Bayesian Improved Surname Geocoding

Yogi Patel

12/16/2014

What is Bayesian Improved Surname Geocoding (BISG) ?

Essentially, BISG is a proxy methodology used to estimate an individual's race when that information is not available.

$$Q(i|j,k) = \frac{p(i|j) \times r(k|i)}{u(1,j,k) + u(2,j,k) + u(3,j,k) + u(4,j,k) + u(5,j,k) + u(6,j,k)}$$

$$Where\ u(i,j,k) = p(i|j) \times r(k|i)$$

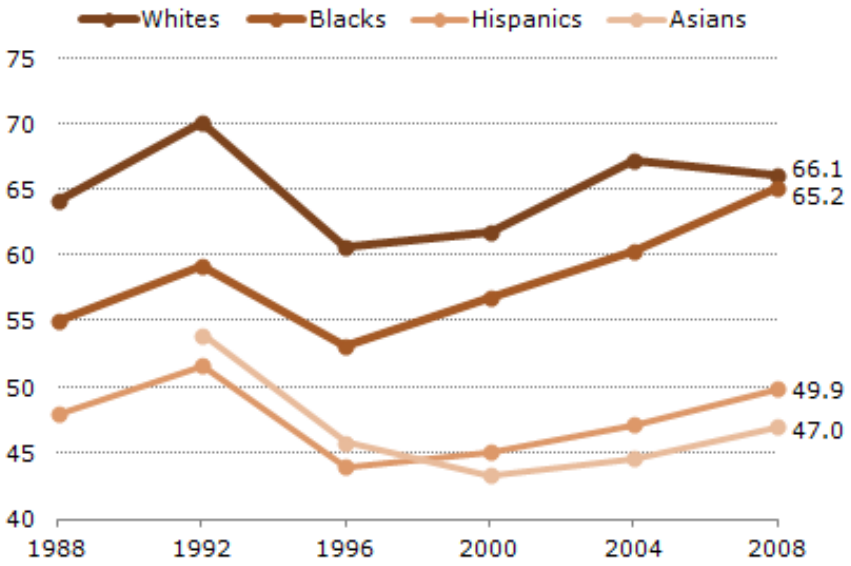
Why is this useful?

- Many policy implications. Can uncover racial discrimination that otherwise is difficult to see or prove.

Figure 2

Voter Turnout Rates in Presidential Elections, 1988-2008

(%)



Notes: Hispanics are of any race. Whites, blacks and Asians include only non-Hispanics. Data for non-Hispanic Asians were not available in 1988.

Source: Pew Hispanic Center tabulations of the Current Population Survey November Supplements

PEW RESEARCH CENTER

What's in the toolbox

- Address Cleaner
- Geocoder via Google Maps API through R
- Geography Data Prep
- Map/Reduce Spatial Join
- Surname Matcher
- BISG Probability Calculator

Address Cleaner

- Regex Pattern Matching

```
#   DEFINING LISTS,VARIBLES, AND REGEXES
arcpy.AddMessage("Compiling RegExes...")
symbls = r"[!@$:#\\^]"
flInfo = re.compile(r"([\d]?[\d]?[\d][\s]?[\s]?((ST)|(ND)|(RD)|(TH))?[\s][\s]?(FL((OOR)|
isPOBox = re.compile(r"P\.?\s?O\.?(\s{1,2})?((BOX)|(BX))")
firstNoDigit = re.compile(r"^\D.*$")
goodZip = re.compile(r"^\d{5}$")
words = (
    "SUITE",
    "STE",
    "APARTMENT",
    "APT",
    "UNIT",
    "FLOOR",
    "FL",
    "BSMT",
    "BLDG",
    "DEPT",
    "FRNT",
    "HNGR",
    "LBBY",
    "LOWR",
    "OFC",
    "PH",
    "TRLR",
    "UPPR",
    "STUDIO",
    "LEVEL"
    )
```

Geocoder via R and ArcGIS Binding

```
###
# Creating funtions to use for geocoding
arc.progress_label("Creating Functions")
# Contructing the geocode URL to use JSON version of google maps geocoding API
construct.geocode.url <- function(address, return.call = "json", sensor = "false") {
  root <- "https://maps.google.com/maps/api/geocode/"
  u <- paste(root, return.call, "?address=", address, "&sensor=", sensor, sep = "")
  return(URLencode(u))
}

# Creating function to send addresses and recieve geocoding results from Google
GeoCode <- function(address,verbose=FALSE) {
  if(verbose) cat(address,"\n")
  # Using function from above to create the URL
  u <- construct.geocode.url(address )
  # Opening URL page and storing results
  doc <- getURL(u, ssl.verifypeer = FALSE)
  # Parsing JSON from page into dataframe
  x <- fromJSON(doc,simplify = FALSE)
  # Getting lat/long
  lat <- x$results[[1]]$geometry$location$lat
  lng <- x$results[[1]]$geometry$location$lng
  # Getting status of result
  status <- x$status
  # Getting Geographic level
  level <- x$results[[1]]$types
  # Getting address matched to
  namefound <- x$results[[1]]$formatted_address
  multipleXY = "N"
  if (length(lng)>1 ){ multipleXY = "Y" }
  # Returning list object to be parsed below
  return(list(status = status, levels=level, lat= lat[1], lng= lng[1], multipleXY= multipleXY,
}
```

Geography Data prep

```
# Converting GEOID's to padded strings in race table
arcpy.AddMessage("Converting Census Race table GEOIDs to padded strings")
fields = arcpy.ListFields(census_race_tbl)
for field in fields:
    if field.name == cenfips:
        if not field.type == "String":
            arcpy.AddField_management(census_race_tbl, "st_fips_string", "TEXT")
            with arcpy.da.UpdateCursor(census_race_tbl, [cenfips, "st_fips_string"]) as cursor_raw:
                for row in cursor_raw:
                    stfips = str(int(row[0]))
                    row[1] = stfips.zfill(2)
                    #   UPDATING INPUT TABLE.
                    cursor_raw.updateRow(row)
            del cursor_raw, row
            cenfips = "st_fips_string"
    elif field.name == cengeoid:
        if not field.type == "String":
            arcpy.AddField_management(census_race_tbl, "geoid_string", "TEXT")
            with arcpy.da.UpdateCursor(census_race_tbl, [cengeoid, "geoid_string"]) as cursor_raw:
                for row in cursor_raw:
                    geoid_num = str(int(row[0]))
                    row[1] = geoid_num.zfill(12)
                    #   UPDATING INPUT TABLE.
                    cursor_raw.updateRow(row)
            del cursor_raw, row
            cengeoid = "geoid_string"
```

“Map/Reduce” Spatial Join

```
# Looping through states in the dataset.
for k,i in enumerate(daStates):
    arcpy.AddMessage("Joining features and merging probabilites in: " + i)
    if i in stateDict:
        fips = stateDict[i]
        arcpy.MakeFeatureLayer_management(dafc, "data_"+i, '"'+dafld+'" = \''+i+'\'')
        arcpy.MakeFeatureLayer_management(bgfc, "bg_"+i, '"'+bgfld+'" = \''+fips+'\'')
        arcpy.MakeTableView_management(census_race_tbl, "census_"+i, '"'+cenfips+'" = \''+fips+'\'' )

        # Performing Spatial Join for the state
        arcpy.SpatialJoin_analysis("data_"+i, "bg_"+i, 'temps_'+i, 'JOIN_ONE_TO_ONE', 'KEEP_ALL','','WITHIN')

        # Converting feature class to table view so we can perform a join
        arcpy.MakeTableView_management('temps_'+i, "temptbl_"+i)

        # Merging in Block Group  probabilites
        arcpy.AddJoin_management("temptbl_"+i, bggeoid, "census_"+i, cengeoid, "KEEP_ALL")

        # Adding feature class and table view to list
        tmp_fcs.append('temps_'+i)
        tmp_tbls.append('temptbl_'+i)

# Reducing temp tables together
if len(daStates) > 1:
    arcpy.AddMessage("Reducing features...")
    arcpy.Merge_management(tmp_tbls, out_fc)
else:
    arcpy.AddMessage("Only one state in dataset, no reduce needed...")
    arcpy.CopyRows_management (tmp_tbls[0], out_fc)
```

Surname Matcher

```
arcpy.AddMessage("Cleaning surnames...")
with arcpy.da.UpdateCursor("tempLayer", [lname_tbl, "Clean_Surname", "Surname2"]) as cursor_raw:
    for row in cursor_raw:
        name = row[0]

        # Making surname upper case
        name = name.upper()
        # Removing numbers are symbols
        name = re.sub(r"[,;\\.0-9]", " ", name)
        # Separately removing apostrophes and replacing with no space to account for Irish folk
        name = re.sub(r"[']", "", name)
        # Removing double quotes
        name = re.sub(r'"', ' ', name)
        # Removing common suffixes
        name = re.sub(r" ((JR)|(SR)|(I)|(II)|(III)|(IV)|(MD)|(DDS)|(PHD)) ", " ", name)
        # Removing any remaining lone letters, most likely initials
        name = re.sub(r" [a-z] ", "", name)
        # Replacing all spaces with non-spaces
        name = re.sub(r" ", "", name)

        # Separating hyphenated last names
        name, sep, lname2 = name.partition("-")
        # Assinging clean names to appropriate columns
        row[1] = name
        row[2] = lname2

        #   UPDATING INPUT TABLE.
        cursor_raw.updateRow(row)
del cursor_raw, row

# This tool will only use the first last name. For added functionality to use the second last name
# as well, follow me on GitHub. Will incorporate in v2.

# Joining the census surname table to the input table, keeping all features in the Input table,
# regardless of match
arcpy.AddMessage("Merging probabilites...")
arcpy.AddJoin_management("tempLayer", "Clean_Surname", census, lname_census, "KEEP_ALL")
```

BISG Calculator

```
# u_race = Pr(race|name) * Pr(this block group | race)
u_white = white_surname * white_here
u_black = black_surname * black_here
u_nativ = nativ_surname * nativ_here
u_asian = asian_surname * asian_here
u_hispanic = hispanic_surname * hispanic_here
u_multi = multi_surname * multi_here

# Calculatin BISG
pr_white = u_white / (u_white + u_black + u_aian + u_api + u_hispanic + u_mult_other)
pr_black = u_black / (u_white + u_black + u_aian + u_api + u_hispanic + u_mult_other)
pr_nativ = u_nativ / (u_white + u_black + u_aian + u_api + u_hispanic + u_mult_other)
pr_asian = u_asian / (u_white + u_black + u_aian + u_api + u_hispanic + u_mult_other)
pr_hispanic = u_hispanic / (u_white + u_black + u_aian + u_api + u_hispanic + u_mult_other)
pr_multi = u_multi / (u_white + u_black + u_aian + u_api + u_hispanic + u_mult_other)
```