Code

Trusting No One, Getting Nowhere: Soft Policy and the Janus-Faced Nature of Social Capital in Evacuation Networks

Facebook Aggregation Code

Timothy Fraser

February 19, 2021

First, this study uses data from Facebook Data for Good, which tallies up the number of Facebook users who were identified to be in the disaster hit area consistently two weeks prior to the disaster. Then, they tally up how many people went from any neighborhood in that region to any other neighborhood. Unfortunately, this data is measured at the neighborhood level, using special polgyons. This is not a level joinable with census data, and by data sharing agreement, the researchers cannot share the original raw data without permission from Facebook. So, to remedy both these issues, we're going to aggregate everything up from the neighborhood level to the census county subdivision level. (After that, we can easily aggregate up to the county level). Researchers can consult this code to see *how* I aggregated data up from the neighborhood level. Those interested in replicating this exact step should contact Facebook's Data for Good program for permission to access the raw data. Otherwise, all replication of models using the aggregated county subdivision network is described in the companion file ("replication_code.Rmd").

0. Packages

First, let's load some packages!

Hide

```
library(tidyverse) # for tidy data manipulation
# Using the most recent version of dplyr, we can bind_rows() for sf objects
# Download the most recent version here:
# library(remotes)
#remotes::install_github("tidyverse/dplyr")
library(dplyr)
library(dtplyr) # for dplyr functions with data.table, the speedy data manipulation pack
age

# GIS packages
library(sf) # for tidy spatial data
library(rgdal) # for spatial operations
library(tigris) # for accessing census boundaries
library(tidycensus) # for downloading census data
library(censusapi) # for downloading census data
# Networks Packages
library(tidygraph)
library(ggraph)
#remotes::install_github("luukvdmeer/sfnetworks")
library(sfnetworks)
```

1. Geographic Data

1.1 Explore Points

First, let's investigate which states and counties saw evacuation from Hurricane Dorian in Florida.

Hide

```
data.frame(file = dir("fb_data/dorian", full.names = TRUE)) %>%
  split(.$file) %>%
  map_dfr(~read_csv(.$file), .id = "file") %>%
  # Extract date and time from file name
    mutate(file = file %>% str_remove("fb_data/dorian/") %>% str_remove(".csv")) %>%
  mutate(date_time = lubridate::make_datetime(
    year = str_sub(file, 1,4) %>% as.numeric(),
    month = str_sub(file, 5,6) %>% as.numeric(),
    day = str_sub(file, 7,8) %>% as.numeric(),
    hour = str_sub(file, 10,11) %>% as.numeric())) %>%
  # Give better names
  magrittr::set_colnames(value = names(.) %>% tolower() %>% str_replace_all(" |[(]|[)]",
"_")) %>%
   magrittr::set_colnames(value = names(.) %>% str_replace_all(" |[(]|[)]|[:]", "_") %>%
                            str_replace_all("__", "_") %>%
                            str_replace_all("_km_", "_km")) %>%
  # Grab selection of variables at the end
  select(date_time,start_name = starting_region_name, start_polygon_id = starting_locati
on,
    end_name = ending_region_name, end_polygon_id = ending_location,
    n_crisis = crisis_people_moving, n_baseline = baseline_people_moving, length_km, geo
metry) %>%
    # Split geometry into latitude and longitude
    mutate(geometry = geometry %>%
           str_remove(pattern = "LINESTRING [(]") %>%
           str_remove("[)]") %>% str_remove("[,]")) %>%
  separate(col = geometry, into = c("start_long", "start_lat", "end_long", "end_lat"), s
ep = " ", convert = TRUE) %>%
    saveRDS("fb_data/dorian.rds")
```

Hide

```
# Compile Facebook data
# First, let's grab all the names of the files from the disaster,
read_rds("fb_data/dorian.rds") %>%
# now save!
  # Give each edge a unique ID
  mutate(id = 1:n()) %>%
  # Keep only complete observations
  filter(!is.na(n_baseline) & !is.na(n_crisis)) %>%
  # Extract longitude and latitude from geometry
  rename(
    from_x = start_long,
    from_y = start_lat,
    to_x = end_long,
    to_y = end_lat) %>%
  # Convert to numeric
  mutate_at(vars(from_x, from_y, to_x, to_y), as.numeric) %>%
 # Filter out any geocodes that didn't make it
  filter(!is.na(from_x) & !is.na(from_y) & !is.na(to_x) & !is.na(to_y)) %>%
  # Create line-springs in C really really fast
  mutate(geometry = sprintf("LINESTRING(%s %s, %s %s)",
                            from_x, from_y,
                            to_x, to_y)) %>%
  # Set the crs and tell the object to become sf format using the geometry column
  # For the coordinate reference system, let's use:
  # the World Geodetic System from 1984, commonly used with degrees
  # WGS 84 https://spatialreference.org/ref/epsg/wgs-84/
  st_as_sf(crs = CRS(paste0("+proj=longlat +ellps=WGS84 +datum=WGS84 +no_defs")),
           wkt = "geometry") %>%
  saveRDS("fb_data/dorian_edges_neighborhood.rds")
```

1.2 Get Polygons

Next, let’s use the excellent tigris package to get polygons for every level of geography we’re looking at for Hurricane Dorian.

Hide

```
# Now, let's import the states where our study captured evacuation points
tigris::states(cb = TRUE, year = 2019)  %>%
  select(state = STUSPS, name = NAME, geometry) %>%
  # Zoom into just Arkansas, Louisiana, and Mississippi
  filter(state %in% c("FL", "GA", "AL")) %>%
  st_as_sf() %>%
  # Convert them to the WSG84 coordinate references system
  st_transform(crs = CRS(paste0("+proj=longlat +ellps=WGS84 +datum=WGS84 +no_defs"))) %
>%
  saveRDS("shapes/states.rds")


# Next, let's identify the counties where evacuation occurred
counties <- tigris::counties(cb = TRUE, year = 2019, state = c("FL", "GA", "AL")) %>%
  select(geoid = GEOID, name = NAME, geometry) %>%
  st_as_sf() %>%
  # Convert them to the WSG84 coordinate references system
  st_transform(crs = CRS(paste0("+proj=longlat +ellps=WGS84 +datum=WGS84 +no_defs")))
  # Now convert to the Equal Area Conic projection,

# Get Albers Equal Area Conic Projection
#https://spatialreference.org/ref/esri/north-america-albers-equal-area-conic/
aea <- "+proj=aea +lat_1=20 +lat_2=60 +lat_0=40 +lon_0=-96 +x_0=0 +y_0=0 +ellps=GRS80 +d
atum=NAD83 +units=m +no_defs"

# Now, let's extract all the counties in Florida...
fl <- counties %>%
  filter(str_sub(geoid, 1,2) == "12") %>%
  as.data.frame() %>%
  select(geoid) %>%
  unlist()
# And let's extract only the counties with evacuation in alabama or Georgia
alga <- counties %>%
  filter(str_sub(geoid, 1,2) != "12") %>%
  st_transform(crs = aea) %>%
  st_join(read_rds("fb_data/dorian_edges_neighborhood.rds") %>%
            st_transform(crs = aea)) %>%
  filter(!is.na(start_name)) %>%
  as.data.frame() %>%
  select(geoid) %>%
  unlist() %>%
  unique()

mycounties <- c(fl, alga)

# Please save this vector of Dorian affected counties
counties %>%
  filter(geoid %in% mycounties) %>%
  saveRDS("shapes/counties.rds")

# While we are at it, let's get the census tracts.

# For each state of interest,
```

```
c("FL", "GA", "AL") %>%
  # Gather census tracts
  map(~tigris::tracts(state = ., cb = TRUE, year = 2019) %>%
        st_as_sf() %>%
        # Convert them to the WSG84 coordinate references system
        st_transform(crs = CRS(paste0("+proj=longlat +ellps=WGS84 +datum=WGS84 +no_defs"
)))) %>%
  # Use the new sf-friendly dplyr's function to bind these polygons into one data.frame
  dplyr::bind_rows() %>%
  # Rename columns
  select(geoid = GEOID, area_land = ALAND, geometry) %>%
  # Now filter to just census tracts in our study area!
  filter(str_sub(geoid, 1,5) %in% mycounties) %>%
  # Save to file
  saveRDS("shapes/tracts.rds")

# While we are at it, let's get county subdivisions.


# For each state of interest,
csub <- c("FL", "GA", "AL") %>%
  # Gather census tracts
  map(~tigris::county_subdivisions(state = ., cb = TRUE, year = 2019) %>%
        st_as_sf() %>%
        # Convert them to the WSG84 coordinate references system
        st_transform(crs = CRS(paste0("+proj=longlat +ellps=WGS84 +datum=WGS84 +no_defs"
)))) %>%
  # Use the new sf-friendly dplyr's function to bind these polygons into one data.frame
  dplyr::bind_rows() %>%
  # Rename columns
  select(geoid = GEOID, name = NAME, area_land = ALAND, geometry) %>%
  # Now filter to just subdivisions in our study area!
  filter(str_sub(geoid, 1,5) %in% read_rds("shapes/counties.rds")$geoid) %>%
  # Save to file
  saveRDS("shapes/csub.rds")



# Import counties
counties <- read_rds("shapes/counties.rds")
# Import state boundaries, using counties
states <- read_rds("shapes/counties.rds") %>%
  group_by(state = str_sub(geoid, 1,2) %>%
             recode_factor("12" = "Florida",
                           "13" = "Georgia",
                           "01" = "Alabama")) %>%
  summarize(geometry = st_union(geometry))

# Import edges
dat <- read_rds("fb_data/dorian_edges_neighborhood.rds")


# Finally, let's visualize them
ggplot() +
```

```
  # With a darkgrey border for the states, and a white fill overtop
  geom_sf(data = states, fill = NA, color = "darkgrey", size = 5) +
  geom_sf(data = states, fill = "white", color = NA) +
  # And a nice layer of just the counties in our study
  geom_sf(data = counties, fill = "grey", color = "black", size = 0.25) +
  # Some nice state boundaries
  geom_sf(data = states, fill = NA, color = "black", size = 1.2) +
  # With edges highlights using red
  geom_sf(data = dat, color = "firebrick", alpha = 0.2) +
  # And state labels overtop
  geom_sf_label(data = states, mapping = aes(label = state)) +
  # Plus some formatting
  theme_void(base_size = 14) +
  theme(plot.subtitle = element_text(hjust = 0.5)) +
  labs(subtitle = "Evacuation among Neighborhoods\nduring Hurricane Dorian")  +
  ggsave("viz/edges_neighborhoods.png", dpi = 500, width = 3, height = 4)


remove(states, counties, mycounties, dat)




# Import edges
dat <- read_rds("fb_data/dorian_edges_neighborhood.rds")
# Import counties
counties <- read_rds("shapes/counties.rds")
# Get a solid background in the shape of all the counties
back <- counties %>%
  summarize(geometry = st_union(geometry))
# Import tracts
sub <- read_rds("shapes/csub.rds")

# Finally, let's visualize them
ggplot() +
  # With a darkgrey border for the states, and a white fill overtop
  geom_sf(data = back, fill = NA, color = "darkgrey", size = 5) +
  geom_sf(data = back, fill = "white", color = NA) +
  # Specify state by background color
  geom_sf(data = counties, mapping = aes(fill = str_sub(geoid, 1,2) %>%
                                           recode_factor("12" = "Florida",
                                                         "13" = "Georgia",
                                                         "01" = "Alabama")),
          color = NA, alpha = 0.5) +
  scale_fill_manual(values = c("#648FFF", "#DC267F", "#FFB000")) +
  # And a nice layer of just the tracts in our study
  geom_sf(data = sub, fill = NA, color = "darkgrey", size = 0.25) +
  # Some nice county boundaries
  geom_sf(data = counties, fill = NA, color = "black", size = 0.50) +
  # With edges highlights using red
  geom_sf(data = dat, color = "white", alpha = 0.2, size = 0.1) +
  # Plus some formatting
  theme_void(base_size = 14) +
  theme(plot.subtitle = element_text(hjust = 0.5),
```

```
        legend.position = c(0.3,0.4)) +
  labs(subtitle = "Evacuation among Neighborhoods\nduring Hurricane Dorian",
       fill = "State")  +
  ggsave("viz/fig_B4_edges_neighborhoods_csub.png", dpi = 500, width = 5, height = 5)

remove(back,counties,tracts,dat)
```

Last, let's also download the zipcodes located in our counties of interest.

Hide

```
tigris::zctas(cb = TRUE, year = 2019) %>%
  st_as_sf() %>%
    select(zipcode = GEOID10, area_land = ALAND10, geometry) %>%
  # Transform back to WSG84
  st_transform(CRS(paste0("+proj=longlat +ellps=WGS84 +datum=WGS84 +no_defs"))) %>%
  # Join in our counties of interest
  st_join(read_rds("shapes/counties.rds")) %>%
  # Filter to just the ones which joined successfully (our study region)
  filter(!is.na(name)) %>%
  select(-name, -geoid) %>%
  # and save!
  saveRDS("shapes/zipcodes.rds")
```

1.3 Get Centroids

Now, let's give each zipcode, tract, and county a centroid.

Hide

```
# For census tracts
read_rds("shapes/tracts.rds") %>%
  # Transform to Equal Area Conic Projection, for best centroid calculation
  st_transform(crs = CRS(paste0(aea))) %>%
  # Calculate centroids for every county
  group_by(geoid) %>%
  summarize(geometry = st_centroid(geometry)) %>%
  ungroup() %>%
  # Transform back to WSG84
  st_transform(CRS(paste0("+proj=longlat +ellps=WGS84 +datum=WGS84 +no_defs"))) %>%
  saveRDS("shapes/tracts_centroid.rds")


# For county subdivisions
read_rds("shapes/csub.rds") %>%
  # Transform to Equal Area Conic Projection, for best centroid calculation
  st_transform(crs = CRS(paste0(aea))) %>%
  # Calculate centroids for every county
  group_by(geoid) %>%
  summarize(geometry = st_centroid(geometry)) %>%
  ungroup() %>%
  # Transform back to WSG84
  st_transform(CRS(paste0("+proj=longlat +ellps=WGS84 +datum=WGS84 +no_defs"))) %>%
  saveRDS("shapes/csub_centroid.rds")
```

1.4 Check Level

Finally, let's test out using the case of Miami, what is the best way to aggregate this data? County Subdivision? Census tract?

Hide

```
county <- read_rds("shapes/counties.rds") %>%
  filter(name == "Miami-Dade")
tracts <- read_rds("shapes/tracts.rds") %>%
  filter(str_sub(geoid, 1,5) %in% county$geoid)
sub <- read_rds("shapes/csub.rds") %>%
  filter(str_sub(geoid,1,5) %in% county$geoid)
point <- read_rds("fb_data/dorian_nodes_neighborhood.rds") %>%
  st_join(county) %>%
  filter(!is.na(name))

# Wow! This is absolutely a county subdivision map.
ggplot() +
  geom_sf(data = county, color = "darkgrey", size = 5) +
  geom_sf(data = tracts, fill = NA, color = "grey", size = 0.2) +
  geom_sf(data = sub, fill = NA, color = "darkgrey", size = 0.5) +
  geom_sf(data = point, color = "firebrick", size = 2) +
  theme_void()

remove(county, point, sub, tracts)
```

2. Geolocate Neighborhoods and Edgelist

Hide

```
# Get Albers Equal Area Conic Projection
#https://spatialreference.org/ref/esri/north-america-albers-equal-area-conic/
aea <- "+proj=aea +lat_1=20 +lat_2=60 +lat_0=40 +lon_0=-96 +x_0=0 +y_0=0 +ellps=GRS80 +d
atum=NAD83 +units=m +no_defs"

# Let's get points for each polygon
bind_rows(
  # Import just the origin points
  read_rds("fb_data/dorian_edges_neighborhood.rds") %>%
    as.data.frame() %>%
    select(polygon_id = start_polygon_id,
           x = from_x, y = from_y),
  # Import just the destination points
  read_rds("fb_data/dorian_edges_neighborhood.rds") %>%
    as.data.frame() %>%
    select(polygon_id = end_polygon_id,
           x = to_x, y = to_y)) %>%
  # Remove duplicates
  # Take every polygon_id,
  # shorted each x and y coordinate to 5 decimal places
  # (so that we don't have duplicative points)
  # Some polygons have the same ID but different coordinates;
  # but upon inspection, they are NEARLY identical coordinates;
  # Think: about a 100 meter difference.
  # For each polygon, we're going to grab just ONE coordinate each,
  # rounded to 5 decimal places
  group_by(polygon_id) %>%
  # For each polygon id, grab the first x coordinate of its kind and round to 5
  summarize(x = round(x[1], 5),
            y = round(y[1], 5)) %>%
  ungroup() %>%
  distinct() %>%
  # Create points in C really really fast
  mutate(geometry = sprintf("POINT(%s %s)", x, y)) %>%
  # Set the crs and tell the object to become sf format using the geometry column
  st_as_sf(crs = CRS(paste0("+proj=longlat +ellps=WGS84 +datum=WGS84 +no_defs")),
           wkt = "geometry") %>%
  st_transform(crs = CRS(paste0(aea))) %>%
  # Let's join in the subdivision this neighborhood sits in
  st_join(read_rds("shapes/csub.rds") %>%
            st_transform(crs = CRS(paste0(aea))),
          # Return just the geometry with the largest overlap
          largest = TRUE) %>%
  # Now keep just the points in Louisiana tracts
  filter(!is.na(geoid)) %>%
  # Transform back to WGS84 coordinate reference system, for easy interaction with edges
  st_transform(crs = CRS(paste0("+proj=longlat +ellps=WGS84 +datum=WGS84 +no_defs"))) %
>%
  saveRDS("fb_data/dorian_nodes_neighborhood.rds")
```

2.2 Aggregate to County Subdivision

Now, we want to aggregate to the subdivison level.

Hide

```
# Now, let's extract just the edges within Southeastern Louisiana
edgelist <- read_rds("fb_data/dorian_edges_neighborhood.rds") %>%
  as_tibble() %>%
  # Grab key fields
  select(start_polygon_id, end_polygon_id,
         n_crisis, n_baseline, date_time)
# Import neighborhoods
neighborhoods <- read_rds("fb_data/dorian_nodes_neighborhood.rds") %>%
  as_tibble() %>%
  select(polygon_id, geoid = geoid)


# Get centroids as points, which we can easily join together
centroids <- read_rds("shapes/csub_centroid.rds") %>%
  mutate(x = st_coordinates(geometry)[,1],
         y = st_coordinates(geometry)[,2]) %>%
  as.data.frame() %>%
  select(-geometry) %>%
  distinct()

# Aggregate edgelist from neighborhood to subdivision level
edgelist %>%
  # First, let's join into the edgelist the subdivision of the origin neighborhood
  left_join(by = c("start_polygon_id" = "polygon_id"),
            y = neighborhoods %>%
              rename(from = geoid)) %>%
  # Now let's join into the edgelist the subdivision of the destination neighborhood
  left_join(by = c("end_polygon_id" = "polygon_id"),
            y = neighborhoods %>%
              rename(to = geoid)) %>%
  # Aggregate from neighborhood to subdivision
  # Adding together all people who moved between neighborhoods for this tract
  group_by(from, to, date_time) %>%
  summarize(n_crisis = sum(n_crisis, na.rm = TRUE),
            n_baseline = sum(n_baseline, na.rm = TRUE)) %>%
    ungroup() %>%
  filter(!is.na(from), !is.na(to), !is.na(date_time),
         !is.na(n_crisis), !is.na(n_baseline)) %>%
  # Next, let's join in the origin subdivision centroid points
  left_join(by = c("from" = "geoid"),
            y = centroids %>%
              rename(from_x = x,
                     from_y = y)) %>%
  # Next, let's join in the destination subdivision centroid points
  left_join(by = c("to" = "geoid"),
            y = centroids %>%
              rename(to_x = x,
                     to_y = y)) %>%
  # Filter out any geocodes that didn't make it
  filter(!is.na(from_x) & !is.na(from_y) & !is.na(to_x) & !is.na(to_y))  %>%
  # Create line-springs in C really really fast
  mutate(geometry = sprintf("LINESTRING(%s %s, %s %s)",
                            from_x, from_y,
```

```
                            to_x, to_y)) %>%
  # Set the crs and tell the object to become sf format using the geometry column
  # For the coordinate reference system, let's use:
  # the World Geodetic System from 1984, commonly used with degrees
  # WGS 84 https://spatialreference.org/ref/epsg/wgs-84/
  st_as_sf(crs = CRS(paste0("+proj=longlat +ellps=WGS84 +datum=WGS84 +no_defs")),
           wkt = "geometry") %>%
  # Finally, let's tally evacuation
  # If crisis movement was greater than baseline movement, record it as evacuation;
  # If crisis movement was lower than baseline movement, leave it 0;
  # We don't really want to capture negative evacuation.
  mutate(evacuation = n_crisis - n_baseline) %>%
  select(from, to, date_time, evacuation, geometry) %>%
  saveRDS("raw_data/dorian_edges_csub.rds")


remove(centroids, edgelist, neighborhoods)
```