Getting Distance In Communities

Shukry Zablah

2019-10-21

Question and Approach

Question: How many miles were driven per community?

Approach: 1) Find way to get distance of a route. 2) Find way to decrease resolution of route data (get only n equally spaced points throughout time 1 to T). 3) Find way to define the space of the communities 4) Find out which community entry corresponds to 5) Process all files and aggregate based on community (which is now on every entry in the gps data).

Tools

Our usual favorites and the `sf` simple features package for working with spatial data.

```
library(dplyr)
library(purrr)
library(tidyr)
library(leaflet)
library(sf)
```

We have already the stations and the communities. I got the communities by clustering around 6 centers using k means to classify all the stations (they all were correct). Then I process and end up doing a Voronoi tesellation with the help of the sf package to get the polygon regions that will be our communities. **Thus we already solved point # 3**

```
load("../data/stations.rda")
load("../data/communities.rda")
```

```
glimpse(stations)
```

```
## Observations: 54
## Variables: 7
## $ serial_num     <dbl> 19, 50, 17, 22, 23, 13, 30, 603, 45, 2, 35, 43,...
## $ address        <chr> "330 Homestead Avenue Holyoke Community College...
## $ station_name   <chr> "Holyoke Community College", "Congress Street",...
## $ num_docks      <int> 16, 10, 15, 17, 13, 4, 16, 10, 9, 16, 14, 10, 2...
## $ latitude       <dbl> 42.19513, 42.10925, 42.19757, 42.32858, 42.3167...
## $ longitude      <dbl> -72.65270, -72.59456, -72.60377, -72.64394, -72...
## $ community_name <chr> "Holyoke", "Springfield", "Holyoke", "Northampt...
```

```
communities
```

```
## Simple feature collection with 6 features and 2 fields
## geometry type:  POLYGON
## dimension:      XY
## bbox:           xmin: -72.94247 ymin: 41.83314 xmax: -72.25085 ymax: 42.65747
## epsg (SRID):    4326
## proj4string:    +proj=longlat +datum=WGS84 +no_defs
##   id community_name                       location
## 1  1    Easthampton POLYGON ((-72.94247 42.0402...
```

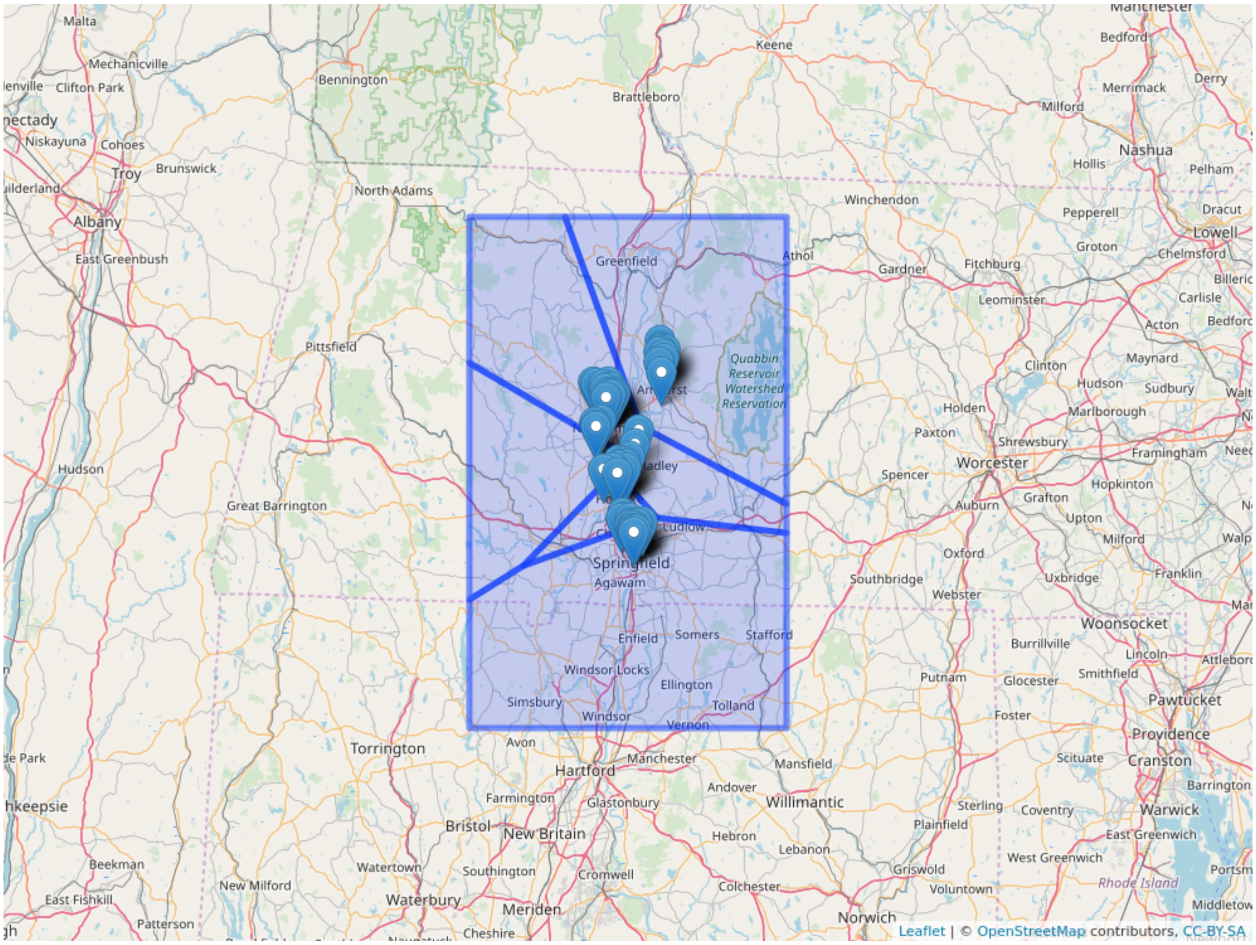

Figure 1: map of stations on communities in pioneer valley

```
## 2  2   South Hadley POLYGON ((-72.25085 42.1959...
## 3  3    Northampton POLYGON ((-72.94247 42.4231...
## 4  4        Amherst POLYGON ((-72.73426 42.6574...
## 5  5        Holyoke POLYGON ((-72.62578 42.2484...
## 6  6    Springfield POLYGON ((-72.25085 42.1488...
```

```
stations <- stations %>%
    mutate(location = map2(longitude, latitude,
                           ~ st_point(c(.x, .y)))) %>%
    st_as_sf() %>%
    st_set_crs(4326)
```

Right now we can see already how the polygons and the spatial data we got of the stations will help us answer our question.

```
leaflet() %>%
    addTiles() %>%
    addPolygons(data = communities, popup = ~ community_name) %>%
    addMarkers(data = stations) %>%
    mapview::mapshot(file = "../resources/stations_on_communities.png")
```

This function solves point # 2. Decreasing the resolution of the data.

```
make_data_thinner <- function(data, num_obs) {
    if(num_obs == nrow(data)) {
        return(data)
    }
    n <- min(nrow(data), num_obs)
    mask <- round(seq(1, nrow(data), length.out = n))
    thinner_data <- data[mask, ]
    return(thinner_data)
}
```

Here is an example to get intuition behind function.

```
make_data_thinner(mtcars, num_obs = 3)
```

```
##                      mpg cyl disp  hp drat    wt  qsec vs am gear carb
## Mazda RX4           21.0   6  160 110 3.90 2.620 16.46  0  1    4    4
## Lincoln Continental 10.4   8  460 215 3.00 5.424 17.82  0  0    3    4
## Volvo 142E          21.4   4  121 109 4.11 2.780 18.60  1  1    4    2
```

These function solves point # 4. Defining the community at the current entry of the gps data.

```
.get_current_community <- function(polygons, point) {
    mask <- st_contains(polygons$location, point, sparse = F)
    res <- polygons$community_name[mask]
    if(length(res) == 0) NA else res
}

get_current_community <- quietly(.get_current_community)
```

This function solves #1 by using the geosphere package to calculate the N-1 distance values between N (long, lat) points. We could use other distance measures but the important part is that this happens after the resolution of the data is decided on.

```
get_path_distance <- function(data) {
    data %>%
        as.data.frame() %>%
        select(longitude, latitude) %>%
        geosphere::distHaversine() %>%
        sum()
}

get_distance <- possibly(get_path_distance, NA)
```

And this is what we will do for every file.

Review:

1) Read in and choose data resolution (equally spaced points through time for route). Also creation of spatial feature column for use in 2.
2) For every route entry, determine which community it is currently in.
3) Grouping by route and community, get the corresponding distances, and then aggregate based on the community.

```
get_distance_community <- function(file_path, NUM, comment = TRUE) {
    if(comment == TRUE) {
        print(file_path)
    }

    day <- file_path %>%
        data.table::fread(skip = 2) %>%
        janitor::clean_names() %>%
        group_by(route_id, bike, user_id) %>%
        nest() %>%
        mutate(data = map(data,
                          ~ make_data_thinner(.x, num_obs = NUM))) %>%
        unnest(data) %>%
        mutate(location = map2(longitude, latitude,
                               ~ st_point(c(.x, .y)))) %>%
        st_as_sf() %>%
        st_set_crs(4326)

    day <- day %>%
        mutate(community = map_chr(
                   location, ~ get_current_community(communities,
                                                     .x)$result
               ))

    distances <- day %>%
        group_by(route_id, community) %>%
        nest() %>%
        mutate(distance = map_dbl(data, get_distance)) %>%
        group_by(community) %>%
        summarize(total_distance = sum(distance, na.rm=T))

    return(distances)
}
```

```
empty <- tibble(community = NA_character_,
                total_distance = NA_real_)

possibly_get_distance_community <- possibly(get_distance_community,
                                            otherwise = empty)
```

```
file_names <- fs::dir_ls("../inst/extdata",
                         regexp = "VB_Routes_Data_2019_.*")

res <- file_names %>%
    map_dfr(possibly_get_distance_community, NUM=10, comment=F)
```

```
per_community <- res %>% group_by(community) %>% summarize(total_distance = sum(total_distance) / 1609.3

per_community
```

```
## # A tibble: 7 x 2
##   community    total_distance
##   <chr>                 <dbl>
## 1 Amherst              30428.
## 2 Easthampton           6278.
## 3 Holyoke              15290.
## 4 Northampton          38425.
## 5 South Hadley          1263.
## 6 Springfield          18087.
## 7 <NA>                       NA
```

```
per_community %>% summarize(total_distance = sum(total_distance, na.rm=TRUE))
```

```
## # A tibble: 1 x 1
##   total_distance
##            <dbl>
## 1        109771.
```