Rvoterdistance: Calculating Distances Between Voters and Multiple Polling Locations

by Loren Collingwood

Abstract Recent developments in how and when voters cast their ballots have led analysts and scholars to inquire whether these developments hinder or help voter turnout. For example, on the one hand, in the United States, voters in the state of Washington cast their ballots via mail, although increasingly voters are depositing their ballots into drop boxes as opposed to mailing them in. On the other hand, in 2016, states like North Carolina reduced the number of early voting hours available at polling locations. **Rvoterdistance** is a new package that helps scholars and analysts assess whether these changes influence turnout by calculating the distance between each individual voter and their closest drop box, early voting location, or other such similar voting location. Users can then perform standard statistical operations (e.g. difference in difference; conditional logit), to assess changes in voting propensity pre/post the new implementation.

Introduction

Recent developments in how and when voters cast their ballots have led analysts and scholars to inquire whether these developments hinder or help voter turnout. On the one hand, in the United States, voters in the state of Washington cast their ballots via mail, although increasingly voters are depositing their ballots into drop boxes as opposed to mailing ballots. King County, Washington – the home of Seattle – constructed 33 new drop boxes around the county in 2016, leading analysts to inquire whether such boxes tend to improve voter turnout.

On the other hand, since *Shelby County v. Holder* (2013), which struck down federal pre-clearance, states like North Carolina are now free to make changes to their voting laws without clearing such changes by the federal government. As such, in 2016, North Carolina reduced the hours of operation at early voting locations. In both Washington and North Carolina, it remains an open question whether these changes influenced voter turnout.

Given a set of latitude and longitude coordinates – which can be easily extracted with the `ggmap` package (Kahle and Wickham, 2013) , **Rvoterdistance** helps scholars and analysts assess whether changes to voting laws influence voter turnout by calculating the distance between each individual voter and their closest drop box, early voting location, or other such polling location. Users simply supply geospatial latitude and longitude coordinates for each voter and for each polling location. This distance, then, becomes a proxy for the average treatment effect of exposing voters to new ways of voting or taking away polling locations that were previously there, for example. Users can then perform standard statistical operations (e.g. difference in difference; conditional logit), to assess changes in vote propensity for each voter.

While users can fairly easily craft this distance calculation with a few loops in R (R Core Team, 2015) or perhaps with other packages, **Rvoterdistance** relies on **Rcpp** (Eddelbuettel et al., 2011; Eddelbuettel, 2013) to vastly speed up the process. This is crucial to workflow for users operating on large datasets such as voter files, which often contain upwards of five million records, along with hundreds or thousands of potential voting locations. This article reviews how to use the new package, then demonstrates speed improvements, using random samples of data from both Washington State and North Carolina.

1. Installation and Data

To begin, install (`install.packages(''Rvoterdistance'')`) and load the **Rvoterdistance** package (`library(Rvoterdistance)`) from the CRAN repository. The package comes with two `.RData` files, each with a dataset of voter latitude and longitude coordinates, and a dataset of drop box/polling location geospatial coordinates. First, the Washington data can be accessed:

```
# LOAD DATA, King County, WA
data(king_dbox)
# LOOK AT VOTER FILE COORDINATES
str(king_geo)
```

```
# LOOK AT DROP BOX LOCATIONS
str(dbox)
```

Above, the first dataset consists of 5,000 voters in King County, WA, where the two columns are longitude and latitude of class numeric. The second dataset is a collection of all drop box locations in King County, where lat/long are the key variables. Second, the North Carolina data can be accessed:

```
# LOAD DATA, Mecklenburg County, NC
data(meck_ev)
str(voter_meck)
str(early_meck)
```

Above, the first dataset consists of 4,552 voters in Mecklenburg County, NC (Charlotte), where long/lat are the key variables we will use for the distance calculation. The second dataset includes 21 early polling vote locations spread around the county, also including long/lat variables.

2. Calculate Nearest Drop Box and Check for NAs

The main package function is `nearest_dbox()`, which calculates the nearest polling location (drop box in the case of the King County data) for each voter. Behind the scenes, this function ports to C++, which uses a double loop to calculate each voter's distance to each polling location, then returns the minimum distance. The result is a vector of haversine distances measured in meters, one for each voter (Robusto, 1957; Shumaker and Sinnott, 1984).

`nearest_dbox()` takes four arguments, `lat1d_vec`, `lon1d_vec`, `lat2d_vec`, `lon2d_vec`, where the first two are numeric vectors of voter coordinates, and the latter two vectors of polling location coordinates. These arguments must follow the correct order: latitude 1, longitude 1, latitude 2, longitude 2. Often, the first two vectors will be attached to the same dataset, and the same for the latter two vectors. It is important that the user omit any missing data before submitting the vectors to `nearest_dbox()`.

```
# Create vectors of lat/longs, and check
king_lat <- king_geo$Residence_Addresses_Latitude
king_long <- king_geo$Residence_Addresses_Longitude
dbox_lat <- dbox$lat
dbox_long <- dbox$long

# Check for NAs
isNAcheck <- function(vec) table(is.na(vec))
sapply(list(king_lat, king_long, dbox_lat, dbox_long), isNAcheck)

# Execute Calculation
king_nearest <- nearest_dbox (king_lat, king_long,dbox_lat, dbox_long)
summary(king_nearest)
```

The result is a vector of haversine distances, of `length(lat1d_vec)`. Table 1 reports the output's summary statistics for the King County example.

	Min.	1st Qu.	Median	Mean	3rd Qu.	Max.
Summary king_nearest	37.48	1991.00	3636.00	3857.00	5607.00	53160.00

Table 1: Summary statistics of nearest_dbox() output

3. Convert Distance to Kilometers or Miles

When reporting results, many laypersons will be unfamiliar with haversines. Therefore, `Rvoterdistance` lets users calculate the distances in kilometers and/or miles. The functions `dist_km()` and `dist_mile()` take identical arguments, either the same four vectors as `nearest_dbox()` or the results of `nearest_dbox()`. With the latter, the two arguments to use are now `num_vec` – which takes the output from `nearest_dbox()` – and `vec_only` is set to `TRUE`. The two ways are shown below:

```
# Distance in Kilometers
summary (dk1 <- dist_km(num_vec=king_nearest, vec_only = TRUE))
```

```
# Distance in Miles
summary (dm1 <-  dist_mile(num_vec=king_nearest, vec_only = TRUE) )
summary (dm2 <-  dist_mile(king_lat, king_long,dbox_lat, dbox_long ))
```

Table 2 summarizes the distributions of the two `dist_mile()` calculations, revealing that they are identical. Thus, users can employ whichever approach they prefer.

	Min.	1st Qu.	Median	Mean	3rd Qu.	Max.
Summary dm1	0.02	1.24	2.26	2.40	3.48	33.03
Summary dm2	0.02	1.24	2.26	2.40	3.48	33.03

Table 2: Summary statistics of two mile calculations

Finally, a few robustness checks are necessary to ensure our confidence in the correct calculations. The code below shows that both versions of mile calculation produce the same result, and that mile conversion correlates at one with the haversine output, which is what we would expect. We can now add on the haversine and mile distance calculations to the original voter file with a simple call to `data.frame()`.

```
# Conduct a few checks
all.equal(dm1, dm2) # Two methods equal
cor(dm1, king_nearest) # Correlation

# Merge/Append back to original dataset
king_final <- data.frame(king_geo, king_nearest, dm1)
str(king_final)
```

4. Including Polling Location Data

Many users will simply want a distance measure to incorporate into their analysis. However, in addition to returning a single distance measure, **Rvoterdistance** will also return all information contained in the polling location dataset appended onto the individual voter dataset. The function `smorgesboard()` does just this, by taking arguments `data1`, `data2`, `lat_long1_char`, `lat_long2_char`, and returning a dataset of length `nrow(data1)`. The first two arguments are data frames containing the requisite lat/long coordinates and any other individual level (data1) or polling location (data2) information contained in the original datasets. The latter two arguments are character vectors of length 2 specifying the column names of the lat/long coordinates in the respective datasets.

In the code below, `smorgesboard()` takes the Mecklenburg County voter file data, followed by the dataset of early voting polling locations. The first column of `data2` is deleted in the example because the information is already contained in `data1`. The next two arguments specify the respective column names.

```
vote_distance <- smorgesboard(voter_meck, early_meck[,-1], c("lat", "long"),
        c("lat", "long"))
head(vote_distance)
```

Table 3 reveals the first few lines of the example's output, which is a combination of individual-level voter information (i.e., the voter file) combined with polling location information, and the three distance measures. The first three columns (county, long, lat) are the voter's home county, and the voter's geographic longitude and latitude coordinates. The next three columns (office_addr, long.1, and lat.1) specify the name and address of the closest early vote location, in addition to that location's geospatial coordinates. Finally, the distance calculation between that polling location and the voter are returned (haversines, miles, kilometers).

	county	long	lat	office_addr	long.1	lat.1	distance_haversine	distance_mile	distance_km
1	MECKLENBURG	-80.93	35.21	WEST BOULEVARD LIBRARY 2157 WEST BLVD	-80.90	35.21	2950.37	1.83	2.95
2	MECKLENBURG	-81.00	35.11	STEELE CREEK AREA 11130 SOUTH TRYON ST	-80.96	35.12	3517.65	2.19	3.52
3	MECKLENBURG	-80.81	35.22	MIDWOOD CULTURAL CENTER 1817 CENTRAL AVE	-80.81	35.22	574.03	0.36	0.57
4	MECKLENBURG	-80.79	35.26	SUGAR CREEK LIBRARY 4045 N TRYON ST	-80.80	35.26	676.05	0.42	0.68
5	MECKLENBURG	-80.87	35.04	SOUTH COUNTY REGIONAL LIBRARY 5801 REA RD	-80.81	35.09	7223.92	4.49	7.22
6	MECKLENBURG	-80.78	35.47	CORNELIUS TOWN HALL 21445 CATAWBA AVE	-80.86	35.48	7184.10	4.46	7.18

Table 3: Smorgesboard() output appends useful polling location information onto the voter file

5. Speed Improvements

The main benefit of **Rvoterdistance** is the speed at which it calculates geospatial distances. Using the Mecklenburg County, NC, data above, here I show just how much faster the `nearest_dbox()` calculation is compared to a comparable operation in base R, using the **geosphere** package (Hijmans, 2016). The first line of code below employs the `nearest_dbox()` function to calculate the distance between each voter and their closest early voting location. The next block of code conducts the same calculation but with more traditional R means. Finally, the last line of code demonstrates that the methods produce identical distance values.

```
# Speed Improvements of Rvoterdistance
system.time( hav_meck <- nearest_dbox (voter_meck$lat, voter_meck$long,
                          early_meck$lat, early_meck$long) )

#install.packages("geosphere")
library(geosphere)
ev_dist_close_g<- rep(NA, nrow(voter_meck)) # Create vector of NAs to fill in
# Initiate Loop across Voters
system.time(
for (i in 1:nrow(voter_meck)) {
  voter_loc <- voter_meck[i,c("long", "lat")]
  ev_fill <- rep(NA, nrow(early_meck)) # collect distance between voter and each dropbox, for each voter
  # Initiate Loop for Boxes
  for (j in 1:nrow(early_meck)) {
    ev_fill[j] <- distHaversine(voter_loc, early_meck[j, c("long", "lat")]) # Haversine Distance
  }
  ev_dist_close_g[i] <- ev_fill[which.min(ev_fill)] # closest box
}
)
# Testing to see that all is same
all.equal(hav_meck, ev_dist_close_g)
```

Table 4 reports the output of the two calculation operations. The total time for `nearest_dbox()` on a Mac OSX platform, x86_64-apple-darwin13.4.0, is 0.013 seconds. The R double-loop records 52.5 seconds – essentially 4,000 times slower. While the double-loop is manageable for small datasets – in this case just over 4,500 voters – the benefits are obvious as the size of the voter file increases.

	user	system	elapsed
nearest_dbox	0.01	0.00	0.01
Base R	51.88	0.29	52.52

Table 4: Computational speed comparison between two distance calculations

6. Summary and Conclusion

Rvoterdistance was designed to expedite the distance calculations of individual voters to multiple polling locations. Given the myriad changes occurring in the way voters cast their ballots, it is important for scholars and analysts to know whether these changes influence individuals' propensity to vote. But, because voter files often contain millions or records, and polling locations can reach into the hundreds or thousands, base R procedures take a long time to process such calculations, greatly slowing down workflow. **Rvoterdistance** greatly expedites this process allowing analysts to focus their efforts on whether such procedures hinder or enhance voter turnout.

Bibliography

D. Eddelbuettel. *Seamless R and C++ integration with Rcpp*. Springer, 2013. [p1]

D. Eddelbuettel, R. François, J. Allaire, J. Chambers, D. Bates, and K. Ushey. Rcpp: Seamless r and c++ integration. *Journal of Statistical Software*, 40(8):1–18, 2011. [p1]

R. J. Hijmans. *geosphere: Spherical Trigonometry*, 2016. URL https://CRAN.R-project.org/package=geosphere. R package version 1.5-5. [p4]

D. Kahle and H. Wickham. ggmap: Spatial visualization with ggplot2. *The R Journal*, 5(1):144–161, 2013. URL http://journal.r-project.org/archive/2013-1/kahle-wickham.pdf. [p1]

R Core Team. *R: A Language and Environment for Statistical Computing*. R Foundation for Statistical Computing, Vienna, Austria, 2015. URL https://www.R-project.org/. [p1]

C. C. Robusto. The cosine-haversine formula. *The American Mathematical Monthly*, 64(1):38–40, 1957. [p2]

B. Shumaker and R. Sinnott. Astronomical computing: 1. computing under the open sky. 2. virtues of the haversine. *Sky and telescope*, 68:158–159, 1984. [p2]

Loren Collingwood
Department of Political Science
University of California, Riverside
900 University Avenue
Riverside, CA 92521
USA
loren.collingwood@ucr.edu