Placing Entries in Geographic Boundaries Using `point.in.polygon()` in R

Alyssa Savo

April 23, 2017

1 The Goal

You have a dataset containing many entries including longitude and latitude indicating where each entry is located. You want to sort the entries into a set of regions (ex. congressional districts, city wards, countries) based on their location data using R.

2 Sorting with the `point.in.polygon()` function

2.1 Load base data and install packages

Load the dataframe containing the entries you want to sort by region. For this example, I'm going to use Washington, D.C.'s campaign financial contributions data set (http://opendata.dc.gov/datasets/campaign-financial-contributions) to sort campaign contributors by the D.C. ward they're located in.

```
contributors <- read.csv("C:/Users/alyss/Documents/Campaign_Financial_Contributions.csv")
```

In addition, install the `sp`, `maptools`, and `rgeos` packages in the console:

```
> install.packages("sp")
> install.packages("maptools")
> install.packages("rgeos")
```

In your console or RMarkdown file, load `sp` and `maptools` using `library()`. You don't need to load `rgeos`, just install it.

2.2 Find and load a KML shape file

Find a KML shape file mapping the geographic regions that you want to match your entries into. Political shape files can be found fairly easily for many US regions such as counties, congressional/legislative districts, and voting precincts on local government GIS websites. Others can be found on university websites, github, etc. with a bit of searching. For this example, I'll be using the map of for Washington, D.C.'s wards from opendata.dc.gov (http://opendata.dc.gov/datasets/ward-from-2012).

Before loading your KML file in R, I recommend previewing it on the website or using a GIS service like Google Fusion tables to check if the geographic entries in numerical or alphabetical order. It helps if the entries are in order to make it easier to label each of the regions later.

Load the KML file into R as a list object using the `getKMLcoordinates()` function, with the argument `ignoreAltitude` set to `TRUE`.

```
ward.shapes <- getKMLcoordinates(kmlfile="C:/Users/alyss/Documents/Ward_from_2012.kml",
  ignoreAltitude=T)
```

2.3 Create polygons

Use the `Polygon()` function to create shapes from each region in the KML file. The D.C. ward data doesn't list the wards in numeric order, so the entry being specified in double brackets doesn't match to the ward number. If your KML data is sorted numerically or alphabetically, you'll have a simpler time generating the regions in order. If you're dealing with a large number of regions you also may want to create a `for` loop instead of using separate lines for each region.

```
ward.1 <- Polygon(ward.shapes[[5]])

ward.2 <- Polygon(ward.shapes[[4]])

ward.3 <- Polygon(ward.shapes[[7]])

ward.4 <- Polygon(ward.shapes[[8]])

ward.5 <- Polygon(ward.shapes[[6]])

ward.6 <- Polygon(ward.shapes[[2]])

ward.7 <- Polygon(ward.shapes[[3]])

ward.8 <- Polygon(ward.shapes[[1]])
```

2.4 Create indicator variables for each region

For each of the regions, create a variable in the dataframe to indicate whether or not each entry is located in the region. You'll be using the `point.in.polygon()` function to generate a list of 1s and 0s for each variable, where 1 indicates that the entry was located in the region polygon and 0 indicates that it was not. Again, if you're dealing with a large number of regions you may want to create a `for` loop here.

In the `point.in.polygon()` function, you'll be plugging in four arguments: the longitude variable for your dataframe, the latitude variable for your dataframe, the longitude column for the region polygon, and the latitude column for the region polygon. The `polygon()` function generates `s4` objects, which will use @ instead of $ to specify `coords` for the polygon.

```
contributors$contributor.in.ward1 <- point.in.polygon(contributors$LONGITUDE,
contributors$LATITUDE, ward.1@coords[,1], ward.1@coords[,2])

contributors$contributor.in.ward2 <- point.in.polygon(contributors$LONGITUDE,
contributors$LATITUDE, ward.2@coords[,1], ward.2@coords[,2])

contributors$contributor.in.ward3 <- point.in.polygon(contributors$LONGITUDE,
contributors$LATITUDE, ward.3@coords[,1], ward.3@coords[,2])

contributors$contributor.in.ward4 <- point.in.polygon(contributors$LONGITUDE,
contributors$LATITUDE, ward.4@coords[,1], ward.4@coords[,2])

contributors$contributor.in.ward5 <- point.in.polygon(contributors$LONGITUDE,
contributors$LATITUDE, ward.p5@coords[,1], ward.5@coords[,2])

contributors$contributor.in.ward6 <- point.in.polygon(contributors$LONGITUDE,
contributors$LATITUDE, ward.6@coords[,1], ward.6@coords[,2])
```

```
contributors$contributor.in.ward7 <- point.in.polygon(contributors$LONGITUDE,
contributors$LATITUDE, ward.7@coords[,1], ward.7@coords[,2])

contributors$contributor.in.ward8 <- point.in.polygon(contributors$LONGITUDE,
contributors$LATITUDE, ward.8@coords[,1], ward.8@coords[,2])
```

2.5 Create a single region variable

Create a single variable based on the indicators you created that directly states the region each entry is located in.

```
contributors$WARD <- NA

contributors$WARD[contributors$contributor.in.ward1 == 1] <- "Ward 1"

contributors$WARD[contributors$contributor.in.ward2 == 1] <- "Ward 2"

contributors$WARD[contributors$contributor.in.ward3 == 1] <- "Ward 3"

contributors$WARD[contributors$contributor.in.ward4 == 1] <- "Ward 4"

contributors$WARD[contributors$contributor.in.ward5 == 1] <- "Ward 5"

contributors$WARD[contributors$contributor.in.ward6 == 1] <- "Ward 6"

contributors$WARD[contributors$contributor.in.ward7 == 1] <- "Ward 7"

contributors$WARD[contributors$contributor.in.ward8 == 1] <- "Ward 8"
```

It's also a good idea to assign a value for entries that weren't located in any region, if there are any left in the dataset:

```
contributors$WARD[is.na(contributors$WARD)] <- "Not in D.C."
```

Although optional, I recommend saving the modified dataframe to a new .csv file using the `write.csv()` function so that you don't have to have all of this code written out every time you want to analyze your dataset:

```
write.csv(contributors, "C:/Users/alyss/Documents/Campaign_Contributions_mapped.csv")
```

That aside, you're done!