EES2019 Stacking Process

Generic Distance/Proximity Variables Estimation

Giuseppe Carteny

09.09.2021

Contents

1 Introduction

Our next tasks consist in (1) the estimation of generic **distance/proximity** variables and (2) the preparation of the SDM codebook.

The workflow follows up the one used for our previous task: we will modify the 'genvars' country-specific scripts developed a few days ago, adding the computation of the distance/proximity variables.

2 Estimating Generic Distance/Proximity Variables and Codebook Preparation

As suggested by their name, distance/proximity variables concern the distance or proximity between the voter and the set of relevant parties on some dimensions of interest. Thus, such variables measure whether and to what extent the position of an individual is more or less close to/distant from the position of a party on general dimensions (for istance, the left-right continuum) or on specific issues (such as European integration, immigration policies, minority rights, and so on).

As shown below, we will estimate distance rather than proximity variables. Thus, the higher the values of such variables the more distant a specific voter and a specific party are. The only exception is the propensity to vote (PTV) variable, that as suggested by the name itself measures an individual propensity to vote for a given party. This variable measures the "proximity" between voters and parties, thus the higher the values of said variable the closer voter and party are.

2.1 Distance as Absolute Difference

With the sole exception of the PTV variable, this kind of generic variables needs to be estimated. Theoretically, for estimating distances between voters and parties we could use a geometric distance formula:

$$d = \sqrt{(x_i - x_k)^2 + (y_i - y_k)^2}$$

Where x_i and y_i represent the positions of individual i on issues x and y, and x_k and y_k represent the position of party k on the same issues. However, since we are computing individual-party distances on a single issue, this implies that we are estimating these distances on a single axis. Thus, we can drop the second half of the formula below the radical symbol, rewriting it as it follows:

$$d = \sqrt{(x_i - x_k)^2}$$

We could use this formula for estimating our variables. Nonetheless, such variables are usually computed exploiting a similar formula, avoiding the radical and using the absolute difference of individual and party positions, that is:

$$d = |x_i - x_k|$$

This is the formula that we will use in our scripts for computing our generic variables, PTV excluded.

2.2 The `gendis.fun` function

As in the case of dichotomous variables, for computing generic distance variables I implemented a function that will make our job much easier. This function (`gendis.fun`) is slightly more elaborated than the one used before.

In short, it will take the value of an individual on a given dimension (say, respondents' self-placement on the left-right ideological axis, variable `Q11`) and the value of a party on the same dimension *as perceived by the same individual* (hence, an idiosyncratic value as specified in the `Q13` variables set) or *as perceived by all the respondents* (thus, an average value of respondents' scores on the same set of variables). These two methods are only some of those used for computing such distance variables, but for now we will rely on these two, and more specifically on the second method.

Returning to the function, as you will see it takes several arguments:

. `data` = The main data basis for computing our variables. Essentially it takes only one value, that is the 'EES2019' (country-specific) original data frame, thus this argument could be avoided. However, the function might be generalized in the following weeks to allow different data sources, that's why that argument is there;

. `cdbk` = The auxiliary data frame for computing our variables. As above, it takes only one value, that is the 'EES2019_cdbk' (country-specific) data frame. However, also this argument is there because of possible implementations. In any case, the reason why we use the EES codebook is that it allows to automatically select the EES2019 columns referring to party positions. This leaves our hands free from manual coding (that is always a good thing);

. `vrbl` = This argument so far takes three values: `Q10`, `Q11`, and `Q23`. With the sole exception of the PTV variable (`Q10`), when specifying the other two self-placement variables the function automatically identifies the party-specific variables, using the EES2019 codebook correspondence tables;

. `crit` = This argument refers to the criterion used for computing the distance variables, namely whether the measure should be computed using an 'idiosyncratic' approach or using the average of party positions (see few lines above). Thus, it just take two values: '`idiosyncratic`' or '`average`'.

. `rescale` = All the variables that we are dealing with are measured on a 0 to 10 scale. Nonetheless, such values are often rescaled in order to fit into the `[0,1]` interval. This argument, that is a logical argument (`TRUE` or `FALSE`), simply allows us to specify whether we want to rescale the values or not;

. `check` = Likely this is the most useful argument for developing your script. It is another logical argument. If `TRUE` then it will return (as a list object) the final output (a data frame) but also an intermediate data frame that will allow you to check the original values used for estimating the distance varialbe. If `FALSE` it will return just the distance variable (however, see also the last argument of the function, a few lines below). I suggest to set the argument as `TRUE` during the development of your scripts and then to `FALSE` when you are sure that everything went fine;

. `keep_id` = This (logical) argument ask you whether the output should be a data frame containing the estimated distance variable and the `respid` and `party` identification variables (`TRUE` value) or a single column data frame with just the generic variable. This option essentially allows you to use the `gendis.fun` in different workflows. When `TRUE` you can merge the estimated variable with the SDM using a `dplyr *join` function (such as `left_join()`). When `FALSE` you can simply bind the new column to the SDM.

2.3 Procedure for computing generic distance variables

The procedure for computing our distance variables is rather similar (in terms of code) to the one used for estimating the dichotomous generic variables. For implementing your scripts you just have to run first the 'EES2019_stack.R' without the last section ('Estimate the generic variables'), and then run your country-specific 'genvars.R' script, without the last section (that is, without 'tidying' the environment).

If you want to estimate the generic variable and check the results, you can just run the function, setting to `TRUE` the `check` argument:

```
x <-
  gendis.fun(data = EES2019_it,
                   cdbk = EES2019_cdbk_it,
                   vrbl = 'Q11',
                   crit = 'average',
                   rescale = T,
                   check = T,
                   keep_id = F)
```

Then, if everything works fine, you can develop an additional paragraph just below the one dedicated to the dichotomous variables[1].

```
EES2019_it_stack %<>%
  cbind(.,
        gendis.fun(data = EES2019_it,
                   cdbk = EES2019_cdbk_it,
                   vrbl = 'Q11',
                   crit = 'average',
                   rescale = T,
                   check = F,
                   keep_id = F)) %>%
  as_tibble()
```

Of course, we could add more variables to the chunk above, just copy-pasting the `gendis.fun` function and changing the argument `vrbl`, or we could apply such function to a vector containing the variables of interest, then binding the results, and merging the output with the SDM. The latter is the solution that you will actually find in the exemplary script, as shown below:

```
EES2019_it_stack %<>%
  cbind(.,
        lapply(data = EES2019_it,
               cdbk = EES2019_cdbk_it,
               crit = 'average',
               rescale = T,
               check = F,
               keep_id = F,
               X = list('Q10','Q11','Q23'),
               FUN = gendis.fun) %>%
          do.call('cbind',.)) %>%
  as_tibble()
```

[1]Actually you could also put the `gendis.fun` in the same chunk of code used for estimating the dichotomous variables with the `gendis.fun`. For sake of clarity, I personally prefer to keep the two estimations separated.

In any case, once computed the variables we just have to tidy up the environment as already specified in the current version of our scripts.

2.4 Codebook development

Once finished the estimation of the variable(s) of interest, then the next step is to insert new entries in the SDM codebook, that you can find as a R Markdown script ('Codebook.Rmd') in the '~/EESstacked/Docs/docs_scripts' subdirectory.

There's not much to say about *how to do it*, since you can rely on what has been done previously. But I recommend you to be careful about the indentation of the R Markdown script. In particular, when specifying the values of the variables of interest remember to leave *two blank spaces* at the end of the row. This tells to `rmarkdown` to enforce a *line break*. You can find more about this topic (and other R Markdown features concerning text) on this webpage, in particular in the 'Preventing and enforcing line and paragraph breaks' subsection (Sect. F.3.2).

About the *content* of the paragraphs I suggest you to drop me an email.

2.5 Which Variables

The list of variables that we are going to use for computing our generic distance variables are the following:

- `Q10`: Respondent's propensity to vote for a specific party;
- `Q11`: Respondent's self-placement on the left-right *continuum*;
- `Q13`: Respondent's perception of a specific party position on the left-right *continuum*;
- `Q23`: Respondent's position about EU integration[2];
- `Q24`: Respondent's perception of a specific party position about the EU integration process.

3 Who Does What

Like for the last task, I will take care of Belgium, Bulgaria, Cyprus, and Italy, while you will take care of the following countries:

- **Willie**: Denmark, Estonia, Germany, Luxembourg, Malta, Netherlands, Spain, United Kingdom;
- **Julian**: Czech Rep., Finland, Greece, Hungary, Lithuania, Slovakia, Poland, Sweden;
- **Matthias**: Austria, Croatia, France, Ireland, Latvia, Portugal, Romania, Slovenia.

This time **Matthias** will fill the codebook with a set of entries referring to the generic variables generated from the variables mentioned above.

[2]In the original EES 2019 Master Questionnaire reference is made to EU *unification* rather than *integration* process. Nonetheless, the two concepts can be considered almost equivalent.

4 The Deadline

I believe that also in this case a few days should be enough to finish the tasks listed above (deadline: **13.09.2021**).