stage

The options 'verbose' and 'plot' have been mentioned already under 'Usage'. If the plots and webpages are to be kept, 'outdirec' should be set to the desired directory.

The actual vote count calculations are performed in the function *transfer* which revises the keep values k at each stage. This is done by solving a polynomial equation in the only non-trivial transfer values $t = 1 - k$, *i.e.* those of already elected candidates, with coefficients calculated by summing over the contributions of each vote using function *share*. A function *select* provides the binary coding of numbers $0 : (2^s - 1)$ required to store the polynomial coefficients neatly. [This use of polynomial coefficients avoids having to go through the vote file more than once per stage, speeding up the calculation by an order of magnitude.]

Following transfers, the stage is completed by the function *decision*: if any addtional 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).

Plots and web pages are calculated by functions *voteplot* and *webpages* respectively.

Functions associated with data input

The input format required by *stv* is a list *ed* with 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 function *stv.data* converts some common preference data formats into the above format:

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

The options here cover:

mult - if T, the multiplicity of each row is given as its first element
details - if F, data are simply a 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 F, indicates that first row is c, s, and candidate names and election title are at end of file

Other input functions are *abbrev*, to calculate a suitable abbreviated name, name2; *party_colour*, to associate party colours for 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.

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.

Examples

The first two examples come from Scottish Council elections. Note that the official count used a different STV method, 'Weighted Inclusive Gregory' (WIG). The web pages noted below provide both the official count and one using Meek STV, with links to switch between the two for ease of comparison.

hc12 Helensburgh Central 2012 A very simple example where all seats were filled from first preferences, so that no specialist software was needed!

p17 Partick East - Kelvindale 2017 A more typical example of an STV council election.

j02 John Muir Trust 2002 An example from an election allowing equal preferences.

yale A faculty election from Yale University An example with a large number of candidates, 44 for 4 places. While the graphics cope, they are difficult to read, so ...

y12 'Yale - last 12' ... this version, starting after 32 candidates have been excluded (but noone yet elected) is provided, to give clearer plots of the later (and decisive) stages of the election.

Further examples can be found at www.macs.hw.ac.uk/∼denis/stv_elections

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.

Feedback

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

denis.mollison (at) gmail.com