Ordinal Variables for Blocking

July 4, 2019

Committee meeting May 21

- I thought my Table 2.2 shows evidence that OP is better than making things numeric. It doesn't. I am using a placebo treatment, i.e. there is no treatment, so the Group T2 coefficient should be zero. In other words: There would be differences with anything here, since it's just one single draw. I need evidence that OP works better
  - Evidence 1: Monte Carlo simulations
  - Evidence 2: Repeat the Table 2.2. estimations 100 times and show the distributions around zero
- Perform a test to check that OP assigns people correctly
- Write up the `shiny` survey environment in the appendix
- Emphasize machine learning in the write-up

`shiny`

- The only current option to design an online survey environment is through a provider like `Qualtrics`. You can randomize there, but only by clicking "Randomize". It is not possible to use blocking online. I create a survey environment which can do that
- I found the package `ShinyPsych`. It creates a survey environment with `shiny` (among other things)
  - Overview
  - How to create a survey
  - Specifications of `.txt` files for questions
- A survey environment is created by running the code in the file `app.R` in the package folder `ShinyPsych/shiny-examples/Survey`. This code reads in questions in `.txt` format from the package folder `ShinyPsych/extdata`. The `.txt` files need to be very specifically formatted (a question goes into one cell, all answer options go into one cell etc.). To adapt everything for my needs, I need to write my own questions as `.txt` files and I need to edit the code in `app.R`
- It's a pain to edit a `.txt` question file. It's much better to turn it into a `.csv`, edit that, then turn it back into a `.txt`
- I made a copy of `app.R` called `shiny_psych_survey.R` that I will continue to edit. Running `shiny_psych_survey.R` creates my survey environment setup. Running `app.R` creates the the original package example survey environment
- Questions as `.txt` files:
  - I copied an existing `.txt` file and edited it to contain an example of the questions I need. So far, I created the following files in `/questions`:
    - `code.txt`
    - `education.txt`
    - `goodbye.txt`
    - `instructions.txt`

    - `demographics.txt`

  And the following ones in `/questions/treatment`:

    - `mw.control.txt`
    - `mw.m.opp.txt`
    - `mw.m.supp.txt`
    - `mw.p.opp.txt`
    - `mw.p.supp.txt`
    - `tb.control.txt`
    - `tb.m.opp.txt`
    - `tb.m.supp.txt`
    - `tb.p.opp.txt`
    - `tb.p.supp.txt`
  - I copied the code from `app.R` into `shiny_psych_survey.R` and adapted it to load my question files
  - Crucial: The code to load the question files needs to include any file path and `.txt` as well as have `defaulttxt = FALSE`. Otherwise these external files will not be read in by the code
- Saving data
  - I can save locally and to Dropbox
  - Locally
    - Use the `saveData` function from `shinyPsych` and set `location = "local"`. That saves each response as a `.csv` in whatever `outputDir` is
  - Dropbox setup:
    - Much more complicated
    - Use the `rdrop2` (link) and `dplyr` packages
    - Setup things:
      - Sign into my AU Dropbox in the browser
      - Generate a Dropbox access token
      - Pull and save the access token to the same folder where `shiny_psych_survey` is located:

        ```
        library(rdrop2)
        token <- drop_auth()
        saveRDS(token, "droptoken.rds")
        ```
    - Create the function `savedata` to upload a `.csv` to Dropbox folder `/block_data` (instructions are here)
    - Create the function `loaddata` (instructions are here) to download all the `.csv` files from Dropbox `/block_data`, turn them into a list, then a data frame, then save that data frame in my local folder `/block_data`
    - `savedata` needs to be defined before the actual `shiny` code and is executed in the app code by adding `savedata(data.list)` instead of the original `shinyPsych` data saving code
    - `loaddata` is defined and executed from `edit_package.R`
- Randomly assigning respondents to one of five treatment groups
  - Took me a lot of experimenting, but the end result is very nice and simple
  - For each issue, create an `R` vector with the names of the group `.txt` files, then

randomly sample one of those names and store it. Then paste that stored object name into the `createPageList` function and create the needed objects in the other sections
  - Each input question in each treatment `.txt` needs to be labelled "treat", otherwise the data can't be read. With this generic labelling, it is not shown what group the respondent was assigned to in the resulting data frame. That's why I added the stored randomly sampled group name for each issue as a variable to the list of data saved in `data.list`
- `edit_package.R`:
  - Code to edit question files
  - Code to source `shiny_psych_survey.R` and run the app to test that the questions are properly loaded. Responses are saved to my AU Dropbox folder `/block_data`
  - Code to re-deploy the app on my `shiny` server account
  - Code to create function `loaddata`
  - Code to execute `loaddata`
  - Code to source `ShinyPsych`'s `app.R` if I want to look up some things in the original version. Responses are saved locally under `/package_data`
- `shiny_psych_survey.R`:
  - Code to create function `savedata`
  - Code to create the setup of my app survey environment
  - No additional code for authentication with Dropbox needed. The original setup above is enough for use on my local machine
- Deployed app on my shiny server account: https://sheuberger.shinyapps.io/survey_experiment/
  - Everything is in `/survey_experiment`:
    - `app.R`
      - Code to create authentication with Dropbox (adapted from here, not needed locally but needed when deployed to create Dropbox authentication and save the data):
        `drop_auth(rdstoken = "droptoken.rds")`
      - Code to create function `savedata`
      - All app code
      - Data saved to AU Dropbox `/block_data`
      - Randomized treatment questions for both issues
      - Code set up so I have to adjust only the bare minimum
    - `questions/code.txt`
    - `questions/education.txt`
    - `questions/instructions.txt`
    - `questions/demographics.txt`
    - `questions/pid_foll_dem.txt`
    - `questions/pid_foll_ind_else.txt`
    - `questions/pid_foll_rep.txt`
    - `questions/treatment/mw.control.txt`
    - `questions/treatment/mw.m.opp.txt`
    - `questions/treatment/mw.m.supp.txt`
    - `questions/treatment/mw.p.opp.txt`

    * questions/treatment/mw.p.supp.txt
    * questions/treatment/tb.control.txt
    * questions/treatment/tb.m.opp.txt
    * questions/treatment/tb.m.supp.txt
    * questions/treatment/tb.p.opp.txt
    * questions/treatment/tb.p.supp.txt
    * droptoken.rds
- shiny_psych_survey.R is for my local machine where I can test and make changes as I go along. app.R in /survey_experiment is the live survey on the server. If I make any code changes to shiny_psych_survey.R that I want to see in the deployed app, I need to manually copy it to app.R. This is for safety, so I don't accidentally overwrite it with stupid things
- The same goes for the questions. All question files are under /questions for my local machine. There are separate versions of them in /survey_experiment. Any changes need to be copied manually
- Over the course of development, I massively changed the original code in shiny_psych_survey.R so that I had to adjust only a minimum of things as I move forward (I had to keep going through and changing all the cases over and over again). I have commented things out and also saved the old version. In the current one, only the sampled questions need to be adjusted throughout (otherwise I would keep re-sampling). The rest is set once at the beginning of the document and then called with paste, assign etc.
- Include OPM function OPMord (from OPM above) in existing shiny environment
  - This is actually not necessary because the OPM does not need to be part of the shiny environment. Users run OPMord with data, DV, and EVs of their choice to obtain data-driven ordinal categories. These categories then replace the original categories in ordinal variable in the user's experimental data. She can then use these categories as the basis for sequential blocking. Basically, OPMord is run before anything in shiny and thus does not need to be included in it
- Modify shiny_psych_survey.R to include/run seqblock
  - Overall idea: Feed the user-selected category into seqblock(). If the MD code doesn't set in yet, the respondent is randomly assigned to a treatment group. If the MD code sets in, all previously assigned education data is loaded from Dropbox, and the code blocks the respondent into a treatment group. In both cases, the respective group is extracted, the respective treatment .txt is saved as mw.sample/tb.sample, and the the newly updated assigned data is saved to Dropbox
  - seqblock() only works with .RData files and only stores the variables that are blocked on. So saving the seqblock data and saving all input data needs to be separate. seqblock data goes to Dropbox/seqblock, all input data goes to Dropblox/alldata. The seqblock data needs to be overwritten on Dropbox and within shiny, so it's one .RData file. The input data is one small .csv file for each user that I later download and combine on my laptop (so exactly the same as before)
  - seqblock() creates a .RData file for the first person. All following people are

assigned based on that continuously updated file. The website code doesn't detect whether someone is the first person or not. As a work-around for now, I put in my education category a the first user by running `seqblock()` once before the `shiny` environment and uploading that `.RData` file

- I need a reaction when users click "Continue" on the education page (i.e. they select their category, then the blocking code sets in and draws/uploads stuff from/to Dropbox). Thats' done via `observeEvent()`. I also need to save an output for use in a later function when users click that button (i.e. the assigned treatment group, which determines which treatment page is shown). That's done via `eventReactive()`. I don't know how to combine them, so each issue has one `observeEvent()` and one `eventReactive()`
- Order of code
  - `sequpload()`
    - Function that uploads a `.RData` file to Dropbox/seqblock
  - `seqdownload()`
    - Function that downloads a `.RData` file from Dropbox/seqblock
  - `observeEvent()` when hitting "Continue" on education page
    - Download `.RData` (`seqdownload`)
    - Load `.RData`
    - Run `seqblock()` on user-selected category and loaded `.RData`
    - Upload `.RData` (`sequpload`)
  - `eventReactive()` when hitting "Continue" on education page
    - Download `.RData` (`seqdownload`)
    - Load `.RData`
    - Save assigned treatment group to object for later use to display correct treatment page
- App is deployed and working. The whole `.RData` downloading/uploading works. It only causes a few split seconds of loading before the web page displays the next question. Except for the issue where I have to be the first user, I think this looks good

- Improve the code
  - Am I using the right variables in `seqblock` (I use `exact.vars` but there is also `covar.vars`)?
    - Nope. `exact.vars` requires exact variables. So if an incoming person has a 2 but a treatment group is made up of 1s and 3s, it won't find anything to match
    - I want `covar.vars`, which matches on the distribution
    - I switched the code to `covar.vars`
  - Find a setup where I don't have to put in my own data to create the first person
    - Done. I used the `drop_exists` function and wrapped it in `if ... else`. `drop_exists` tests whether a file exists in a Dropbox path
    - If the mw `.RData` file exists on Dropbox/seqblock – which means other respondents have been assigned before this current respondent – the code now reads in that data, blocks on the data and the current respondent, then uploads the updated `.RData` to Dropbox/seqblock

    - If the mw `.RData` file doesn't exist – which means the current respondent is the first respondent to fill in the survey – the code now blocks only on the current respondent, then uploads the resulting `.RData`
    - Obviously the same repeated for any/all other issues
    - App is deployed and working
  - Added `pid` to block on as an `exact.vars`
  - Incorporate skip logic (if "Republican", give question "Strong or weak?") with `eventReactive()`
    - I used the same technique I used for the two issues. I created separate question `.txt` files for the party ID follow-up questions (one for Dem, one for Rep, one for Ind/Something Else)
    - Depending on the number that represents what the respondent selected, the code displays the corresponding follow-up `.txt`
  - Test the blocking code with the Dropbox upload/download code
    - I want to make sure that everything really does result in balance and that the Dropbox stuff doesn't mess things up somewhere somehow
    - Ideally, I would like to get randomized survey responses. Ryan said there was a way to make the computer fill in responses randomly, but I couldn't find anything on that
    - I don't think this test has to be in the finished survey. I want to simulate the blocking part including the uploading/downloading to Dropbox – I can do that locally on Jeff's machine. I'm also including the `.csv` saving code. It should look identical to the `.RData`, but I just want to make sure there's nothing muddled up there
    - I started doing that on Jeff's machine. It worked well for n = 1,000 for `education` and `pid`. So the Dropbox stuff doesn't mess anything up and we get balance
    - One weird thing: The blocked `education` numbers have decimals. It's something to do with `seqblock()`. I printed a screenshot of the blocked `bdata$x`
    - Show Ryan `bdata$x` with decimals and my code (all printed)
    - For reasons I don't understand, `bdata$x` includes a `jitter()` part, hence the decimals (or noise) around the values
    - Change `bdata$x` to `bdata$orig`, which keeps the original values in `shiny_psych_survey.R`, the deployed `app.R`, and `testing_blocking_dropbox.R`. Then rerun the test for 1,000 simulations
    - Reran test for 1,000 simulations – everything looks good
- Work in redirects for Lucid
  - The Lucid folks need my survey to read in a respondent-based unique alphanumeric number (called `RID`) before anything is loaded. This number is created in their system when a respondent clicks on my survey link and added to the web link, creating an initial Lucid link. For the placeholder `RID` `123456-abc`, the initial link is `https://sheuberger.shinyapps.io/survey_experiment/?rid=123456-abc`. My survey needs to read in `123456-abc` from that link
  - The Lucid folks also need my survey to pass out the read-in `RID`. This means that my survey needs to redirect to a Lucid completion link with the `RID` in it once all

questions have been answered. For the placeholder `RID 123456-abc`, the completion link is `https://notch.insights.supply/cb?token=98b98d10-789d-42ec-ba71-a077 cbbd909c&RID=123456-abc`

- It took a lot of emails and a Zoom meeting to figure out the above details. Here is what I did:
  * I took out the `GoodBye` page (all the code plus the question files) and added a Thank You message on the `Code` page
  * To read the `RID` in: `shiny` stores the current session data, which includes everything in the URL. In order to read in the `RID`, I needed to access the component of the URL in the session data with the `RID` in it. It turns out that this URL component is the query string, which is the stuff after the question mark (it's all described here). The `shiny` session location for that is `session$clientData$url_search`. To read that in, it needed to be wrapped into `parseQueryString()`. The resulting R object is a list, so it needed to be made a character with `[[1]]` (explained here and here). I set all that up
  * I added the stored `RID` to the `data.list` I save on Dropbox, so I can see what is being initially read in
  * To pass the `RID` out: I defined a Javascript function, called `js$browseURL`, that automatically redirects respondents to a website (code taken from here). I set it so that respondents are automatically redirected to a specified website when they hit "Continue" after typing in the survey code. I set the specified website to Lucid's completion link, with the stored `RID` pasted in
  * I also added two `if...else()` wrappers for the reading-in and the passing-out code, so that the same code can be run if there is no query string (which is when I run it) and if there is a query string (which is when Lucid runs it). The background here is that the pulled-in list is empty if there is no query string, which means you can't subset it with `[[1]]` because it throws an error. So now the code stores the query string if there is one and stores "no.query.string" if there isn't. It also redirects to Lucid's completion page in the first case and to Google in the second
- Lucid successfully tested the reading-in and passing-out code on April 22. The technical setup is now ready for the launch with Lucid

`ordered_outcomes_modeling`

- Note: I don't have to invent new R code to block, either normally or sequentially, because OPM comes before blocking. Ryan already developed R code to block normally and sequentially (`seqblock`) using MD. `seqblock` is set up so that the MD blocking code doesn't set in for a specified 'first few people' to be assigned. They are assigned randomly. Then, at a code-specified point, the MD code sets in and takes over for everyone else. Use that and the 'normal' blocking function to block after OPM is applied
- Test OPM model
  - Block on the original 10 ANES education variables, then run OLS regression on some ANES outcome. Then do the same for the 5 re-estimated OPM categories and compare the differences in results. If there were no differences, this would

raise some doubts whether OPM is really necessary
    * `opm_create_model.R` loads the ANES variables, runs the OPM model, saves `.csv`, `.pdf` (I used both of these for the presentation), and `.rds` files
    * `opm_test_model_100.Rmd` loads the `.rds` files, blocks and runs OLS for 100 ANES observations. Outputs a regression table as a `.pdf`
    * `opm_test_model_all_jeff.R` does the same, but for all ANES observations. This is a separate `.R` file because I ran this on Jeff's computer (the loop takes a while on mine). Outputs another set of `.rds` files
    * `opm_test_model_all.Rmd` loads the extra set of `.rds` files. Outputs a regression table as a `.pdf`
    * Most EVs are the same for both regressions, which makes sense. They haven't been modified, so they shouldn't be different. The treatment group coefficient, however, is different. The groups were blocked on education, so that's definitely the effect of the different categories. I think this is different enough to justify OPM. Jeff agrees, so this is all good
- Set up an R function (or functions, whatever is better)
  - I set up `opm_create_function.R`, which creates the function `OPMord`, which:
    * Applies `polr()` to specified training data, DV, EVs
    * Attaches binned cases
    * Creates a new DV with re-estimated categories
    * Attaches new DV to the originally supplied data
    * Saves several objects as outputs as a list, to be called with $
  - I saved `OPMord` as `OPMord.RData` so I can load into any R session
  - The re-estimated levels outputted by `OPMord` can be used by `block` and `seqblock` for any data. In my case, I will block my data on the education levels that result from training the OPM on the ANES data. The OPM part stops after `OPMord` has been run and the re-estimated levels have been obtained
- Simulate things by including different groups as the EVS and see how that affects the estimated new number of categories
  - If run with only one variable as EV, these are the resulting number of categories:
    * `race`: 4 estimated new categories
    * `income`: 3
    * `occupation`: 3
    * `gender`: 2
    * `pid`: 1
    * `age`: 1
  - Using just one variable as EV is silly, but it gives a good indicator of the influence of each variable. I played around with a few different combinations, and that confirmed: You get the same 5 categories just with `race`, `income`, and `occupation`. The others don't affect the overall new number
- `$lp` v. `$fitted.values`
  - I'm using `$lp`. Ryan said that I could also use `$fitted.values`, which lists the probabilities of assignment for each observation for each education category
  - After investigating a bit, we found that `$lp` gave 5 categories but `$fitted.values` only 4

  - I calculated the percentages of observations within each category (all in `opm_create_model.R`) and found the results very confusing. I sent them to Ryan. He didn't respond, and it's been ages, so I'm just going to leave this be
- Do Monte Carlo simulations
  - MVN data with some cross-correlations
  - I know the true treatment effect here
  - So when I then block on categories 1:15 and 1:5, I know which one performs better – the one that's closer to the true treatment effect
  - Set up in `opm_monte_carlo.Rmd`
  - Still needs tweaking, though I'm not sure in which way(s)
  - Further space out `theta` and adjust rounding of `theta.star` to achieve greater difference (play around)
  - Sent it to Jeff
  - Space it out some more; play with the sample size; modify the covariate generation with more cross-correlation
    - I spaced `theta` out in many different ways and played with the sample size – barely any change
    - `Y` is based on `theta` and `X` comes from the ANES correlation matrix
    - I could manually change the correlation matrix, but then it wouldn't be the ANES correlations any more. Is that what he meant?
    - How can I make `Y` correlate more strongly with `X` but still create `Y` from `theta` and `X` from the ANES?
    - Try link Jeff sent me (first steps in R file)
    - The results never really change, no matter what I do with the correlations
    - I implemented exactly what was in the link; now Jeff tells me that this doesn't create realistic data and doesn't go anywhere. Then what was the point of the link?
  - Read through the resources Jeff sent me
    - https://scholar.cu.edu.eg/sites/default/files/mohamed_abonazel/files/how_to_create_a_mon
    - https://statweb.stanford.edu/ owen/mc/
    - http://citeseerx.ist.psu.edu/viewdoc/download?doi=10.1.1.703.5878&rep=rep1&type=pdf
- Repeat and visualize Table 2.2
  - The table is created with a placebo treatment, i.e. there is no real treatment. That means Group T2 should be zero. The table is not evidence that the OP method is better
  - Repeat the estimations for Table 2.2 100 times and visualize the distribution of the Group T2 coefficients. This will show which category estimation is closer to zero, which is the true value of that coefficient
  - I set all this up in the `opm_test_model_regression_100_obs` and `opm_test_model_regression_all_obs` files
  - The setup works for 100 observations (to test it) and all observations
  - I ran 100 repeats for all observations
  - I ran 1,000 repeats for all observations
- Write up the `shiny` environment in the appendix
- Emphasize machine learning in the write-up

  - Model training is what it's all about
  - People are being misclassified based on arbitrary numeric values

Eventual package setup

- Leave this until the very end. It's interesting, but I don't need it for the diss
- For my framing experiment, I use education and ANES to train the OPM, then I block on education and other covariates with `blockTools`. The package will be applicable much more generally: Set the package up so that the user specifies/loads the data, the DV, and the EVs to use for training. My functions then apply polr(), attach binned cases, create a new DV with re-estimated categories, and attach this variable to the originally supplied data. Then the user can block on the resulting variable with `blockTools`
  - `OPMord` comes before any blocking
  - `OPMord` output can be used to `block` normally in interactive R
  - `OPMord` output can be used to `seqblock` sequentially in `shiny` environment
- I currently don't have any 'package' folder because it doesn't make sense to set that up before the functions are developed
- Blog post to include `shiny` in R package and launch app from within package