Code Samples

PJ Paul

December, 2018

Contents

List of source codes

1 Mortality Simulation Code

The code in the following two pages was written in the context of an aquaculture impact evaluation I was working on. Our client wanted to test the effectiveness of two different feeding regimen as measured by the impact on fish growth and mortality.

Making this assessment accurately was challenging since capturing and weighing all the fish in the catchment *individually* would be physically impossible. The team decided to proceed as follows: (1) Sample 50 fish, and weigh them individually to arrive at the estimated average fish weight; (2) Harvest and weigh all the remaining fish together. This gave us the estimated average fish weight, and the true total fish weight. We could then divide the total fish weight (2) by the average fish weight (1) to arrive at the estimate of the surviving number of fish.

While this reasoning made intuitive sense, the team was hesitant to proceed with it without understanding the error bounds for our estimates. We generated bootstrap errors for these estimates using the code in the following two pages. The following results from the bootstraps gave us the confidence to go ahead with our planned procedure.

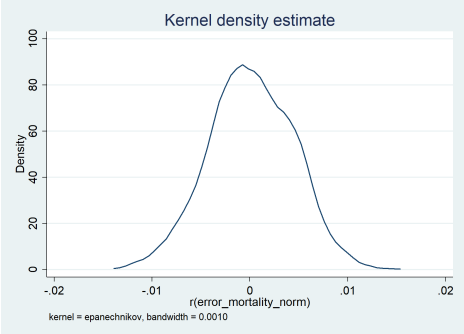

Figure 1: Simulation under Normal distribution

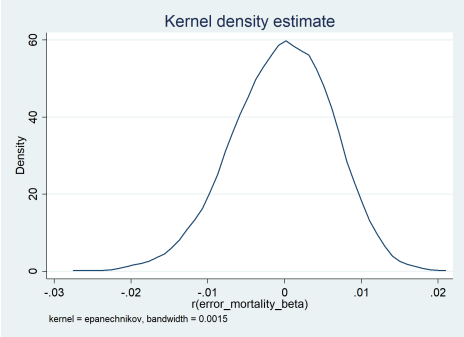

Figure 2: Simulation under Beta distribution

```
cap program drop normalsim

// Define the program
program define normalsim, rclass
    version 14
    syntax [, count_end(integer 1) count_begin(integer 1000) mu(real 0) sigma(real 1)]
    drop _all
    set obs `count_end'
    tempvar weight t_weight a_weight


    gen `weight' = rnormal(`mu',`sigma')
    local total_weight = sum(`weight')

    sample 50, count
    count
    return scalar count_n = r(N)
    // I return this just to check that sampling is correct
        summarize `weight'
        egen `a_weight' = mean(`weight')
        local avg_weight = `a_weight'[_N]

        local est_count_end  = `total_weight'/`avg_weight'

    local est_mortality = 1 - (`est_count_end'/`count_begin')
    local true_mortality = 1 - (`count_end'/`count_begin')

    return scalar error_mortality = `est_mortality' - `true_mortality'
    end

// Run the program
simulate error_norm_1 = r(error_mortality), reps(100): normalsim, ///
    count_end(500) count_begin(1000)  mu(20) sigma(3)
```

Listing 1: Stata program to simulate weights under Normal distribution

```
cap program drop betasim
program define betasim, rclass
   version 14
   syntax [, countel(integer 1) countbl(integer 1000) ///
   alpha(real 0) beta(real 1) var_kb(real 3)]

       // Figure out the variance to plug into
       // the equation to get the proper scale factor
       local variance_x = ///
           (`alpha' * `beta') / ( ((`alpha'+`beta')^2) * ///
           (`alpha'+`beta'+1) )
       local scale = sqrt(`var_kb'/`variance_x')

       // Round the variance for the purposes of graphing
       local var2= round(`variance_x',.01)


       set obs `countel'
   tempvar weight t_weight a_weight
   gen `weight' = rbeta(`alpha',`beta')* `scale' +20- .5*`scale'
   local total_weight = sum(`weight')


       sample 50, count
   return scalar count_n = r(N)
   egen `a_weight' = mean(`weight')
   local avg_weight = `a_weight'[_N]
   local est_count_el  = `total_weight'/`avg_weight'
   local est_mortality = 1 - (`est_count_el'/`countbl')
   local true_mortality = 1 - (`countel'/`countbl')

   return scalar error_mortality_beta = `est_mortality' - `true_mortality'
end
```

Listing 2: Stata program to simulate weights under Beta distribution

2 Python Scraping Code

As part of a personal project on how the presence of criminal incumbents affect downstream electoral competition in their constituencies, I needed to scrape the data on competing candidates from the website of the state's election oversight authority. However, the website provided the data I needed only in the form of a sequence of dynamic Javascript-triggered drop-down menus, with one drop-down for each level of administrative aggregation starting from the district and going all the way down to the ward of a constituency. In order to extract the data I required, I leveraged my knowledge of Python, and taught myself web-scraping using Selenium. In the code sections that follow, I highlight key sections of the code. The full code is over 400 lines and can be accessed at my Github repo.

The first section of the highlight is relatively straightforward, it waits till a particular drop-down is loaded, and then selects one option from the drop-down.

The second section of the highlight was added because I was running the scraper on an extremely slow network, which would result in frequent time-outs from the server. Each time the server timed out, my code would have to start all over again from the first index. In order to avoid this, I saved all the scraped data into csv files at regular intervals, and then used a `pickle` object to record the completion status of each unit to be scraped.

The final section of the highlight (spawn and run new instances), was added to deal with the problems from slow network speed. Specifically, when the server timed out, my code would return an error, and stop. Since I was running the code on my personal laptop overnight while I was asleep, I needed a way to ensure that in case of an error, the instance of the program with the error would quit, and a new instance would be spawned. Hence, the code in Highlight 3.

```
## Highlight 1. Code to select one drop-down element (Lines 48-55)

def get_district_select(self):
    path = '//select[@id="ddldistrict"]'
    WebDriverWait(self.driver, 20).until(
        EC.presence_of_element_located((By.ID, 'ddldistrict'))
        )
    district_select_elem = self.driver.find_element_by_xpath(path)
    district_select = Select(district_select_elem)
    return district_select

## Highlight 2: Code to make and access caches (Lines 298-310)
    # Create and update cache

def result_export(df, filename):
    try:
        result = pd.concat([pd.DataFrame(df[i]) for i in
                            range(len(df))],ignore_index=True)
                    # if file does not exist write header
        if not os.path.isfile(filename):
           result.to_csv(filename,header ='column_names')
           print("CSV Export overwrite")
        else: # else it exists so append without writing the header
            result.to_csv(filename, mode = 'a',header=False)
            print("CSV Export append")
    except:
        traceback.print_exc()
        pass

    # Use a pickled object to access completion status
def dist_status_write_update(district,status):
    '''
    After all entries in a district have been entered,
    Update the status of the district to 1.
    '''
    try:
        dist_status = pickle.load(open("dist_status.pickle", "rb"))
        dist_status[district] = status
        pickle.dump(dist_status, open("dist_status.pickle", "wb"))
    except (OSError, IOError) as e:
        dist_status = {}
        dist_status[district] = status
        pickle.dump(dist_status, open("dist_status.pickle", "wb"))

## Highlight 3: Code to spawn and run new instances automatically (Lines 416-426)
if __name__ == '__main__':
    signal.signal(signal.SIGINT, sigint)
    while True:
        try:
            scraper = Scraper()
            scraper.scrape()
        except Exception as ex:
            logging.info("Caught exception {}".format(ex))
            print(ex)
            traceback.print_exc()
            scraper.driver.quit()
```

Listing 3: Highlights from the Python Selenium scraper

3 Spatial Analysis using POSTGIS

As part of the earlier mentioned project on impact of criminal incumbents, I needed to extract socio-demographic features for each constituency in my target state. Performing a simple merge using Stata/ Python was not helpful due to the inconsistency in geographic names, and the absence of unique identifiers. I decided to use a spatial solution, where I would spatially join/ merge the socio-demographic data from the Census shapefiles, to the constituency boundaries in the Constituency shapefile.

My initial attempts to perform the spatial join in QGIS were not successful due to the limited computing power on my personal laptop. I taught myself PostGIS to use a database oriented solution to the problem which is illustrated through the code highlights 1 and 2.

Both highlights 1 and 2, essentially perform the same operation. They take in two tables specified in the `FROM` statement, namely `wb_census_2011` and `wb_ac_map`. Then they perform a spatial merge to link these tables using different criterion. Highlight 1 matches census villages to constituencies by first calculating the centroids of the census villages, and then matching the census villages to the constituencies where such centroids falls. While this might be seem satisfactory at first glance, several constituencies have highly non-convex shapes, and using the centroid criterion can give misleading results. The code in highlight 2 addresses this problem

In highlight 2, we merge census villages to their parent by using the area shared in common between the census village and the constituency. For each census village, the matched constituency is the one which shares the maximum area. This produced satisfactory results, and fed into the next step of my analysis.

.

The full code used for scraping, data cleaning, and analysis is available in my Github repo.

```
-- Highlight 1. Spatial join census villages and Assembly constituencies
-- on centroid of census village

SELECT AddGeometryColumn ('wb_census_2011','census_centroid',4326,'POINT',2);
UPDATE wb_census_2011 SET census_centroid = ST_Centroid(wb_census_2011.geom);

CREATE TABLE village_centroid_ac_map AS
SELECT census.westb_id, census.panch_name, census.sub_dist11,
        census.district11, census.state_ut, census.census2011,
        census.level_11, census.name_11,
        ac_map.wb_id, ac_map.ac_no, ac_map.ac_name
FROM
    wb_census_2011 AS census, wb_ac_map AS ac_map
WHERE ST_Contains(ac_map.geom, census.census_centroid);

\copy (select * from village_centroid_ac_map) TO
'~/01.Map_Extracted_Data/census_village_centroid_ac_match.csv' CSV HEADER

-- Highlight 2. Spatial join census villages and Assembly constituencies
-- on area of overlap of census village
CREATE TABLE village_area_ac_map AS
SELECT DISTINCT ON (census.westb_id)
    census.westb_id, census.panch_name, census.sub_dist11, census.district11,
    ac_map.wb_id, ac_map.ac_no, ac_map.ac_name
FROM
    wb_census_2011 AS census, wb_ac_map AS ac_map
WHERE ST_Intersects(ac_map.geom, census.geom)
ORDER BY census.westb_id, ST_Area(ST_Intersection(ac_map.geom, census.geom)) DESC;

\copy (select * from village_area_ac_map) TO
'~/01.Map_Extracted_Data/census_village_area_ac_match.csv' CSV HEADER
```

Listing 4: Highlights from spatial analysis using PostgreSQL