ACMT Example: Population density over distance

Weipeng Zhou

2/22/2021

Introduction

Automatic Context Measurement Tool (ACMT) is a convenient tool for studying neighbourhoods in the United States. Based on the user-provided address and radius, ACMT locates a geographical area and outputs context measurements (population, education level, commute time, etc.) for the area. ACMT is easy to install, highly reproducible and works consistently across computer platforms.

There are various ways of using ACMT and here we show one example – 5-city comparison of population density decay as function of distance from City Hall.

We use ACMT to get population densities for 5 cities (Seattle, Los Angeles, Chicago, New York and Boston) over 5 radiuses (1000, 2000, 3000, 4000, 5000). We consider the center of a city to be the location of its City Hall. After getting the densities, we make a plot of density over radius, categoried by cities; and we will be able to tell which city is the most populated across space.

We take the following steps:

1. Find the addresses of each city’s City Hall
2. Use ACMT’s geocoder to convert the addresses to coordinates
3. Use ACMT to get the population measurement for each coordinate over 5 radiuses (1000, 2000, 3000, 4000, 5000)
4. Calculate population density
5. Plot density vs. radius, categoried by cities

Example

1. Find the addresses of each city’s City Hall

We have gathered the City Hall addresses for Seattle, Los Angeles, Chicago, New York and Boston from Google.

```r
source("~/workspace/setup-acmt.R")

library(ggplot2)
city_hall_to_address_list <- list(
  seattle_city_hall="600 4th Ave, Seattle, WA 98104",
  los_angeles_city_hall="200 N Spring St, Los Angeles, CA 90012",
  chicago_city_hall="121 N LaSalle St, Chicago, IL 60602",
  new_york_city_hall="City Hall Park, New York, NY 10007",
  boston_city_hall="1 City Hall Square #500, Boston, MA 02201"
)
```

2. Use ACMT's geocoder to convert the addresses to coordinates

ACMT comes with a handy geocoder that converts addresses to latitude/longtitude coordinates. We check if geocoder is available in the version of ACMT you installed; if it is not available, we use the pre-computed coordinates.

```r
convert_address_to_lat_long <- function (city_hall_to_address_list) { # function to get get lat/long for each address
  city_hall_to_lat_long_list <- vector(mode="list", length=length(city_hall_to_address_list))
  names(city_hall_to_lat_long_list) <- names(city_hall_to_address_list)
  for (name in names(city_hall_to_address_list)){
    city_hall_to_lat_long_list[[name]] <- geocode(city_hall_to_address_list[[name]])
  }
  return(city_hall_to_lat_long_list)
}
```

```r
geocoder_is_available<-as.data.frame(city_hall_to_address_list) %>% t() %>% as.data.fram
e() %>%
  rename(address=V1) %>%
  mutate(city_hall=row.names(.), geocoder_is_available=FALSE)

city_hall_lat_long<-list(seattle_city_hall=list(latitude=47.60328, longitude=-122.3302),
    los_angeles_city_hall=list(latitude=34.05397, longitude=-118.2436),
    chicago_city_hall=list(latitude=41.88334, longitude=-87.63229),
    new_york_city_hall=list(latitude=40.66392, longitude=-73.93835),
    boston_city_hall=list(latitude=42.35773, longitude=-71.05919))

for(address in geocoder_is_available$address){
  tryCatch({geocode(address)
    geocoder_is_available$geocoder_is_available[geocoder_is_available$address==address]=
TRUE}, error=function(condition){
    print(condition$message)
    print("Geocoder not available: using stored address to lat/long mappings instead")
  })
}

# call geocoder if available, use hard coded info otherwise
city_hall_to_lat_long_list <- NULL
for(address in geocoder_is_available$address){
  city_hall_name<-geocoder_is_available$city_hall[geocoder_is_available$address==addres
s]
  if(geocoder_is_available$geocoder_is_available[geocoder_is_available$address==address]
==TRUE){

    city_hall<-list(address)
    names(city_hall)<-city_hall_name
    city_hall_to_lat_long_list<-append(city_hall_to_lat_long_list, convert_address_to_la
t_long(city_hall))
                                                            }
  else(city_hall_to_lat_long_list<-append(city_hall_to_lat_long_list, city_hall_lat_long
[names(city_hall_lat_long)==city_hall_name]))
}
```

```r
print(city_hall_to_lat_long_list[1:2])
```

```
## $seattle_city_hall
## $seattle_city_hall$latitude
## [1] 47.60328
##
## $seattle_city_hall$longitude
## [1] -122.3302
##
## $seattle_city_hall$rating
## [1] 0
##
##
## $los_angeles_city_hall
## $los_angeles_city_hall$latitude
## [1] 34.05397
##
## $los_angeles_city_hall$longitude
## [1] -118.2436
```

3. Use ACMT to get the population measurement for each coordinate over 5 radiuses (1000, 2000, 3000, 4000, 5000)

We create a function for querying ACMT measurements for a list of coordinates and radiuses. We are interested in the variable `total_pop_count`.

```
# function to get the environmental measures for the variables we are intersted
get_variable_measures_from_acmt <- function (city_hall_to_lat_long_list, radius_vector,
year, names_of_variable_to_get, external_data_name_to_info_list=NULL, codes_of_acs_varia
bles_to_get=NA) {
  city_hall_to_radius_to_variable_to_measures_list <- vector(mode="list", length=length
(city_hall_to_lat_long_list))
  names(city_hall_to_radius_to_variable_to_measures_list) <- names(city_hall_to_lat_long
_list)

  for(city_hall in names(city_hall_to_radius_to_variable_to_measures_list)) {
    radius_to_variable_to_measures_list <- vector(mode="list", length=length(radius_vect
or))
    names(radius_to_variable_to_measures_list) <- as.character(radius_vector)
    for (radius in radius_vector) {
      print(city_hall)
      print(radius)
      # get lat/long
      latitude <- city_hall_to_lat_long_list[[city_hall]]$latitude
      longitude <- city_hall_to_lat_long_list[[city_hall]]$longitude

      # get environmental measures for all variables
      environmental_measures <- get_acmt_standard_array(long=longitude, lat=latitude, ra
dius_meters = radius, year=year, external_data_name_to_info_list=external_data_name_to_i
nfo_list, codes_of_acs_variables_to_get=codes_of_acs_variables_to_get)

      # get environmental measures for the variables are interested
      variable_to_measures_list <- vector(mode="list", length=length(names_of_variable_t
o_get))
      names(variable_to_measures_list) <- names_of_variable_to_get
      for (name_of_variable in names_of_variable_to_get) {
        value_of_variable <- environmental_measures[environmental_measures$names == name
_of_variable, ]$values  # 66370.01, seattle, r=2000, year=2017
        variable_to_measures_list[[name_of_variable]] <- value_of_variable
      }
      #radius_to_variable_to_measures_list[[which(radius == radius_vector)]] <- variable
_to_measures_list
      radius_to_variable_to_measures_list[[as.character(radius)]] <- variable_to_measure
s_list
    }
    city_hall_to_radius_to_variable_to_measures_list[[city_hall]] <- radius_to_variable_
to_measures_list
  }
  return(city_hall_to_radius_to_variable_to_measures_list)
}
```

```
setwd('~/workspace')
city_hall_to_lat_long_list <- city_hall_to_lat_long_list
radius_vector <- c(1000, 2000, 3000, 4000, 5000)
year <- 2017
names_of_variable_to_get <- c("total_pop_count")  # the name of the variable in ACMT's r
eturned result
codes_of_acs_variables_to_get <- c("B01001_001")  # speed up; ask ACMT to only query thi
s variable from the ACS server to speed up computation; B01001_001 is population; check
ACSColumns.csv for mappings between codes and variables.

start_time_get_variable_measures_from_acmt <- Sys.time()
city_hall_to_radius_to_variable_to_measures_list <- get_variable_measures_from_acmt(city
_hall_to_lat_long_list=city_hall_to_lat_long_list, radius_vector=radius_vector, year=yea
r, names_of_variable_to_get=names_of_variable_to_get, codes_of_acs_variables_to_get=code
s_of_acs_variables_to_get)
end_time_get_variable_measures_from_acmt <- Sys.time()
```

```
print(city_hall_to_radius_to_variable_to_measures_list[1])
```

```
## $seattle_city_hall
## $seattle_city_hall$`1000`
## $seattle_city_hall$`1000`$total_pop_count
## [1] 20047.59
##
##
## $seattle_city_hall$`2000`
## $seattle_city_hall$`2000`$total_pop_count
## [1] 65734.34
##
##
## $seattle_city_hall$`3000`
## $seattle_city_hall$`3000`$total_pop_count
## [1] 124589.1
##
##
## $seattle_city_hall$`4000`
## $seattle_city_hall$`4000`$total_pop_count
## [1] 171715.4
##
##
## $seattle_city_hall$`5000`
## $seattle_city_hall$`5000`$total_pop_count
## [1] 212559.3
```

```
print("Between start and end of getting ACMT measures: ")
```

```
## [1] "Between start and end of getting ACMT measures: "
```

```
print(end_time_get_variable_measures_from_acmt - start_time_get_variable_measures_from_a
cmt)
```

```
## Time difference of 8.341089 mins
```

4. Calculate population density

We create a function for computing population density for the given radiues.

```
add_density_measures <- function(city_hall_to_radius_to_variable_to_measures_list) {
  city_hall_to_radius_to_variable_to_measures_with_population_density_list <- city_hall_
to_radius_to_variable_to_measures_list  # R's way of making a deep copy
  for (city_hall in names(city_hall_to_radius_to_variable_to_measures_with_population_de
nsity_list)){
    for(radius_character in names(city_hall_to_radius_to_variable_to_measures_with_popul
ation_density_list[[city_hall]])){
      for(variable in names(city_hall_to_radius_to_variable_to_measures_with_population_
density_list[[city_hall]][[radius_character]])) {
        if(variable == "total_pop_count"){  # add other "if" statements if you want to c
ompute other measures
          total_pop_count <- city_hall_to_radius_to_variable_to_measures_with_population
_density_list[[city_hall]][[radius_character]][[variable]]
          radius_numeric <- as.numeric(radius_character)
          population_density <-  total_pop_count/(pi*radius_numeric^2)
          city_hall_to_radius_to_variable_to_measures_with_population_density_list[[city
_hall]][[radius_character]][["population_density"]] <- population_density
        }
      }
    }
  }
  return(city_hall_to_radius_to_variable_to_measures_with_population_density_list)
}
```

```
city_hall_to_radius_to_variable_to_measures_with_population_density_list <- add_density_
measures(city_hall_to_radius_to_variable_to_measures_list)
```

```
print(city_hall_to_radius_to_variable_to_measures_with_population_density_list[1])
```

```
## $seattle_city_hall
## $seattle_city_hall$`1000`
## $seattle_city_hall$`1000`$total_pop_count
## [1] 20047.59
##
## $seattle_city_hall$`1000`$population_density
## [1] 0.006381345
##
##
## $seattle_city_hall$`2000`
## $seattle_city_hall$`2000`$total_pop_count
## [1] 65734.34
##
## $seattle_city_hall$`2000`$population_density
## [1] 0.005230973
##
##
## $seattle_city_hall$`3000`
## $seattle_city_hall$`3000`$total_pop_count
## [1] 124589.1
##
## $seattle_city_hall$`3000`$population_density
## [1] 0.004406438
##
##
## $seattle_city_hall$`4000`
## $seattle_city_hall$`4000`$total_pop_count
## [1] 171715.4
##
## $seattle_city_hall$`4000`$population_density
## [1] 0.003416169
##
##
## $seattle_city_hall$`5000`
## $seattle_city_hall$`5000`$total_pop_count
## [1] 212559.3
##
## $seattle_city_hall$`5000`$population_density
## [1] 0.002706389
```

5. Plot density vs. radius, categoried by cities

We create a function for converting the data we have so far to `ggplot` friendly format. We then use `ggplot` to create our plot.

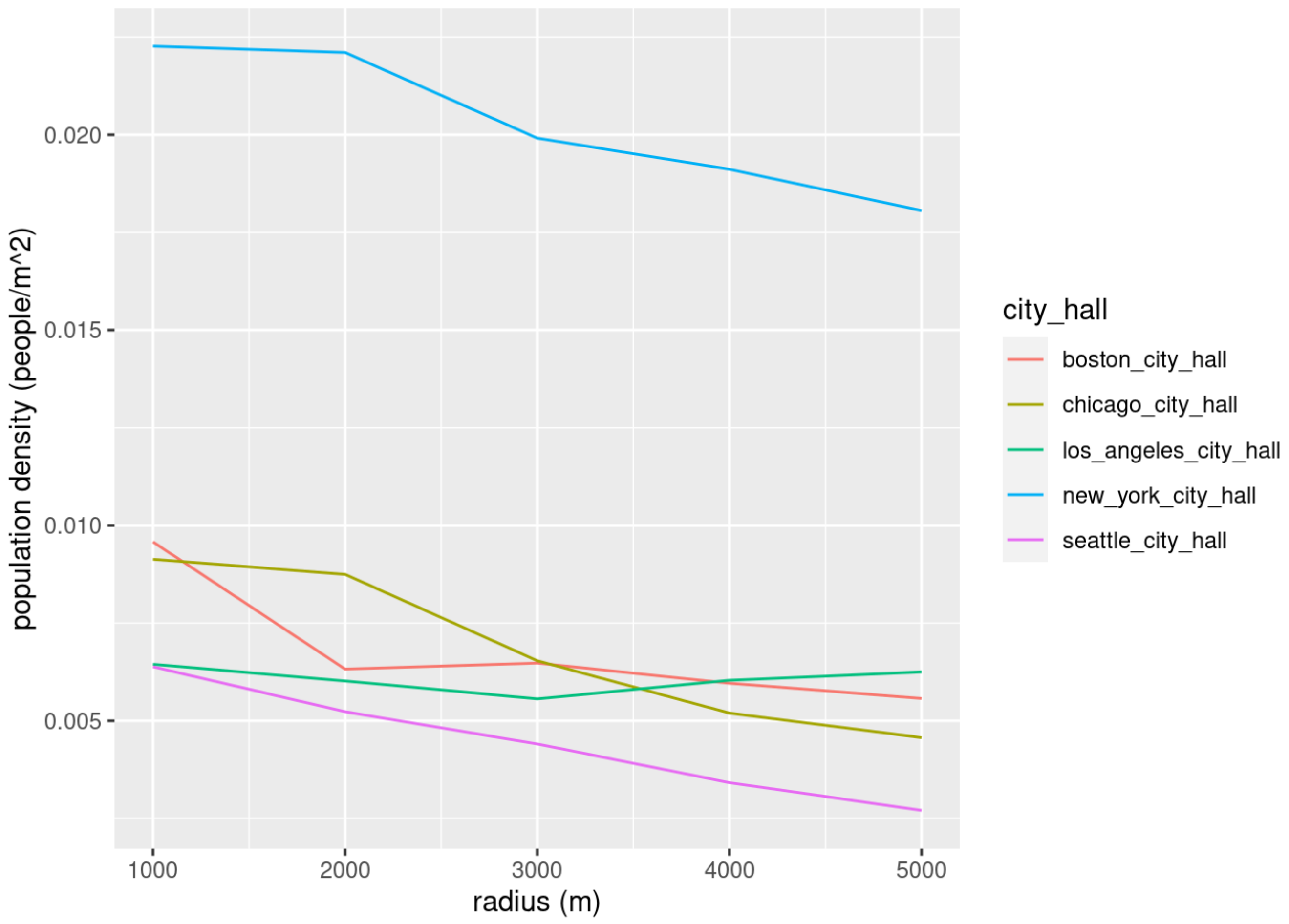
```r
convert_to_dataframe_for_plotting <- function (city_hall_to_radius_to_variable_to_measur
es_with_population_density_list) {
  city_hall_vector <- c()
  radius_vector <- c()
  variable_vector <- c()
  value_vector <- c()

  for(city_hall in names(city_hall_to_radius_to_variable_to_measures_with_population_den
sity_list)){
    for(radius_character in names(city_hall_to_radius_to_variable_to_measures_with_popul
ation_density_list[[city_hall]])){
      for(variable in names(city_hall_to_radius_to_variable_to_measures_with_population_
density_list[[city_hall]][[radius_character]])){
        city_hall_vector <- c(city_hall_vector, city_hall)
        radius_vector <- c(radius_vector, as.numeric(radius_character))
        variable_vector <- c(variable_vector, variable)
        value_vector <- c(value_vector, city_hall_to_radius_to_variable_to_measures_with
_population_density_list[[city_hall]][[radius_character]][[variable]])
      }
    }
  }

  dataframe_for_plotting <- data.frame(city_hall_vector, radius_vector, variable_vector,
value_vector, stringsAsFactors=FALSE)
  names(dataframe_for_plotting) <- c("city_hall", "radius", "variable", "value")
  return(dataframe_for_plotting)
}
```

```r
dataframe_for_plotting <- convert_to_dataframe_for_plotting(city_hall_to_radius_to_varia
ble_to_measures_with_population_density_list)
```

```r
print(head(dataframe_for_plotting))
```

```
##          city_hall radius          variable        value
## 1 seattle_city_hall   1000    total_pop_count 2.004759e+04
## 2 seattle_city_hall   1000 population_density 6.381345e-03
## 3 seattle_city_hall   2000    total_pop_count 6.573434e+04
## 4 seattle_city_hall   2000 population_density 5.230973e-03
## 5 seattle_city_hall   3000    total_pop_count 1.245891e+05
## 6 seattle_city_hall   3000 population_density 4.406438e-03
```

```r
ggplot(dataframe_for_plotting[dataframe_for_plotting$variable == "population_density",
], aes(x=radius)) +
        geom_line(aes(y=value, col=city_hall)) +
        labs(y="population density (people/m^2)",
             x="radius (m)")
```



Results

We see that New York City has a much higher population density, and its density does not reduce much over radiuses.

However, this approach has a critical flaw. The population density is calculated by dividing the total population by a circular area. By doing so, we assume households are eventually distributed within the circular area. Nevertheless, this assumption can be problematic. For example, there is a significant amount of sea area near Seattle City Hall, and households are mostly in the land area. Thus, when calculating population density, the denominator should only be the land part of the circular area.

The problem is, ACMT does not naively provide land area and we need to import external data into ACMT.

In another example, we refine our analysis on population density and demonstrate how external data can be imported into ACMT. We show an example of importing National Walkability Index dataset, which includes land area data, into ACMT. We also include national walkability index in our analysis.