Analysis of Data from Location-Based Social Networks (LBSN)

Michael Dorman
Geography and Environmental Development

2019-02-25

Ben-Gurion University of the Negev

Aim

- Location-Based Social Network (LBSN)
  - Practical methods for processing and analysis
  - Use in social sciences / geography research
- Materials on Dropbox -
  - Slides
  - Code
  - Data

Requirements

- Several packages **need to be installed** to run code examples -

```
install.packages("rtweet")
install.packages("sf")
install.packages("mapsapi")
install.packages("rworldmap")
install.packages("igraph")
install.packages("sentimentr")
```

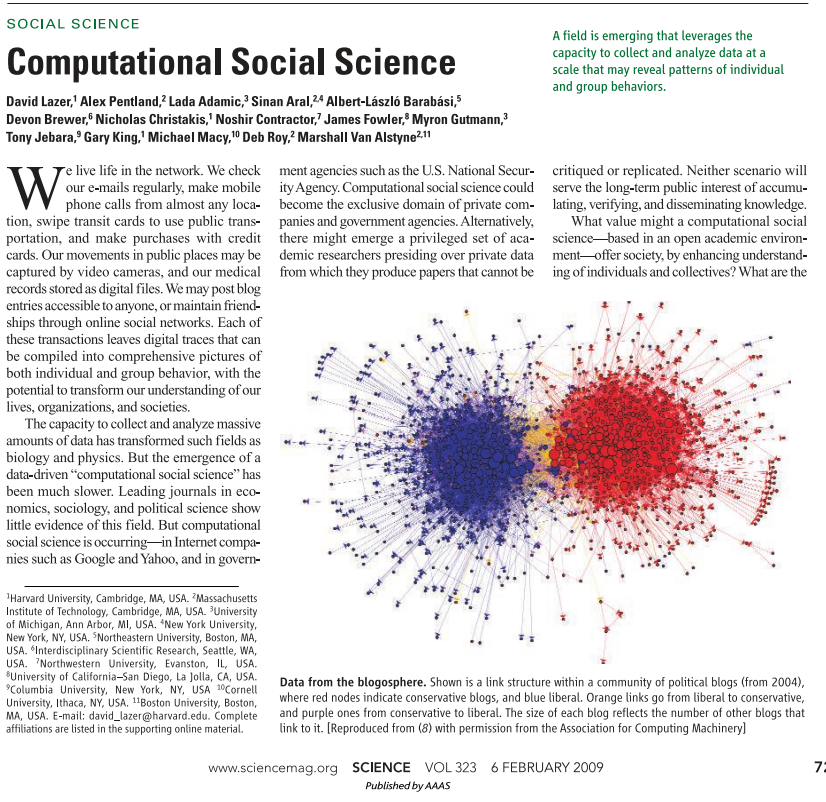
- Set to **working directory** with the data -
  - `stream_boston.json` - **Boston tweets**
  - `borders.shp` - **County borders (Shapefile)**
  - `network.csv` - **Twitter follower edge list**
  - `locations.csv` - **Twitter user locations**
  - `gcp.shp` - **Geocoding result (Shapefile)**

Contents

- Part I: Introduction
  - Location-Based Social Networks (LBSN)
  - Types of LBSN data
  - Twitter APIs
- Part II: Practical Examples
  - Example 1: Setting up Twitter API access
  - Example 2: Collecting tweets
  - Example 3: Analyzing spatial patterns
  - Example 4: Collecting network data
  - Example 5: Network analysis
  - Example 6: Sentiment analysis

Introduction - Location-Based Social Networks (LBSN)

- Social Networks where location information is part of the shared contents are called **Location Based Social Networks** (LBSN)
- LBSN provide new opportunities to study spatial dimensions of human behavior (Steiger *et al.*, 2015)



SOCIAL SCIENCE

Computational Social Science

A field is emerging that leverages the capacity to collect and analyze data at a scale that may reveal patterns of individual and group behaviors.

David Lazer,[1] Alex Pentland,[2] Lada Adamic,[3] Sinan Aral,[2,4] Albert-László Barabási,[5] Devon Brewer,[6] Nicholas Christakis,[1] Noshir Contractor,[7] James Fowler,[8] Myron Gutmann,[3] Tony Jebara,[9] Gary King,[1] Michael Macy,[10] Deb Roy,[2] Marshall Van Alstyne[2,11]

We live life in the network. We check our e-mails regularly, make mobile phone calls from almost any location, swipe transit cards to use public transportation, and make purchases with credit cards. Our movements in public places may be captured by video cameras, and our medical records stored as digital files. We may post blog entries accessible to anyone, or maintain friendships through online social networks. Each of these transactions leaves digital traces that can be compiled into comprehensive pictures of both individual and group behavior, with the potential to transform our understanding of our lives, organizations, and societies.

The capacity to collect and analyze massive amounts of data has transformed such fields as biology and physics. But the emergence of a data-driven "computational social science" has been much slower. Leading journals in economics, sociology, and political science show little evidence of this field. But computational social science is occurring—in Internet companies such as Google and Yahoo, and in government agencies such as the U.S. National Security Agency. Computational social science could become the exclusive domain of private companies and government agencies. Alternatively, there might emerge a privileged set of academic researchers presiding over private data from which they produce papers that cannot be critiqued or replicated. Neither scenario will serve the long-term public interest of accumulating, verifying, and disseminating knowledge.

What value might a computational social science—based in an open academic environment—offer society, by enhancing understanding of individuals and collectives? What are the

[1]Harvard University, Cambridge, MA, USA. [2]Massachusetts Institute of Technology, Cambridge, MA, USA. [3]University of Michigan, Ann Arbor, MI, USA. [4]New York University, New York, NY, USA. [5]Northeastern University, Boston, MA, USA. [6]Interdisciplinary Scientific Research, Seattle, WA, USA. [7]Northwestern University, Evanston, IL, USA. [8]University of California–San Diego, La Jolla, CA, USA. [9]Columbia University, New York, NY, USA [10]Cornell University, Ithaca, NY, USA. [11]Boston University, Boston, MA, USA. E-mail: david_lazer@harvard.edu. Complete affiliations are listed in the supporting online material.

Data from the blogosphere. Shown is a link structure within a community of political blogs (from 2004), where red nodes indicate conservative blogs, and blue liberal. Orange links go from liberal to conservative, and purple ones from conservative to liberal. The size of each blog reflects the number of other blogs that link to it. [Reproduced from (*8*) with permission from the Association for Computing Machinery]

Published by AAAS

Figure 1: Lazer *et al.* 2009, Science

Introduction - Location-Based Social Networks (LBSN)

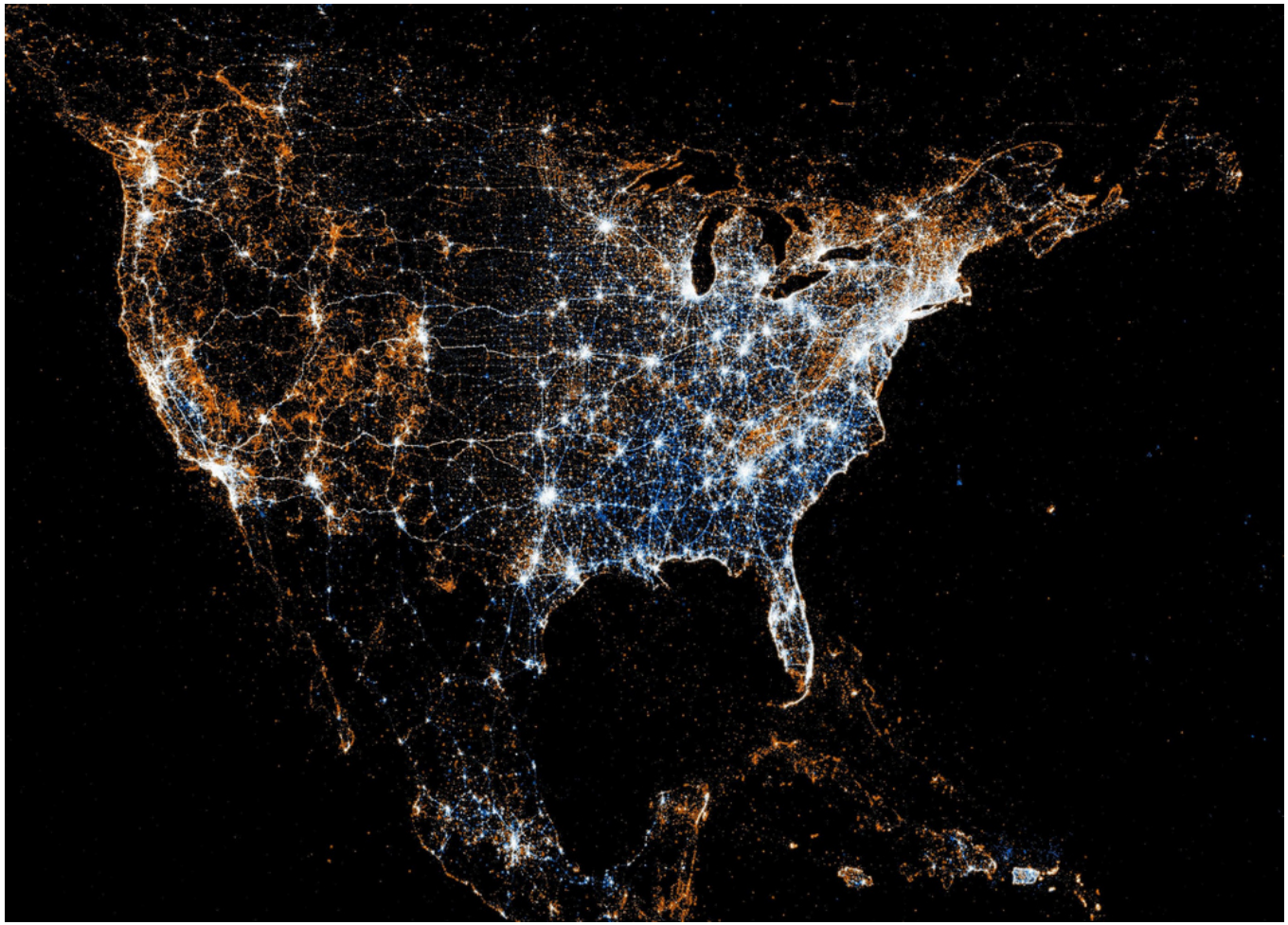


Figure 2: Density of Twitter (blue) and Flickr (red) data in the US[1]

[1] https://www.flickr.com/photos/walkingsf/5912385701

Introduction - LBSN types of data

- Spatial information
  - **Point** – Spatial location of LBSN posts
  - **Path** – Chronologically ordered locations from posts of a given user at different times
- Non-spatial information
  - **User identity** – The division of LBSN data among distinct users (as opposed to disregarding contributor identity)
  - **Text** – Textual components of the post (e.g. Twitter message text or Flickr image textual tags)
  - **Time** – The time-stamp of each post
  - **Network** – The social relations graph (i.e. presence of friends/followers relations between each pair of users)
  - **Image** – Images associated with LBSN posts
  - **Video** – Videos associated with LBSN posts

Introduction - LBSN types of data

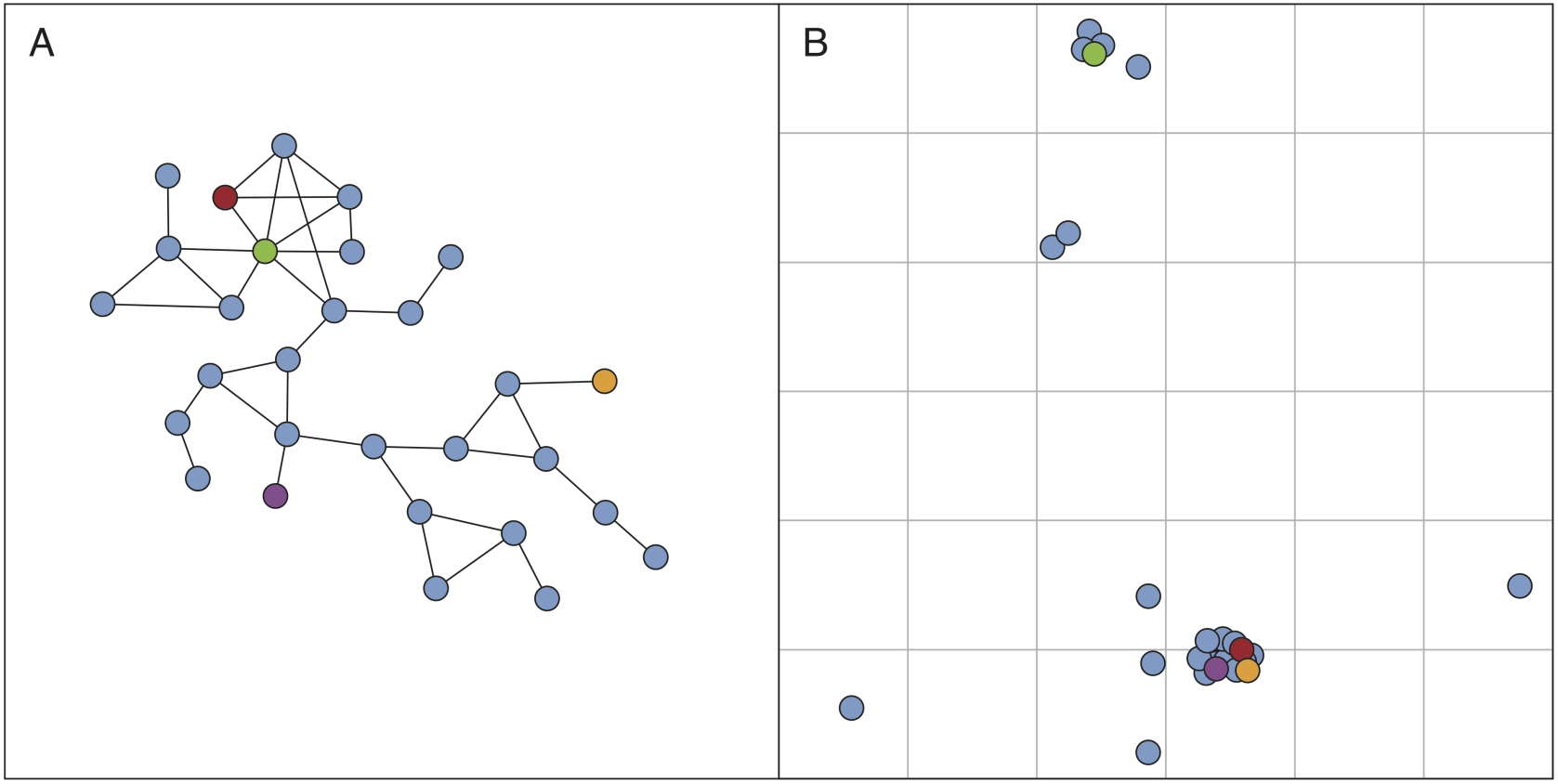


Figure 3: Two points of view on LBSN data: (A) network structure and (B) geographical structure (Onella *et al.* 2011)

Introduction - Twitter APIs

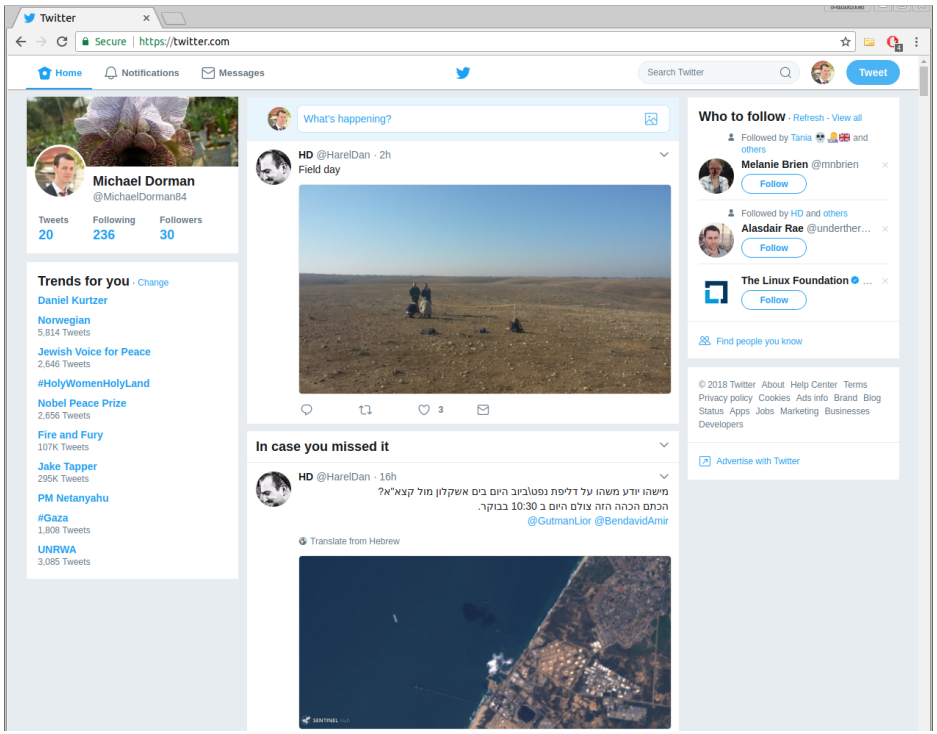


Figure 4: Twitter home page

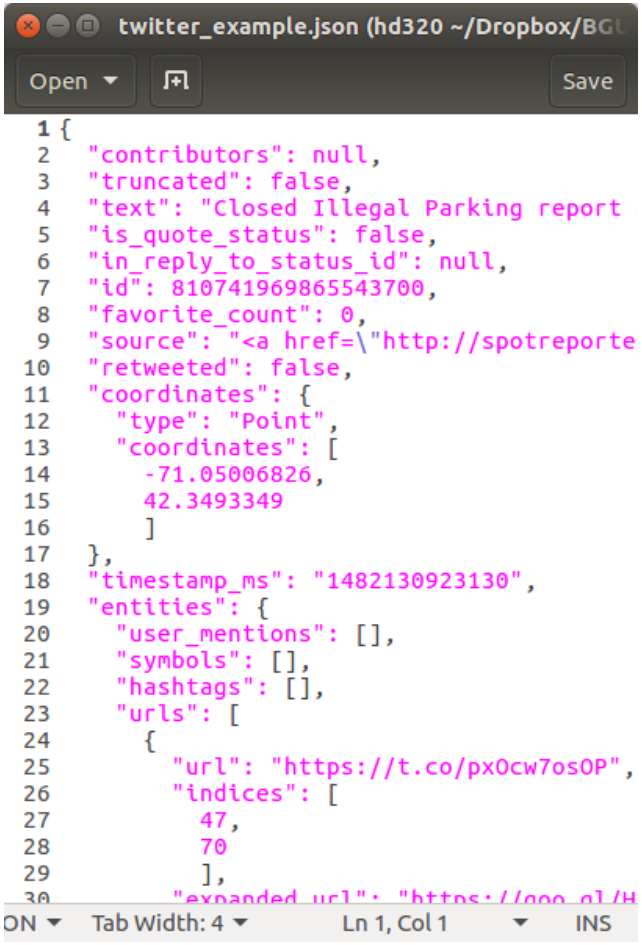


Figure 5: Tweet data

[1] https://developer.twitter.com/en/docs

Example 1: Setting up Twitter account

To access the Twitter API we need to obtain **API keys**

1. Make sure you have a **Twitter account**
2. Navigate to **https://developer.twitter.com/en/apps** and create a new app by providing a Name, Description, Website and Callback URL
3. Check **Yes** to agree and then click **"Create your Twitter application"**
4. Once you've successfully created an app, click the tab labeled **Keys and Access Tokens** to retrieve your keys

Example 1: Setting up Twitter account

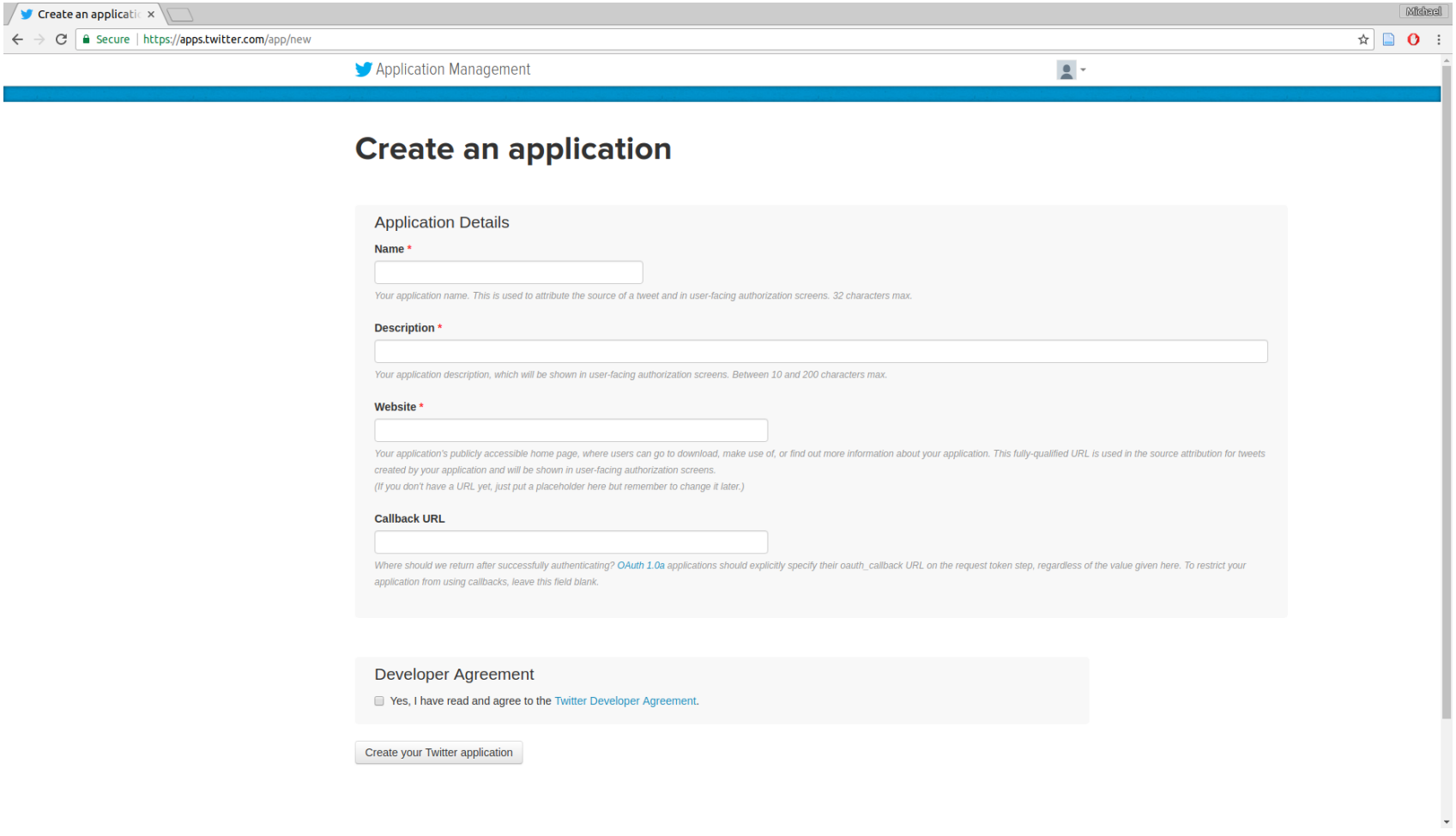


Figure 6: Creating a Twitter application

Example 1: Setting up Twitter account

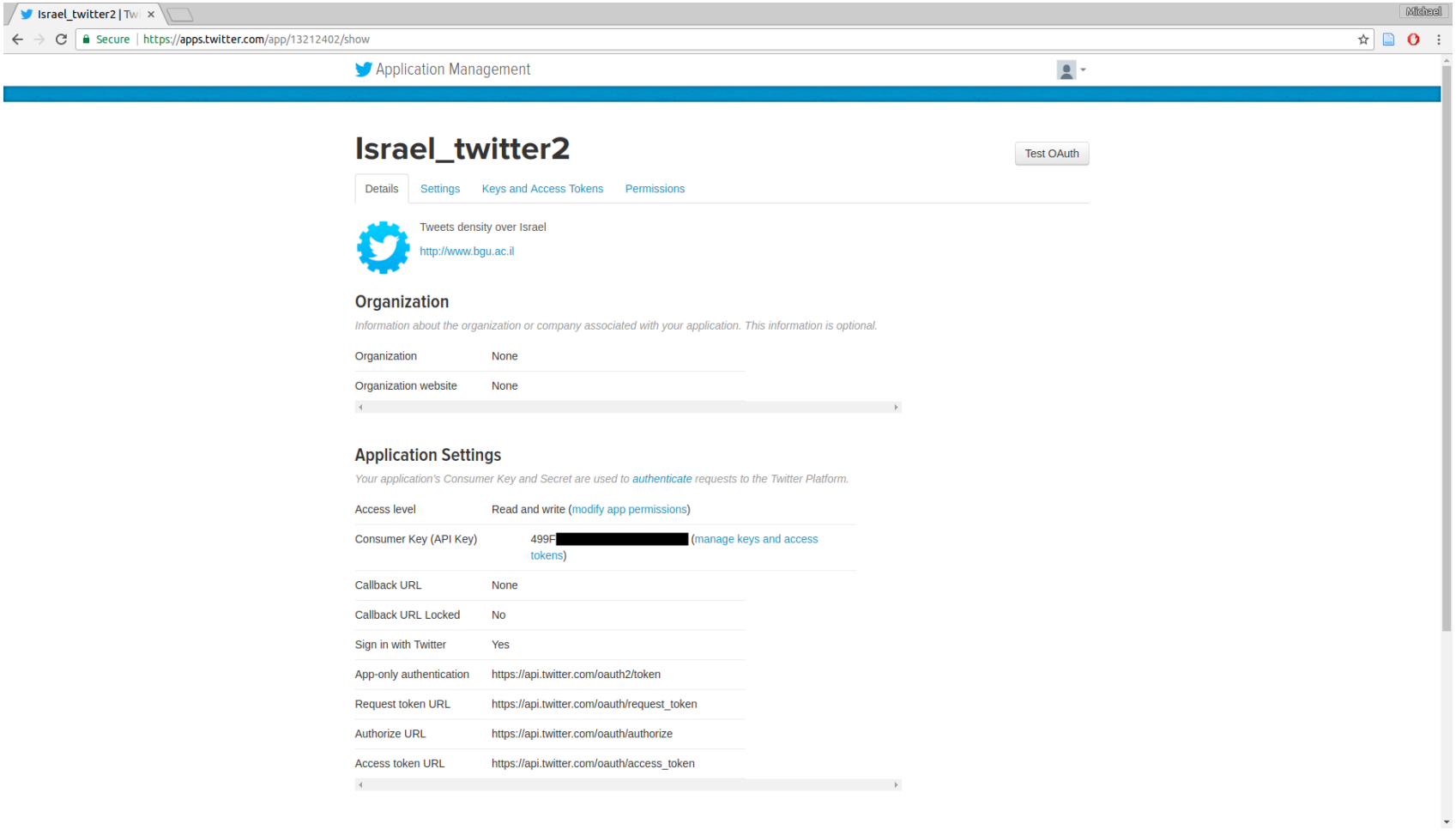


Figure 7: Accessing application settings

Example 1: Setting up Twitter account

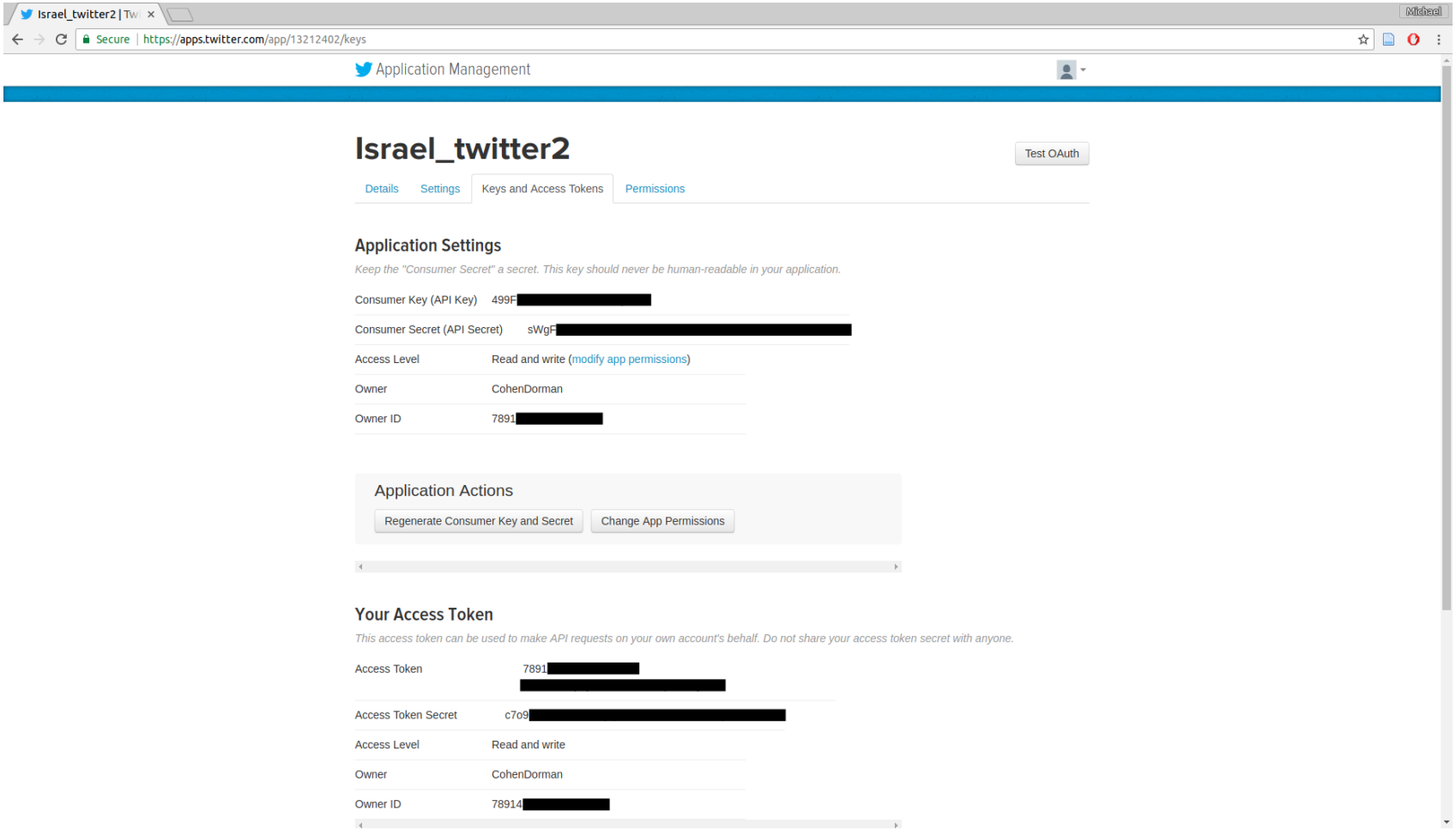


Figure 8: Obtaining Twitter API keys

Example 1: Setting up Twitter account

- **Twitter APIs** can be accessed from R using the `rtweet` R package
- The first step is **establishing a connection**
- **Twitter API keys** are needed to run the `create_token` function

```
library(rtweet)

twitter_token = create_token(
  app = "Israel_twitter2",
  consumer_key = "499F...",
  consumer_secret = "sWgF..."
)
```

Example 2: Collecting tweets

- A **live stream of Twitter data** can be collected with the `stream_tweets` function
- The function establishes a continuous connection to the **Streaming API**
- The function accepts -
  - `q` - **search terms** and/or **geographical extent**
  - `timeout` - Amount of **time** to leave connection open
  - `file_name` - **JSON file name** the data are written to

Example 2: Collecting tweets

- For example, the following expression establishes a **24-hour** connection for collecting tweets posted around **Boston** -

```
stream_tweets(
  q = c(-72.21437, 41.19034, -69.64939, 43.30924),
  timeout = 60 * 60 * 24,
  parse = FALSE,
  file_name = "stream_boston.json",
  token = twitter_token
)
```

Example 2: Collecting tweets

- The JSON file can then be **pre-processed** to a `data.frame` using `parse_stream` -

```
dat = parse_stream("stream_boston.json")
dat = as.data.frame(dat)
```

- We have data for **14,048 tweets** -

```
dim(dat)
## [1] 14048    88
```

Example 2: Collecting tweets

- The resulting table contains **a lot of variables**
- We will keep just the **most useful** ones -

```
vars = c(
  "user_id",
  "screen_name",
  "created_at",
  "text",
  "lang",
  "geo_coords"
)
dat = dat[, vars]
```

Example 2: Collecting tweets

```
head(dat[, c("screen_name", "created_at", "geo_coords")])
##    screen_name          created_at geo_coords
## 1 iclivecooper 2018-01-09 07:06:03     NA, NA
## 2 iclivecooper 2018-01-09 07:10:36     NA, NA
## 3 iclivecooper 2018-01-09 07:06:16     NA, NA
## 4 iclivecooper 2018-01-09 07:06:09     NA, NA
## 5 iclivecooper 2018-01-09 07:10:28     NA, NA
## 6 iclivecooper 2018-01-09 07:06:22     NA, NA
```

Example 3: Analyzing spatial patterns

- The `created_at` variable is a `POSIXct` **date-time** object -

```
class(dat$created_at)
## [1] "POSIXct" "POSIXt"

dat$created_at[1]
## [1] "2018-01-09 07:06:03 UTC"
```

Example 3: Analyzing spatial patterns

- It is more convenient to work with **local times** rather than `UTC`
- **Boston** is in the `US/Eastern` time zone -

```
dat$created_at[1]
## [1] "2018-01-09 07:06:03 UTC"

dat$created_at = format(
  dat$created_at,
  tz = "US/Eastern"
)
dat$created_at = as.POSIXct(
  dat$created_at,
  tz = "US/Eastern"
)

dat$created_at[1]
## [1] "2018-01-09 02:06:03 EST"
```

Example 3: Analyzing spatial patterns

- Now we can find out the **time frame** -

```
min(dat$created_at)
## [1] "2018-01-09 02:06:03 EST"

max(dat$created_at)
## [1] "2018-01-09 12:52:41 EST"

max(dat$created_at) - min(dat$created_at)
## Time difference of 10.77722 hours
```

Example 3: Analyzing spatial patterns

- The `geo_coords` column is a `list` containing **geo-location data** for each tweet
- Missing data means that location was **not shared** from user device

```
dat$geo_coords[1:3]
## [[1]]
## [1] NA NA
##
## [[2]]
## [1] NA NA
##
## [[3]]
## [1] NA NA
```

Example 3: Analyzing spatial patterns

- To work with the spatial component we first **separate** the `lon` and `lat` columns -

```
dat$lon = sapply(dat$geo_coords, "[", 2)
dat$lon[1:4]
## [1] NA NA NA NA

dat$lat = sapply(dat$geo_coords, "[", 1)
dat$lat[1:4]
## [1] NA NA NA NA
```

- The `geo_coords` column is **no longer necessary** -

```
dat$geo_coords = NULL
```

Example 3: Analyzing spatial patterns

```
head(dat[, c("screen_name", "created_at", "lon", "lat")])
##    screen_name          created_at lon lat
## 1 iclivecooper 2018-01-09 02:06:03  NA  NA
## 2 iclivecooper 2018-01-09 02:10:36  NA  NA
## 3 iclivecooper 2018-01-09 02:06:16  NA  NA
## 4 iclivecooper 2018-01-09 02:06:09  NA  NA
## 5 iclivecooper 2018-01-09 02:10:28  NA  NA
## 6 iclivecooper 2018-01-09 02:06:22  NA  NA
```

Example 3: Analyzing spatial patterns

- For spatial analysis we are only interested in the **geo-referenced** tweets -

```
geo = complete.cases(dat[, c("lon", "lat")])
geo[1:4]
## [1] FALSE FALSE FALSE FALSE

dat = dat[geo, ]
```

- We are left with just **2,609 geo-referenced tweets** -

```
dim(dat)
## [1] 2609    7
```

Example 3: Analyzing spatial patterns

- Next, we convert the table to an `sf` **point layer** -

```
library(sf)
## Linking to GEOS 3.6.2, GDAL 2.2.3, PROJ 4.9.3

pnt = st_as_sf(
  x = dat,
  coords = c("lon", "lat"),
  crs = 4326
)
```

Example 3: Analyzing spatial patterns

```
head(pnt[, c("created_at")])
## Simple feature collection with 6 features and 1 field
## geometry type:  POINT
## dimension:      XY
## bbox:           xmin: -71.0637 ymin: 42.35842 xmax: -71
## epsg (SRID):    4326
## proj4string:    +proj=longlat +datum=WGS84 +no_defs
##             created_at                    geometry
## 53 2018-01-09 03:16:21 POINT (-71.0637 42.35842)
## 54 2018-01-09 02:46:07 POINT (-71.0637 42.35842)
## 55 2018-01-09 07:56:12 POINT (-71.0637 42.35842)
## 56 2018-01-09 04:06:04 POINT (-71.0637 42.35842)
## 57 2018-01-09 06:46:07 POINT (-71.0637 42.35842)
## 58 2018-01-09 02:06:25 POINT (-71.0637 42.35842)
```

Example 3: Analyzing spatial patterns

- The **bounding box** we used for our query can also be built into a polygonal layer -

```
p = c(-72.21437, 41.19034, -69.64939, 43.30924)
p1 = st_point(c(p[1], p[2]))
p2 = st_point(c(p[1], p[4]))
p3 = st_point(c(p[3], p[4]))
p4 = st_point(c(p[3], p[2]))
b = c(p1, p2, p3, p4)
b = st_convex_hull(b)
```

- Plotting both layers shows that the twitter stream was **not strictly limited** to the queried extent -

```
plot(st_geometry(pnt))
plot(b, add = TRUE, border = "red")
```

Example 3: Analyzing spatial patterns

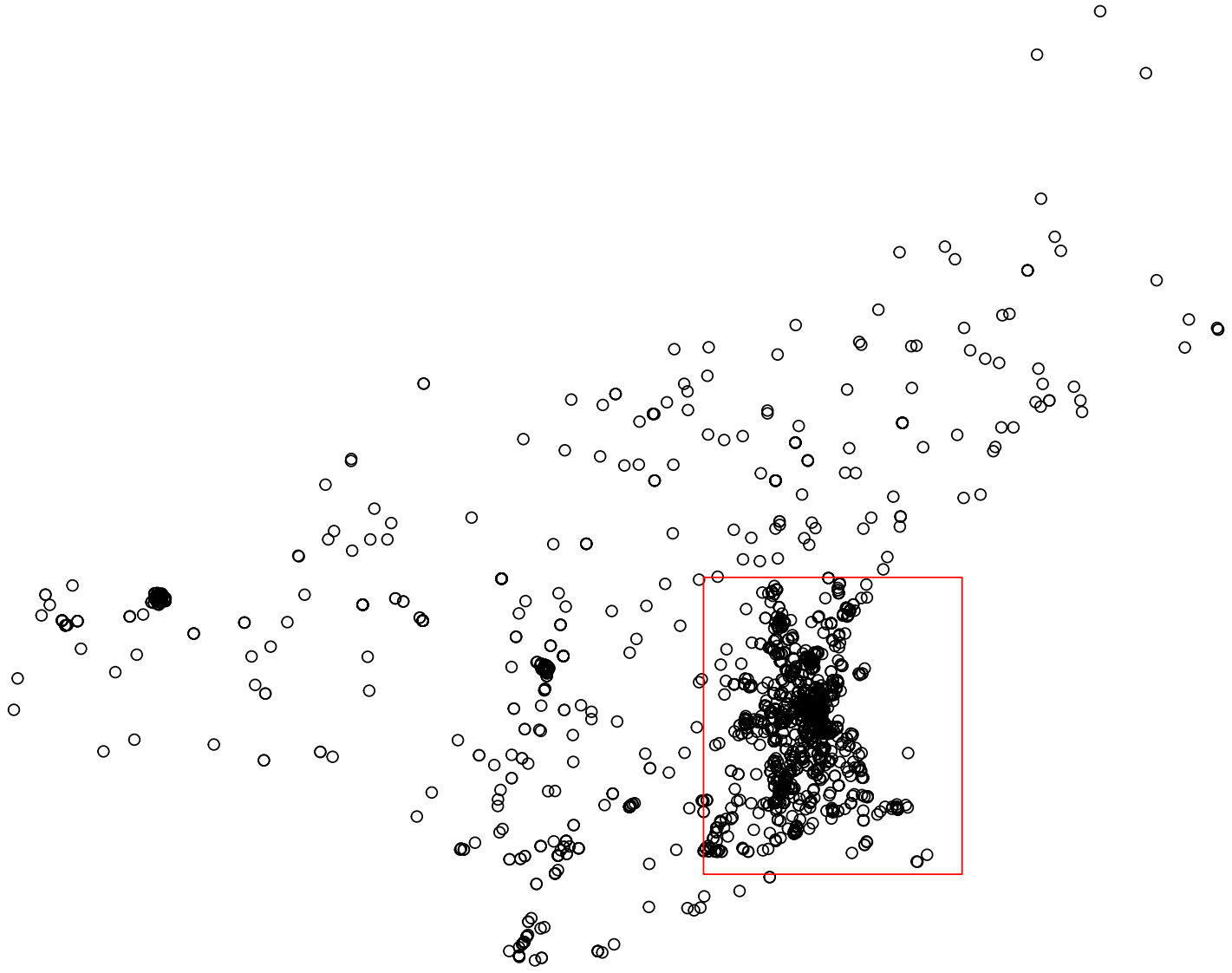


Figure 9: Tweets and query extent

Example 3: Analyzing spatial patterns

- We can **subset** the twitter data according to the **extent polygon** -

```
pnt = pnt[b, ]
```

Example 3: Analyzing spatial patterns

Figure 10: Tweets and query extent

Example 3: Analyzing spatial patterns

- We will find out **which county each tweet falls in**, using a country borders layer
- The layer is being read from a **Shapefile** -

```
borders = st_read("borders.shp")
```

- The following expressions **visualize** the three spatial layers we now have -
  - `borders` - The **county borders**
  - `b` - The **bounding box**
  - `pnt` - **Geo-referenced Tweets**

```
plot(st_geometry(borders), border = "grey")
plot(b, add = TRUE, border = "red")
plot(st_geometry(pnt), add = TRUE)
```

Example 3: Analyzing spatial patterns

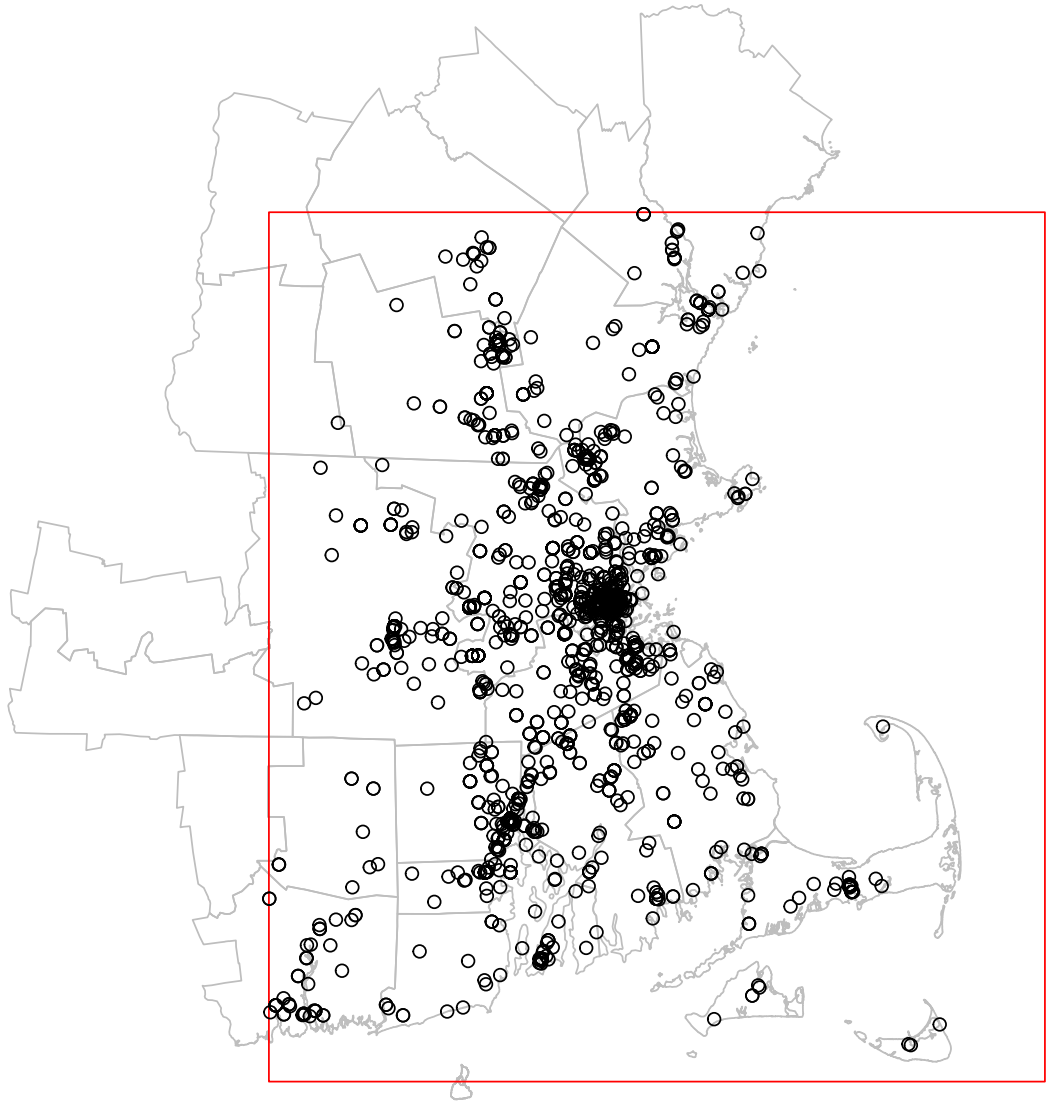


Figure 11: Tweets, query extent and county borders

Example 3: Analyzing spatial patterns

- The simplest **temporal aggregation** is omitting some of the time components, then counting occurences
- For example, calculating a **date+hour** variable with `format` -

```
pnt$hour = format(pnt$created_at, "%m-%d %H")
pnt$hour[1]
## [1] "01-09 03"
```

- Then **counting occurences** with `table` -

```
tab = table(pnt$hour)
tab[1]
## 01-09 02
##       26
```

Example 3: Analyzing spatial patterns

```
barplot(tab, ylab = "Tweets", las = 3)
```

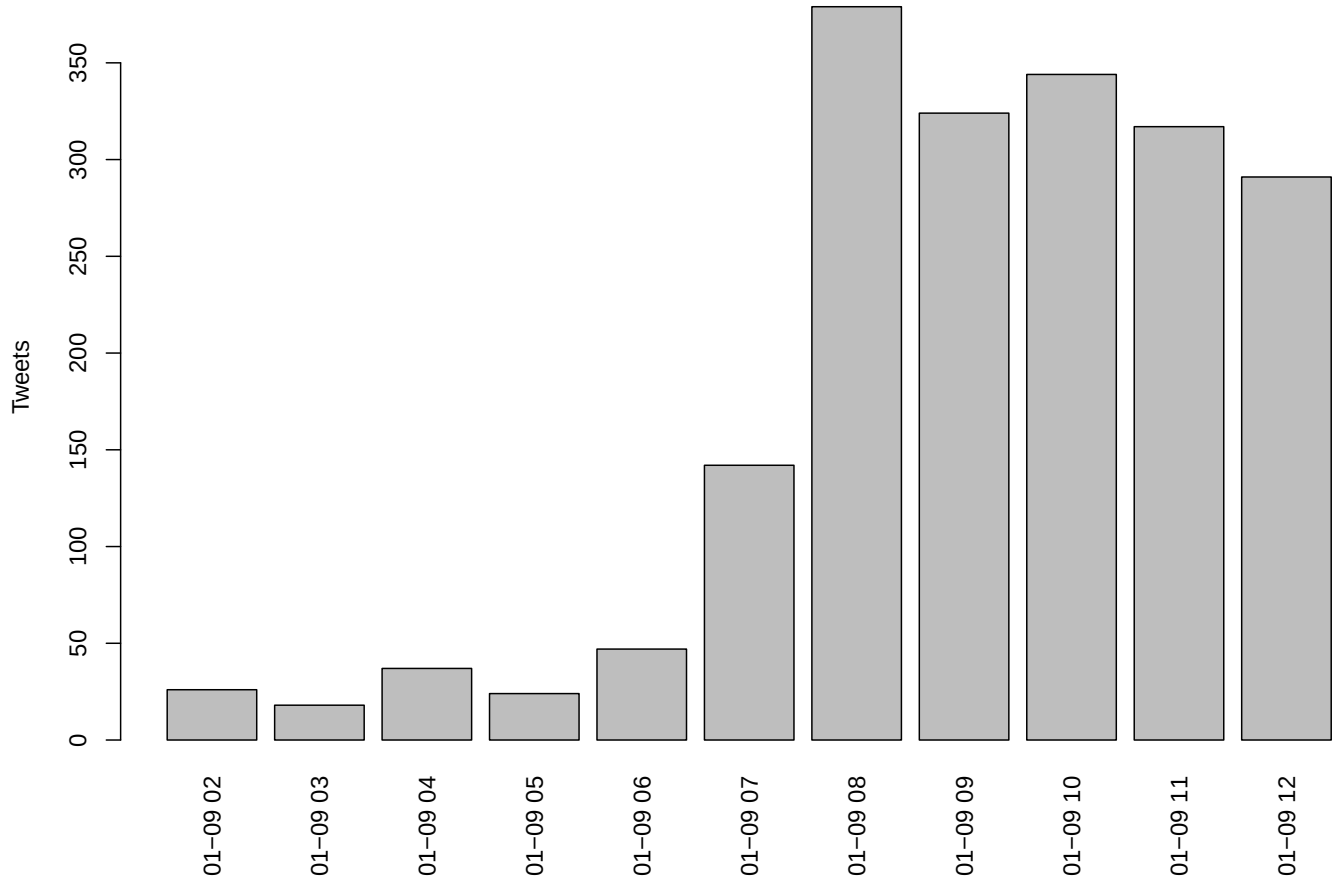


Figure 12: Twitter activity over time

Example 3: Analyzing spatial patterns

- For **spatio-temporal aggregation** we count the location/time occurences
- For example, we can do a **spatial join** between the tweets layer and the counties layer -

```
pnt = st_join(pnt, borders)

pnt1 = st_set_geometry(
  pnt[, c("hour", "NAME_2")],
  NULL
)
```

Example 3: Analyzing spatial patterns

- The result is a table with **date+hour** / **county** per tweet -

```
head(pnt1)
##          hour  NAME_2
## 53 01-09 03 Suffolk
## 54 01-09 02 Suffolk
## 55 01-09 07 Suffolk
## 56 01-09 04 Suffolk
## 57 01-09 06 Suffolk
## 58 01-09 02 Suffolk
```

Example 3: Analyzing spatial patterns

- Finally, we **count occurences** of each **date+hour / county** value -

```
tab = table(pnt1)

tab[1:6, 1:4]
##             NAME_2
## hour         Barnstable Belknap Bristol Cheshire
##   01-09 02            1       0       2        0
##   01-09 03            0       0       0        0
##   01-09 04            0       0       3        0
##   01-09 05            0       0       1        0
##   01-09 06            0       0       3        0
##   01-09 07            4       0       5        0
```

Example 3: Analyzing spatial patterns

- The table can be displayed as a **heatmap** with `image` -

```
image(tab, axes = FALSE)
axis(
  2, at = seq(0, 1, 1/(ncol(tab)-1)),
  labels = colnames(tab),
  las = 2, lwd = 0, lwd.ticks = 1, cex.axis = 0.75
)
axis(
  1, at = seq(0, 1, 1/(nrow(tab)-1)),
  labels = rownames(tab),
  las = 1, lwd = 0, lwd.ticks = 1, cex.axis = 0.75
)
```

Example 3: Analyzing spatial patterns

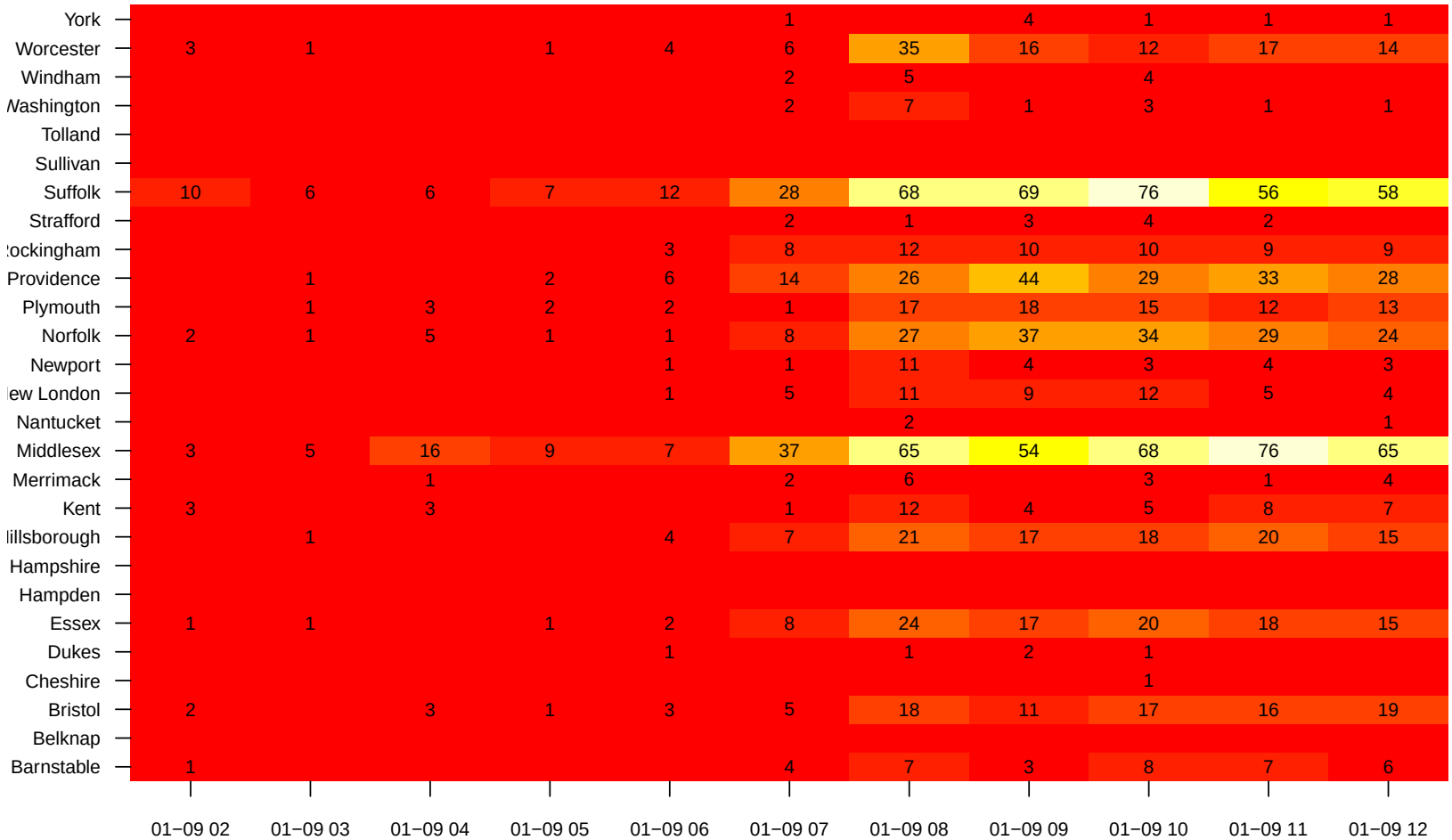


Figure 13: Tweets per county per hour

Example 3: Analyzing spatial patterns

- Chronologically ordered observations per user represent his **path** in space
- To create the **line layer of paths** we first **split** the point layer by user -

```
path = split(pnt, pnt$user_id)
```

- Keep in mind that **most content** is created by **few dominant users** -

```
n = sapply(path, nrow)
barplot(
  table(n),
  xlab = "Tweets",
  ylab = "Number of users"
)
```

Example 3: Analyzing spatial patterns

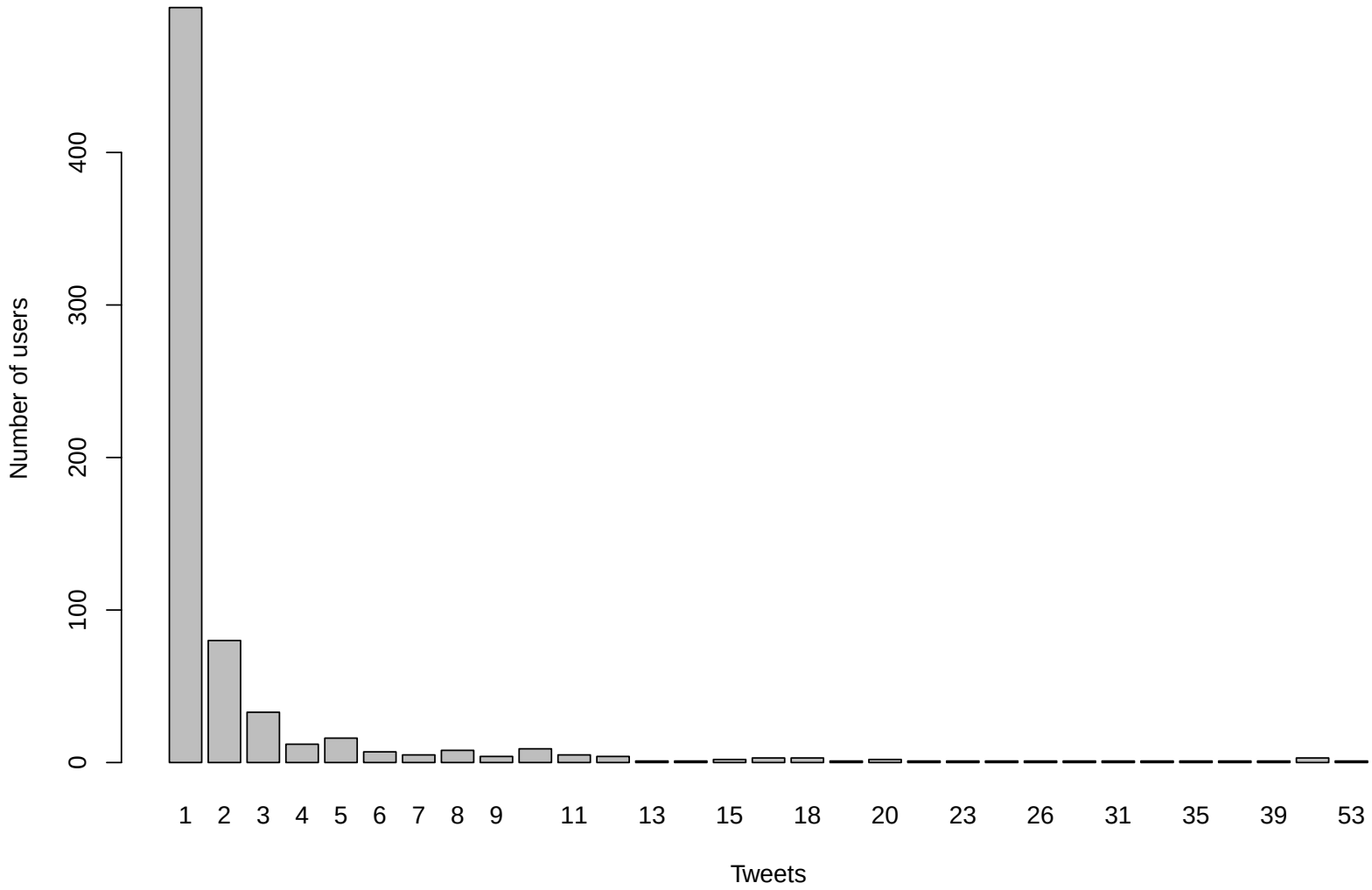


Figure 14: Frequency of users with different tweet counts

Example 3: Analyzing spatial patterns

- Next we **connect** the points into a line layer -

```
path = path[n > 1]
path = lapply(path, st_combine)
path = lapply(path, st_cast, "LINESTRING")
path = do.call(c, path)
```

- **Plotting** the paths layer -

```
plot(st_geometry(borders), border = "grey")
plot(b, add = TRUE, border = "red")
plot(st_geometry(path), add = TRUE)
```

Example 3: Analyzing spatial patterns

Figure 15: Twitter user paths

LBSN research applications (1)

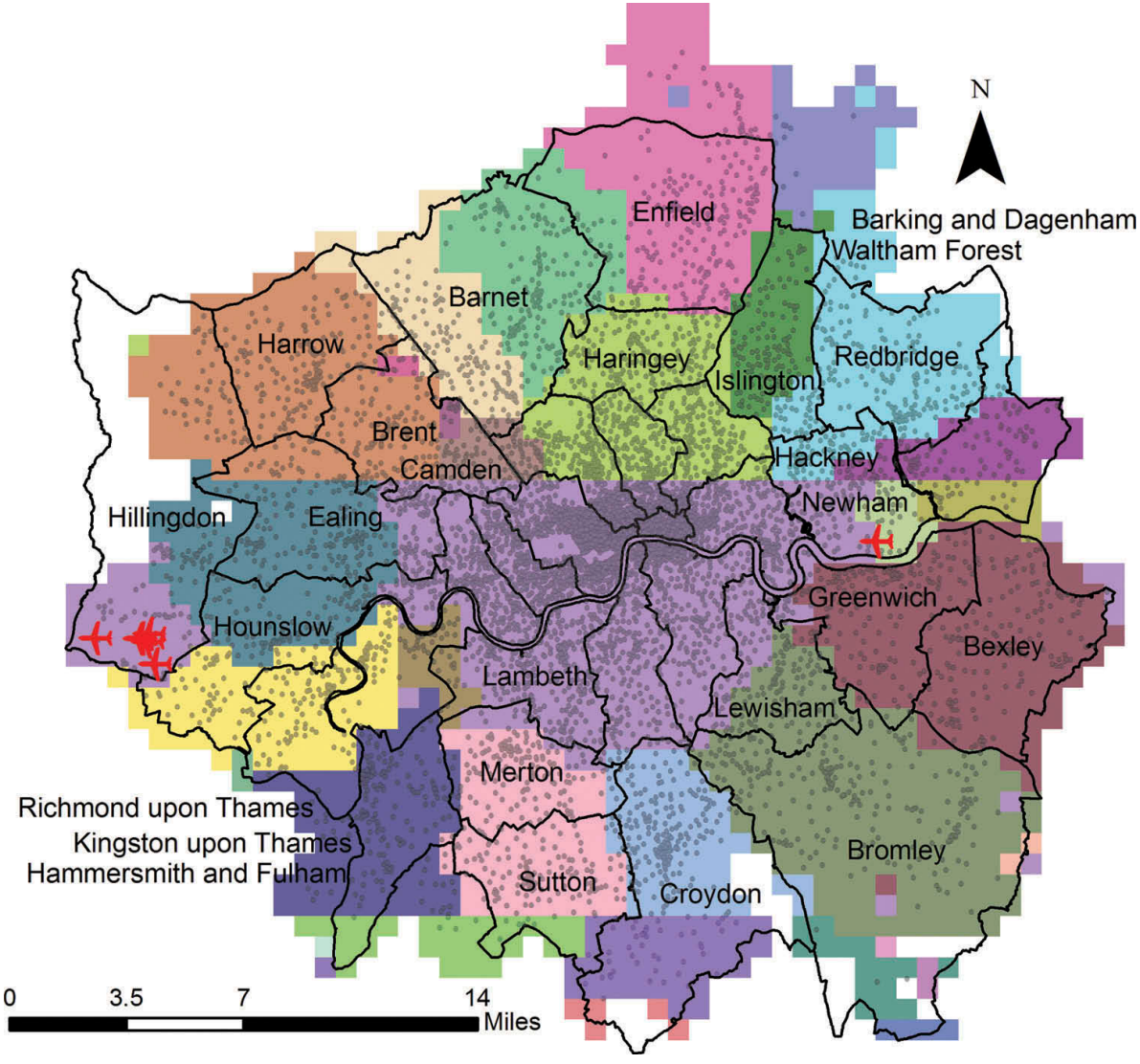


Figure 16: Inferred boundaries from collective Twitter user displacements in London (Yin *et al.* 2017)

LBSN research applications (2)

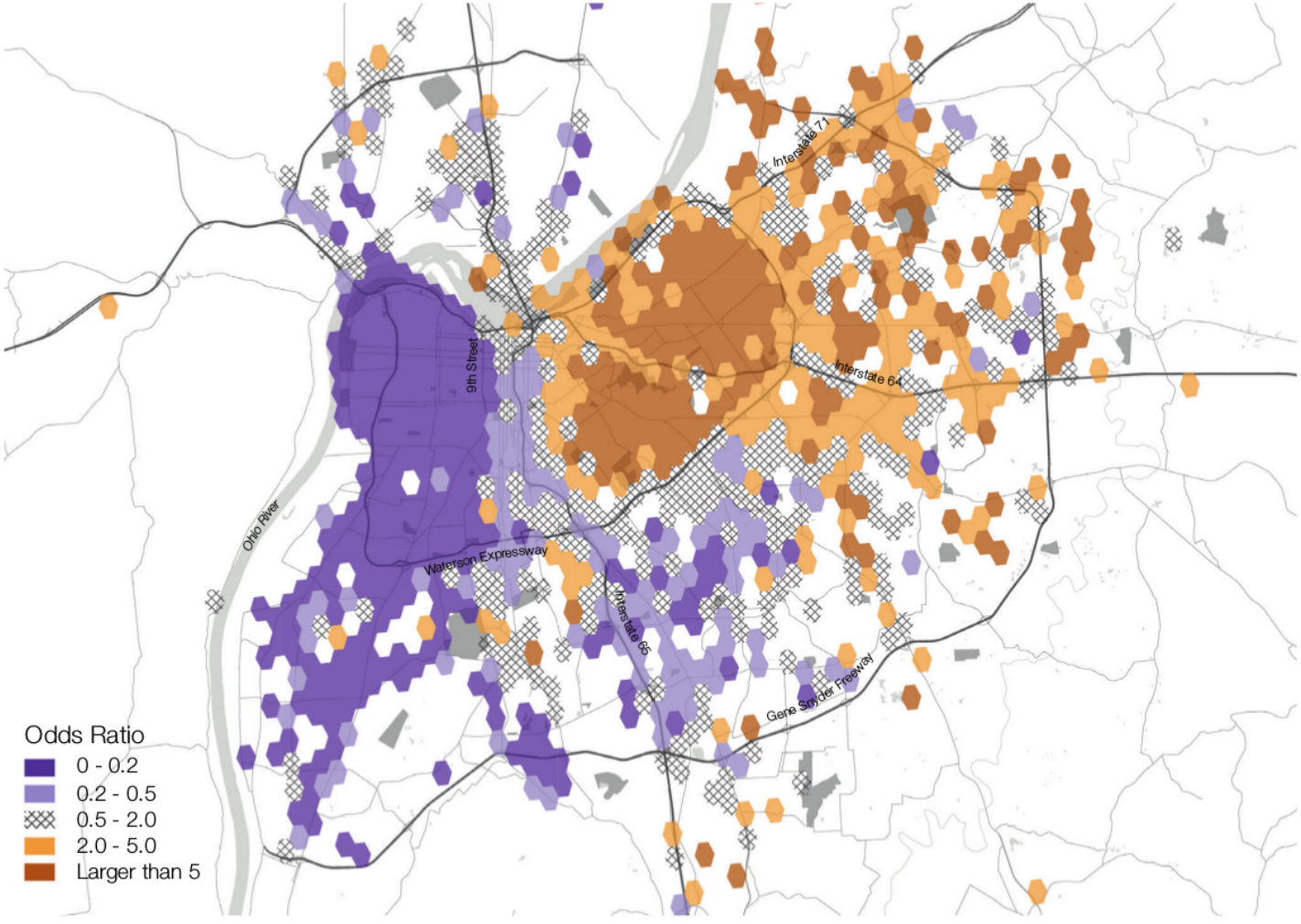


Figure 17: Unevenly segregated activity spaces of West End and East End residents in Louisville, Kentucky (Shelton *et al.* 2015)

LBSN research applications (3)

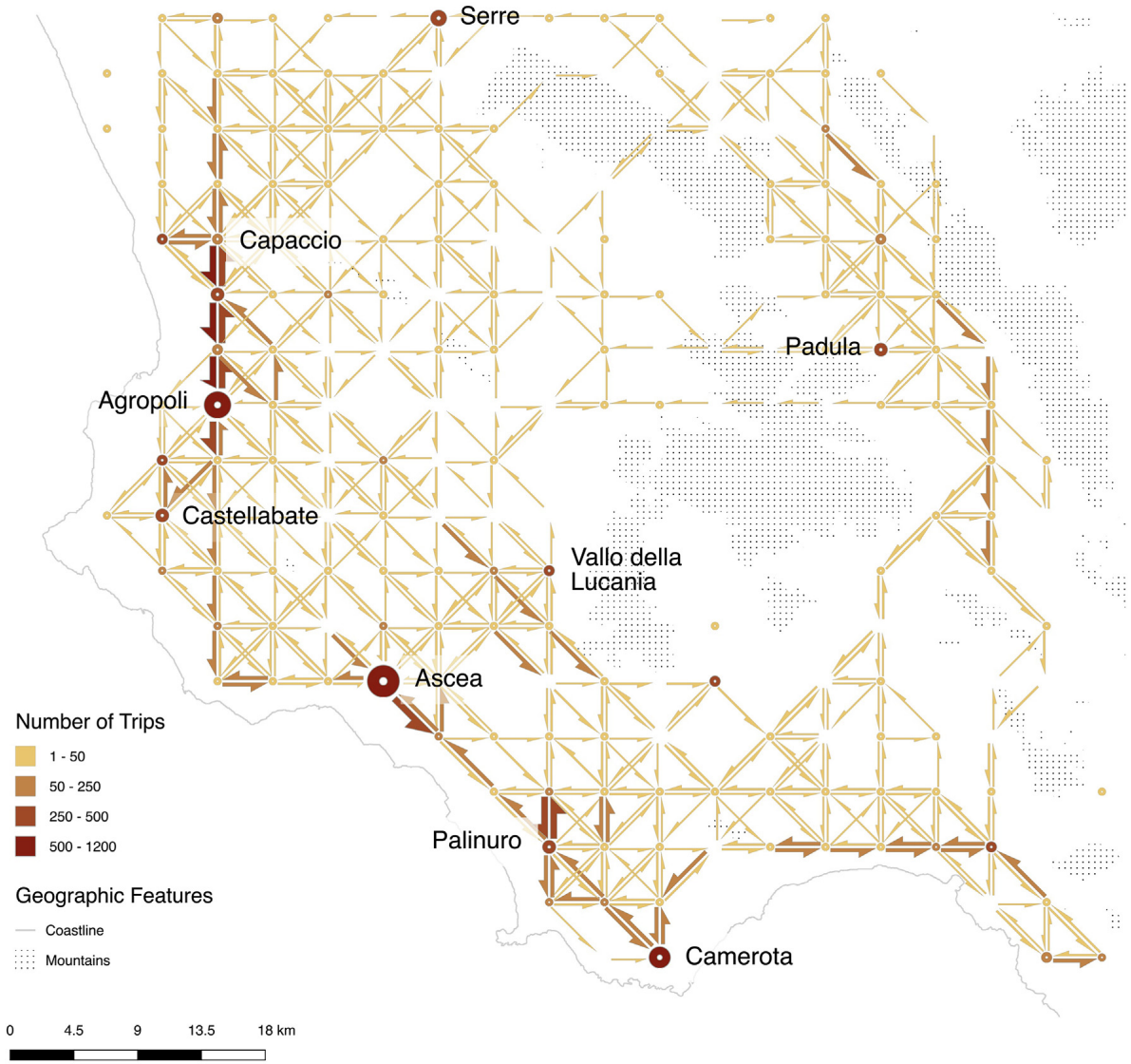


Figure 18: Flows derived from 3135 individuals who posted at least one tweet in Cilento (Chua *et al.* 2016)

Example 4: Collecting network data

- Running a Python script to **construct a Twitter social network**
- Given **starting** point `s`, travelling until entire network is covered up to given **depth** `d`
- The script uses the `tweepy` **library** to access the Twitter API

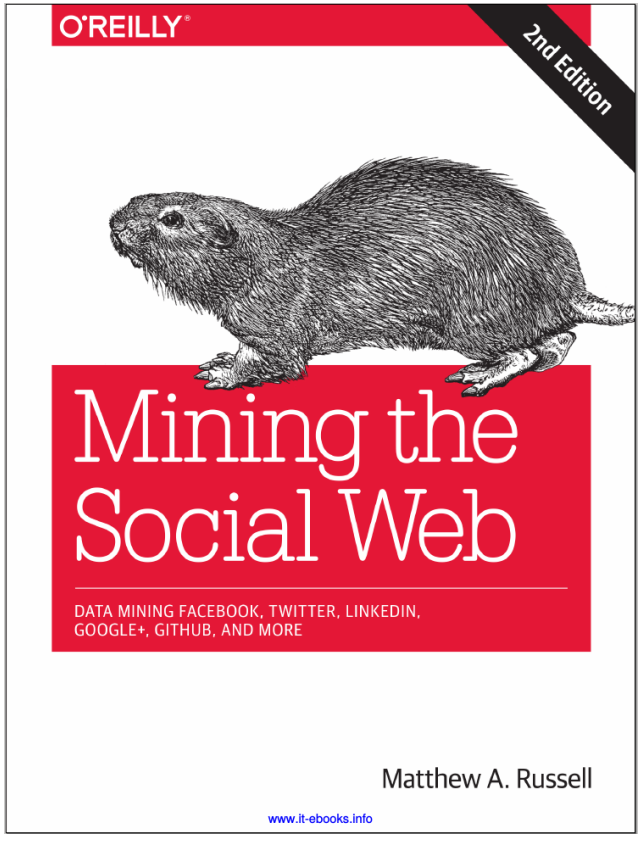


Figure 19: Russell (2013)

```
python get_followers.py -s MichaelDorman84 -d 2
```

Example 4: Collecting network data

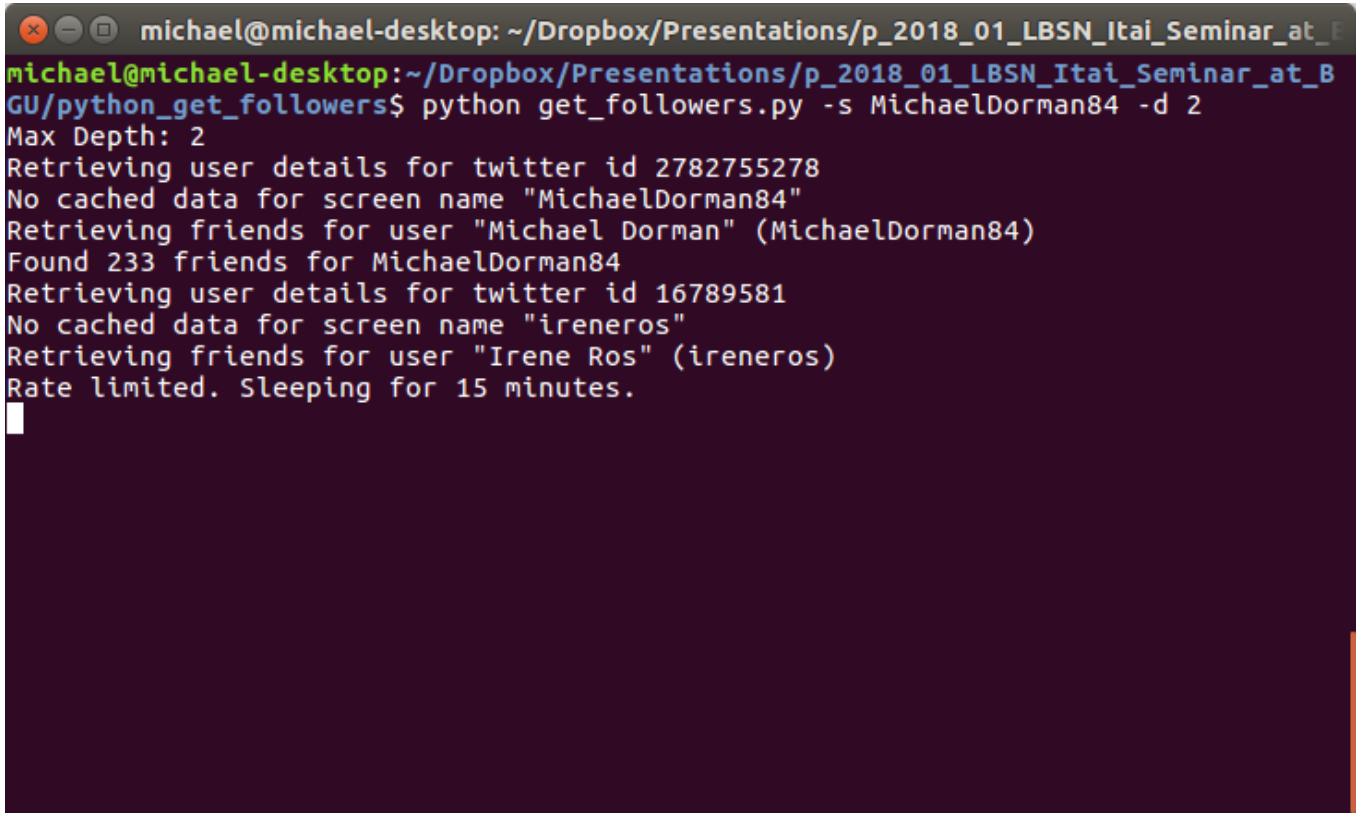


Figure 20: Python script for reconstructing social network of given depth

Example 4: Collecting network data

- The resulting folder of user-metadata processed **into a CSV file** with another Python script -

```
python twitter_network.py
```

- The **processed CSV** file can be read into R with `read.csv` -

```
friends = read.csv(
  "network.csv",
  sep = "\t",
  header = FALSE,
  stringsAsFactors = FALSE
)
```

Example 5: Network analysis

- The table consists of an **edge list** -
  - First column is the **follower**
  - Second column is the **friend**

```
head(friends)
##                   V1             V2   V3
## 1 MichaelDorman84      ireneros   28
## 2         ireneros  fdo_becerra 6143
## 3         ireneros KAUST_Vislab 6143
## 4         ireneros NYCaptionBot 6143
## 5         ireneros beesandbombs 6143
## 6         ireneros     vega_vis 6143
```

```
dim(friends)
## [1] 86858     3
```

Example 5: Network analysis

- We can remove users for whom we have no **friend** data -

```
friends = friends[friends$V2 %in% friends$V1, ]
```

```
dim(friends)
## [1] 3835    3
```

Example 5: Network analysis

- The python script also produces user details, including **self-reported location** -

```
locations = read.csv(
  "locations.csv",
  stringsAsFactors = FALSE
)
```

```
head(locations)
##                name                location
## 1 d3visualization                Montreal
## 2         sckottie                      OR
## 3         mdsumner     Hobart, Tasmania
## 4 spatialanalysis                  London
## 5 precariobecario                Albacete
## 6       dinomirMT Melbourne, Victoria
```

Example 5: Network analysis

- We can use the **Google Maps Geocode API** to convert location text to coordinates
- Geocode with package `mapsapi` -

```
library(mapsapi)

gc = mp_geocode(locations$location, key = "AIza...")
gcp = mp_get_points(gc)
```

- Or load the **pre-calculated** result -

```
gcp = st_read(
  "gcp.shp",
  stringsAsFactors = FALSE
)
```

Example 5: Network analysis

```
head(gcp[, "address"])
## Simple feature collection with 6 features and 1 field
## geometry type:  POINT
## dimension:      XY
## bbox:           xmin: -111.6946 ymin: -42.88214 xmax: 1
## epsg (SRID):    4326
## proj4string:    +proj=longlat +datum=WGS84 +no_defs
##                 address                    geometry
## 1              Montreal  POINT (-73.56726 45.50169)
## 2                    OR   POINT (-111.6946 40.2969)
## 3    Hobart, Tasmania  POINT (147.3272 -42.88214)
## 4                London POINT (-0.1277583 51.50735)
## 5              Albacete  POINT (-1.858542 38.99435)
## 6 Melbourne, Victoria  POINT (144.9631 -37.81363)
```

Example 5: Network analysis

- To get the **country name** for each **user location**, we can use the world borders polygonal layer from package `rworldmap` -

```
library(rworldmap)

world = getMap(resolution = "li")
world = st_as_sf(world)
```

- Plot -

```
plot(st_geometry(world), border = "grey")
plot(st_geometry(world[gcp, ]), col = "grey", add = TRUE)
plot(st_geometry(gcp), add = TRUE, col = "red")
```

Example 5: Network analysis

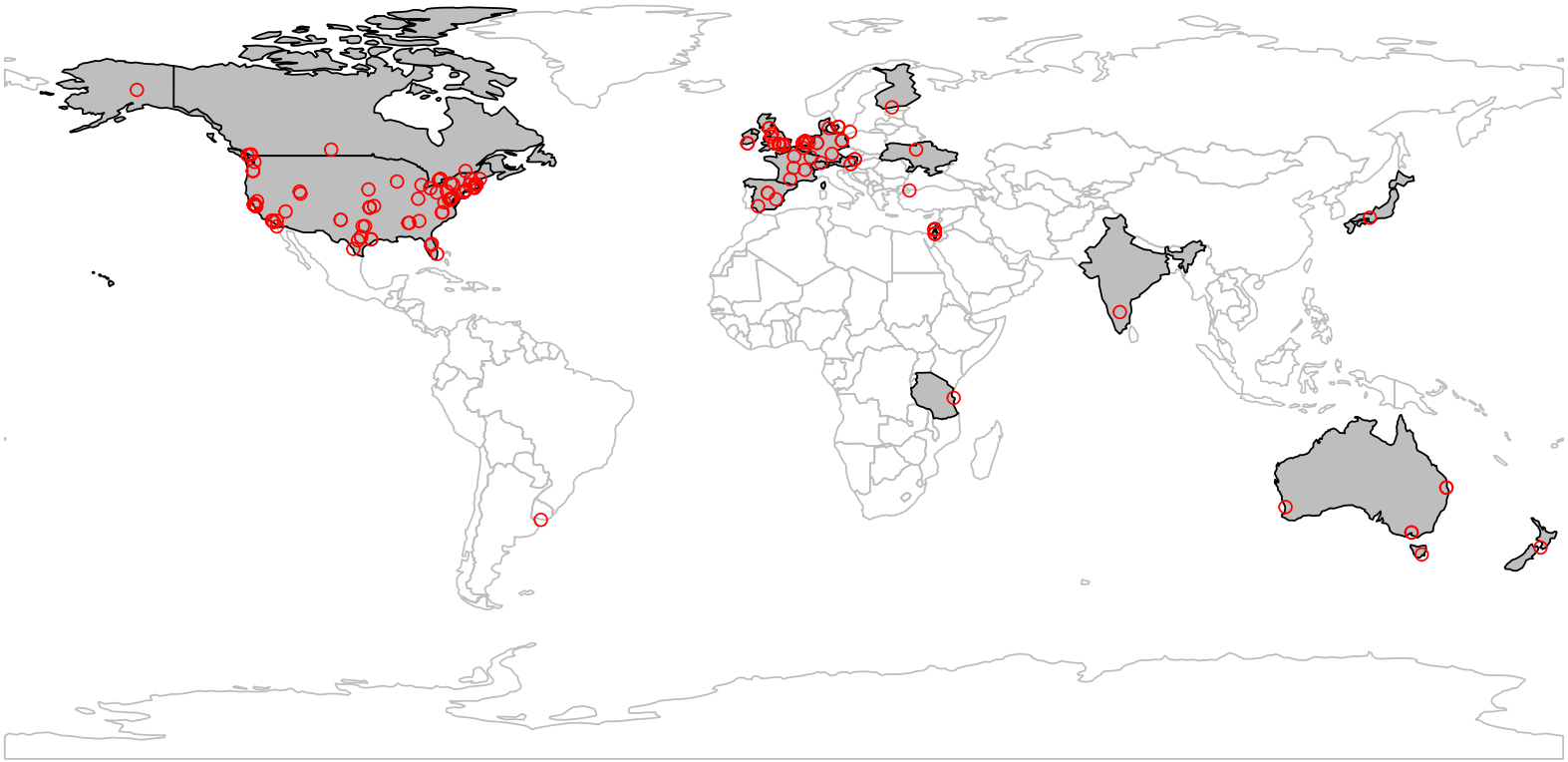


Figure 21: Twitter user home locations

Example 5: Network analysis

- We add the country each user location falls in with a **spatial join** -

```
locations$NAME = st_join(gcp, world)$NAME
```

- The resulting table specifies **the geocoded country** where each user lives -

```
head(locations)
##                name              location           NAME
## 1 d3visualization              Montreal         Canada
## 2         sckottie                    OR  United States
## 3         mdsumner     Hobart, Tasmania      Australia
## 4 spatialanalysis                London United Kingdom
## 5 precariobecario              Albacete          Spain
## 6        dinomirMT Melbourne, Victoria      Australia
```

Example 5: Network analysis

- We can now *replace* all **user names** with their **country name** -

```
friends$from =
  locations$NAME[match(friends$V1, locations$name)]
friends$to =
  locations$NAME[match(friends$V2, locations$name)]
```

- Then subset just the **country columns** -

```
friends = friends[, c("from", "to")]
```

- And remove **missing values** -

```
friends = friends[complete.cases(friends), ]
```

Example 5: Network analysis

- The result is a **country-to-country** edge list
- To benefit from methods for **visualization** and **analysis** of networks we need to convert this table to a **network object**

```
head(friends)
##                from             to
## 1           Israel United States
## 20 United States United States
## 24 United States United States
## 28 United States         France
## 34 United States         Canada
## 81 United States United States
```

```
dim(friends)
## [1] 3144    2
```

Example 5: Network analysis

- The edge list table can be converted to an `igraph` object, so that we can use **network analysis** methods -

```
library(igraph)

g = graph_from_data_frame(friends)
```

- The network can then be **aggregated** by assigning an **edge weigth of 1** for each follower relation, and **summing** -

```
E(g)$weight = 1
g = simplify(
  graph = g,
  remove.loops = FALSE,
  edge.attr.comb = "sum"
)
```

Example 5: Network analysis

```
g
## IGRAPH 9fe7797 DNW- 20 215 --
## + attr: name (v/c), weight (e/n)
## + edges from 9fe7797 (vertex names):
##  [1] Israel->Israel
##  [2] Israel->United States
##  [3] Israel->Canada
##  [4] Israel->Netherlands
##  [5] Israel->France
##  [6] Israel->United Kingdom
##  [7] Israel->Ireland
##  [8] Israel->Austria
## + ... omitted several edges
```

Example 5: Network analysis

- Graph **type** and **size**
  - Directed?
  - Weighted?
  - Vertex (node) count
  - Edge (link) count

```
is.directed(g)
## [1] TRUE
is.weighted(g)
## [1] TRUE
vcount(g)
## [1] 20
ecount(g)
## [1] 215
```

Example 5: Network analysis

- Vertices and edges can have **attributes**
- In our case, the **vertices** have a `name` **attribute** -

```
V(g)[1:2]
## + 2/20 vertices, named, from 9fe7797:
## [1] Israel         United States

V(g)$name[1:2]
## [1] "Israel"         "United States"
```

- And the **edges** have a `weight` **attribute** -

```
E(g)[1:2]
## + 2/215 edges from 9fe7797 (vertex names):
## [1] Israel->Israel         Israel->United States

E(g)$weight[1:2]
## [1]  12 119
```

Example 5: Network analysis

- Graph **properties** -
  - **Density** is the ratio between the number of edges and the number of possible edges
  - **Components** is the number of sub-groups in the network
  - **Diameter** is the the longest of the shortest paths across all pairs of nodes

```
edge_density(g)
## [1] 0.5657895
```

```
count_components(g)
## [1] 1
```

```
diameter(g)
## [1] 5
```

Example 5: Network analysis

- `igraph` can be used to create any kind of **network plot**
- However, the **graphical parameters** need to be manually calculated

```
ew = E(g)$weight
plot(
  g,
  layout = layout_with_graphopt(g),
  edge.arrow.size = 0.5,
  edge.color =
    rgb(1, 0, 0, scales::rescale(log(ew), c(0.25, 1))),
  edge.curved = 0.1,
  edge.width = scales::rescale(log(ew), c(0.1, 5))
)
```

Example 5: Network analysis

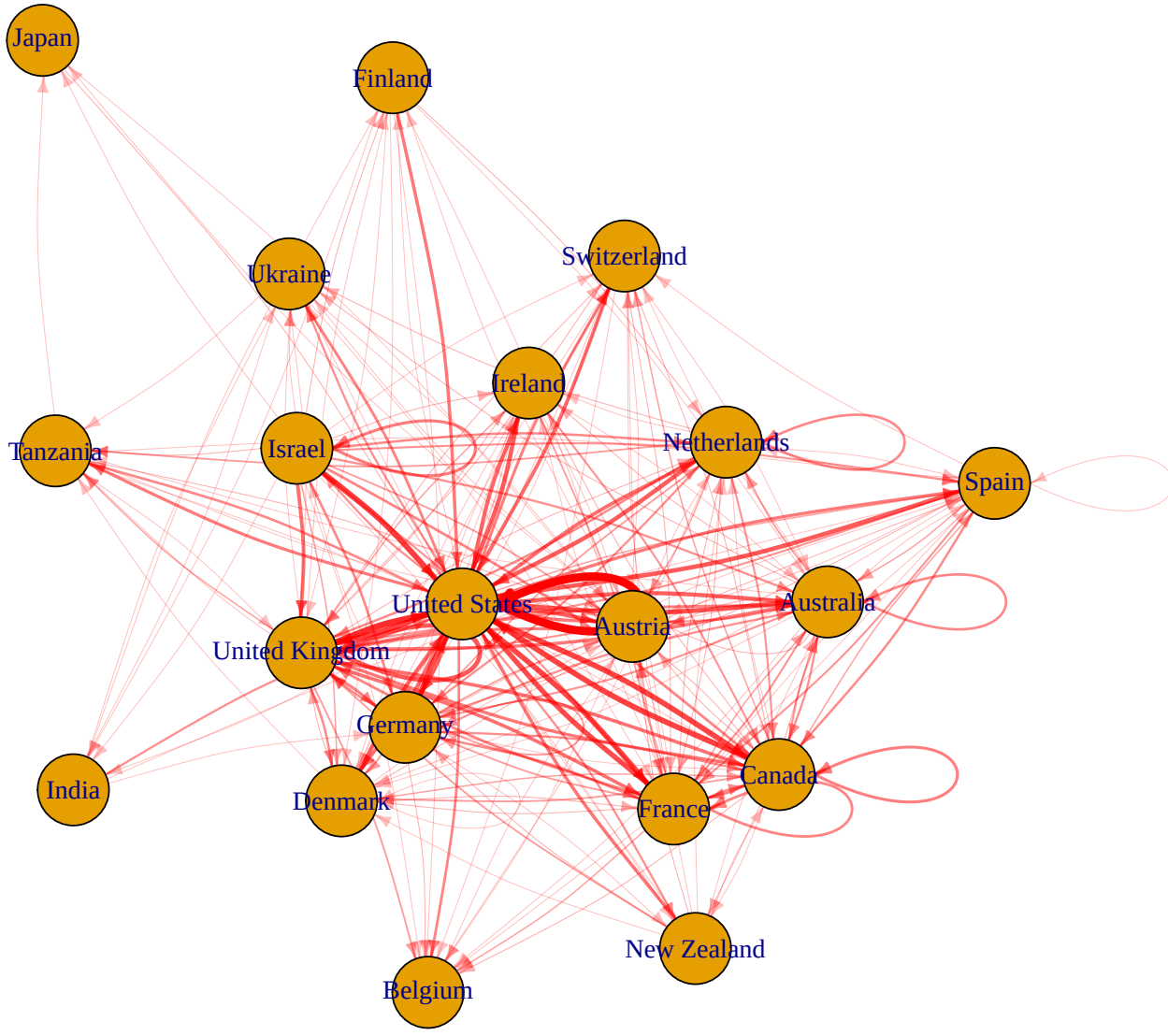


Figure 22: Between-country follower ties on Twitter

Example 5: Network analysis

- **Spatial** layout -

```
coords = st_coordinates(st_centroid(world))
coords = coords[match(V(g)$name, world$NAME), ]
ew = E(g)$weight
plot(st_geometry(world), border = "grey")
plot(
  g,
  layout = coords,
  edge.arrow.size = 0.5,
  edge.color =
    rgb(1, 0, 0, scales::rescale(log(ew), c(0.25, 1))),
  edge.curved = 0.2,
  edge.width = scales::rescale(log(ew), c(0.1, 5)),
  rescale = FALSE,
  add = TRUE
)
```

Example 5: Network analysis

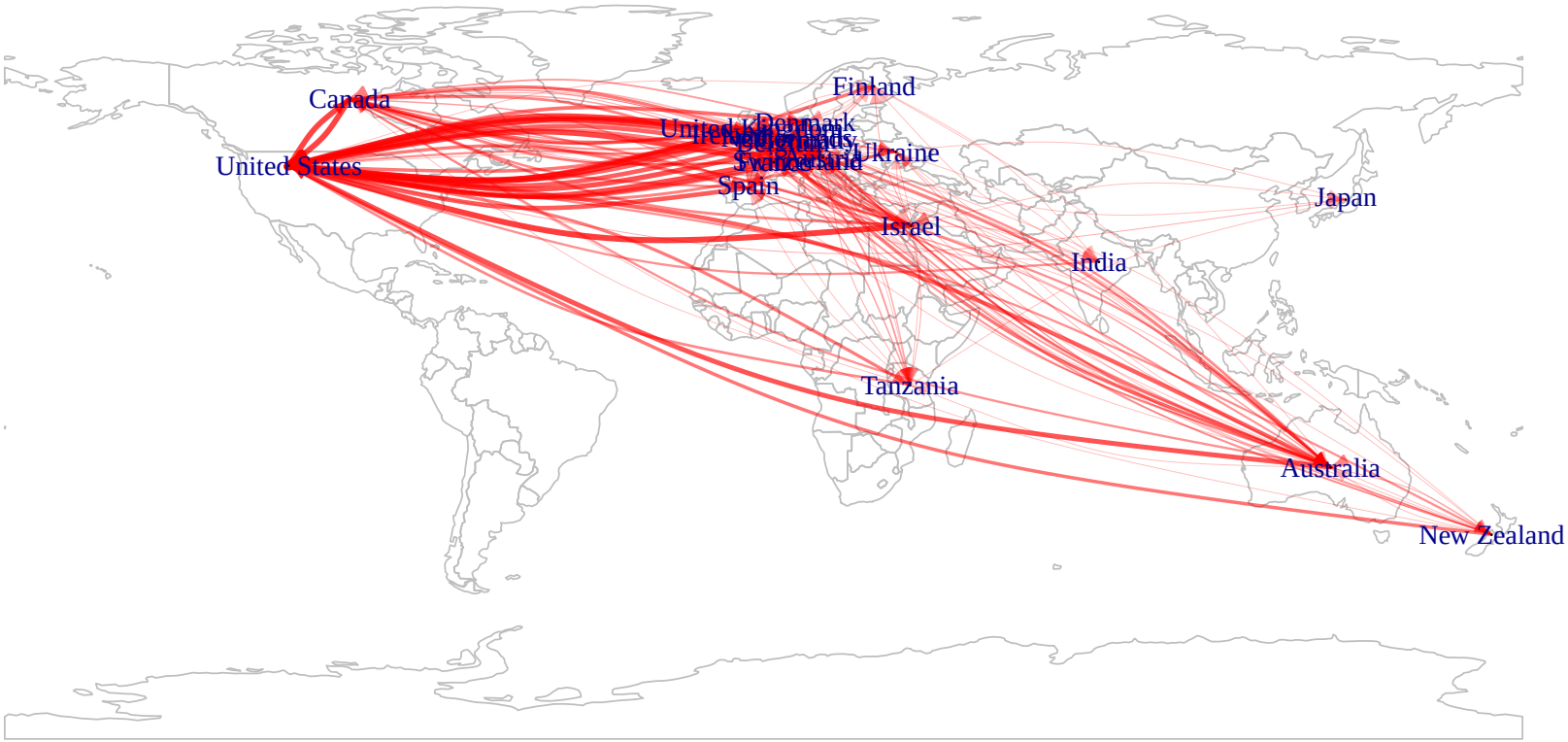


Figure 23: Between-country follower ties on Twitter

Example 5: Network analysis

- Vertex **degree** is its number of adjacent edges -

```
d = degree(g, mode = "total")
d
##          Israel   United States          Canada
##              23              38              29
##     Netherlands          France  United Kingdom
##              24              28              33
##         Ireland         Austria         Germany
##              17              30              29
##         Denmark       Australia           Spain
##              23              29              24
##         Belgium     New Zealand     Switzerland
##              13              15              19
##        Tanzania         Ukraine         Finland
##              15              14              13
##           India           Japan
##               9               5
```

Example 5: Network analysis

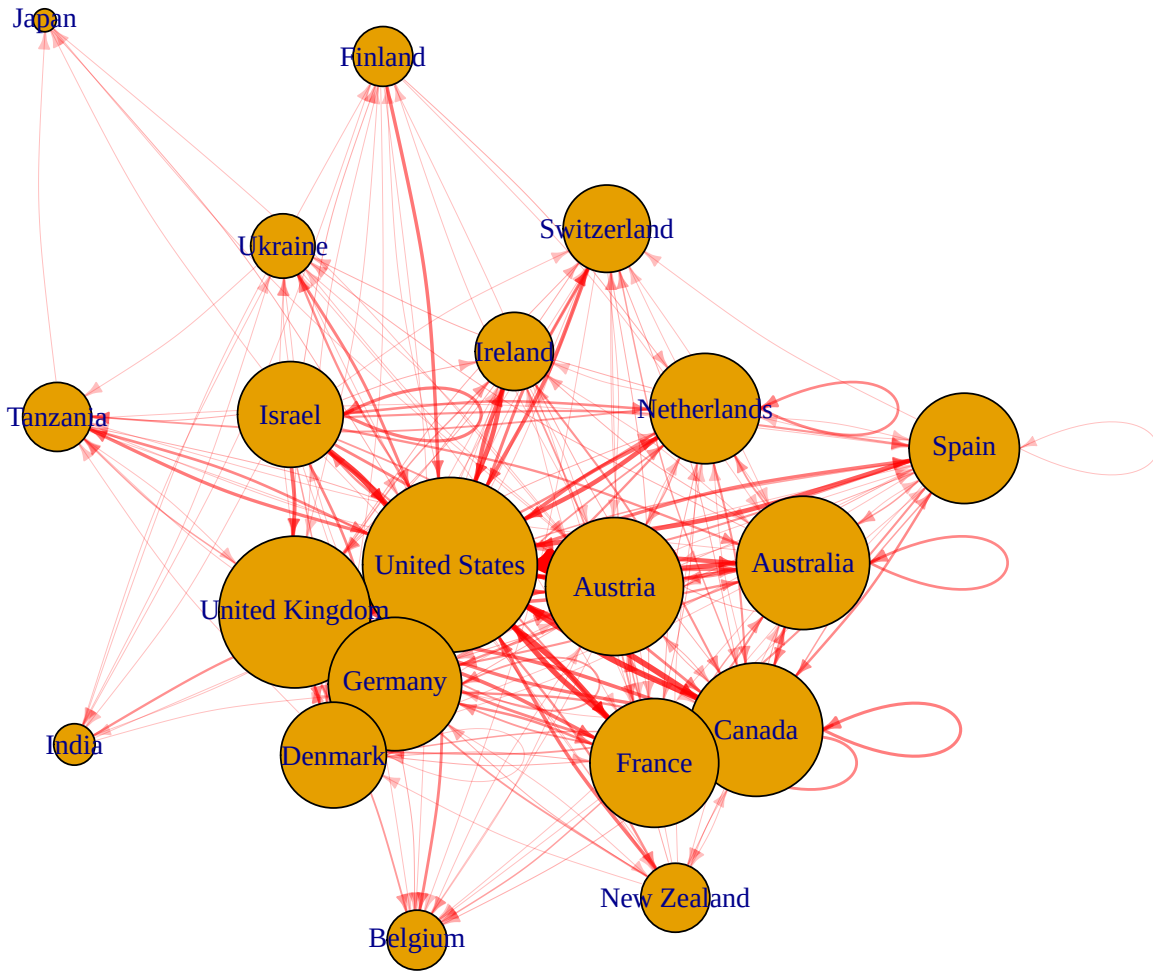


Figure 24: Number of edges per node

Example 5: Network analysis

- **Community detection** algorithms aim at **identifying sub-groups** in a network
- A sub-group is a set of nodes that has a relatively large number of **internal ties**, and also relatively few ties from the group **to other parts** of the network

```
cfg = cluster_spinglass(g)
```

- Plot -

```
plot(
  cfg, g,
  edge.arrow.size = 0.5,
  edge.width = 0.5
)
```

Example 5: Network analysis

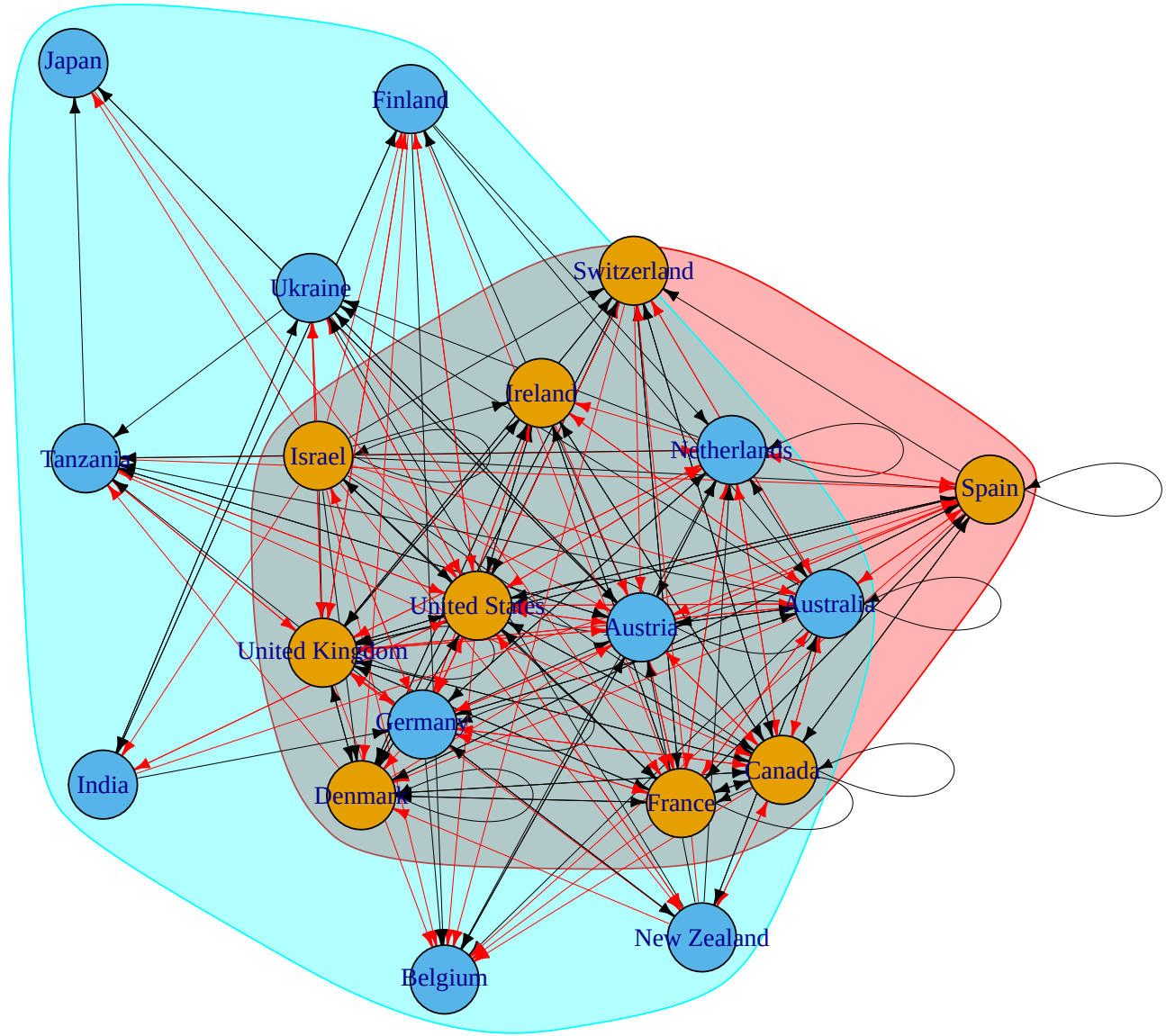


Figure 25: Detected communities

LBSN research applications (4)

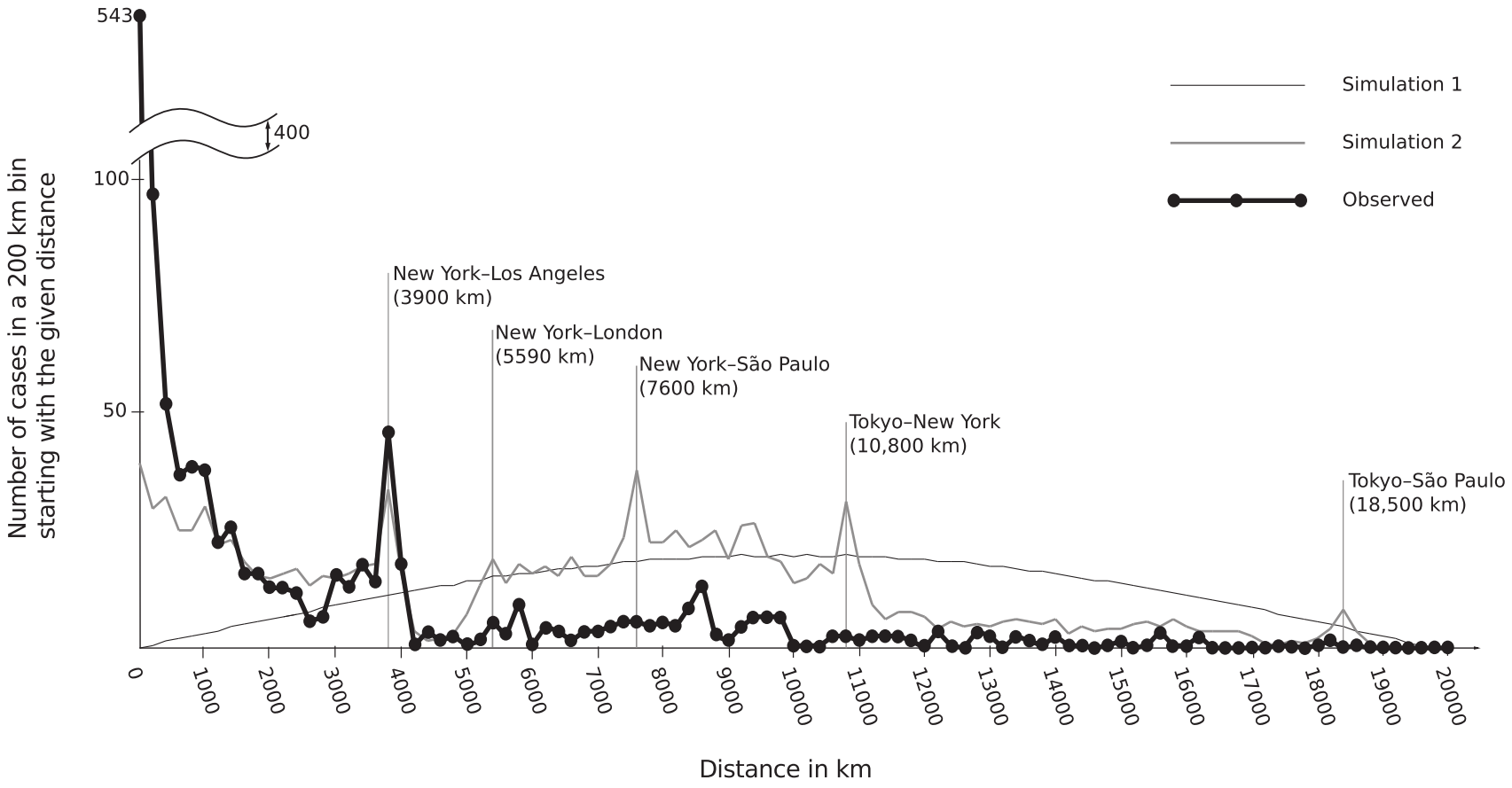


Figure 26: Histogram of physical distances between connected users (Takhteyev *et al.* 2012)

LBSN research applications (5)

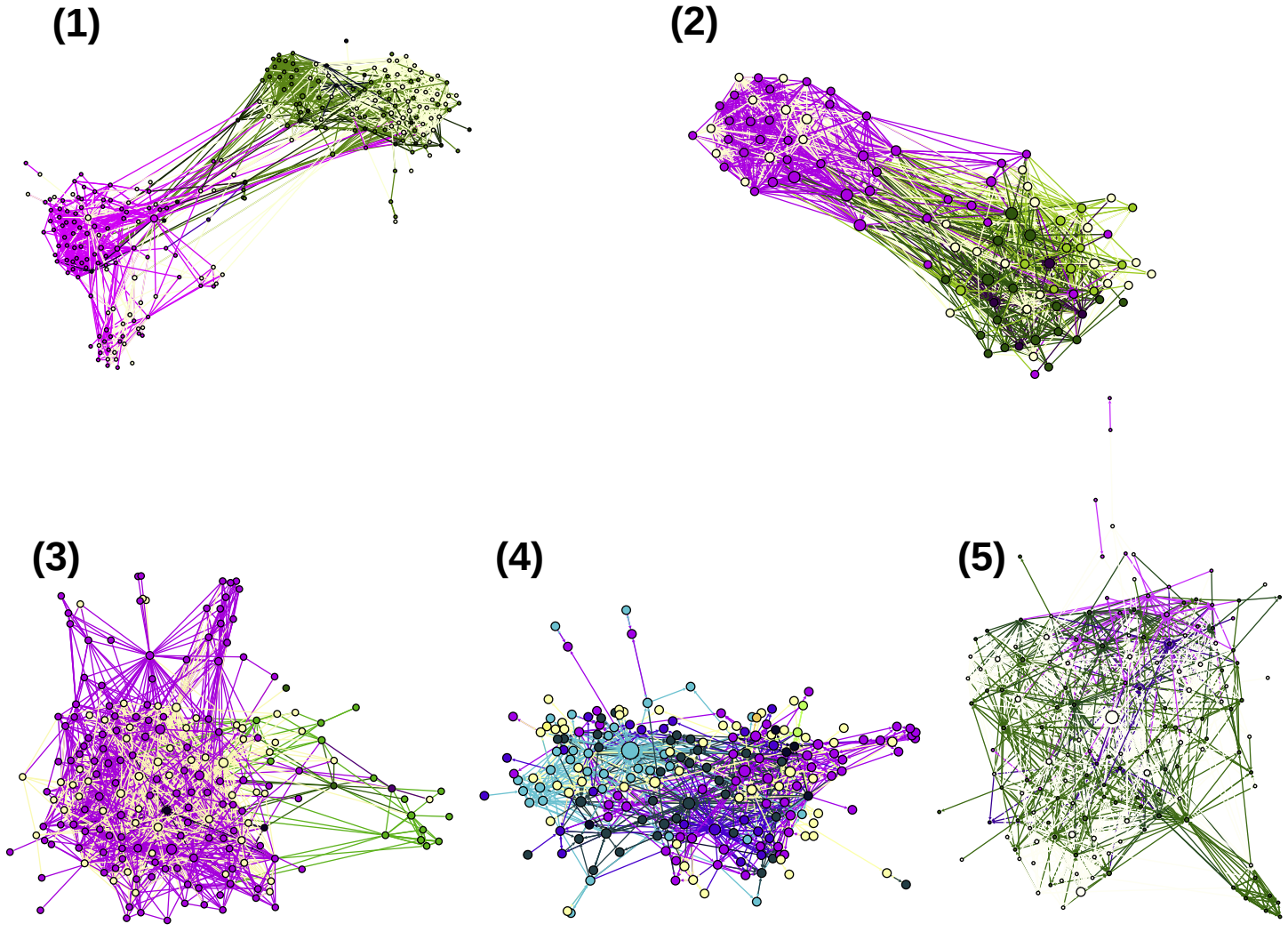


Figure 27: Examples ot five multilingual Twitter user network types, where English is integrated with (1) French, (2) Japanese, (3) Portuguese, (4) Greek and (5) Arabic (Eleta & Golbeck 2014)

Example 6: Sentiment analysis

- **Sentiment analysis** quantifies subjective information from text
- The most commonly used methods are based on dictionaries of **positive** and **negative** words
- For example, function `sentiment` from the `sentimentr` package -

```
library(sentimentr)

text = c(
  "I'm happy", "I'm sad...",
  "I'm very happy!", "I'm not happy"
)
sentiment(text)
##    element_id sentence_id word_count  sentiment
## 1:          1           1          2  0.5303301
## 2:          2           1          2 -0.3535534
## 3:          3           1          3  0.7794229
## 4:          4           1          3 -0.4330127
```

Example 6: Sentiment analysis

- Before calculating polarity on our tweets we need to keep only those in **English** -

```
pnt1 = pnt[pnt$lang == "en", ]
```

- Then we can calculate **polarity** -

```
s = sentiment(pnt1$text)
s = as.data.frame(s)
```

```
head(s)
##   element_id sentence_id word_count sentiment
## 1          1           1         19 0.1835326
## 2          2           1         13 0.2218801
## 3          3           1         12 0.2309401
## 4          4           1         13 0.2218801
## 5          5           1         21 0.3142338
## 6          6           1         22 0.2238608
```

Example 6: Sentiment analysis

- Sentiment is calculated **per sentence**
- In case we are interested in average sentiment **per text** (tweet), we can use `aggregate` -

```
s = aggregate(
  x = s[, "sentiment", drop = FALSE],
  by = s[, "element_id", drop = FALSE],
  FUN = mean
)
```

```
head(s)
##   element_id sentiment
## 1          1 0.1835326
## 2          2 0.2218801
## 3          3 0.2309401
## 4          4 0.2218801
## 5          5 0.3142338
## 6          6 0.2238608
```

Example 6: Sentiment analysis

- Four **most positive** tweets -

```
pos = order(s$sentiment, decreasing = TRUE)[1:4]
st_set_geometry(pnt1[pos, "text"], NULL)
##                                                    text
## 5234   Beyond beautiful beginning breathe it is...
## 5916   Holiday Party celebrating a hard-working...
## 55581  A continental style breakfast - pretty m...
## 2338  #insomniacsvirtue strikes again \nUP AND ...
```

Example 6: Sentiment analysis

- Four **most negative** tweets -

```
pos = order(s$sentiment)[1:4]
st_set_geometry(pnt1[pos, "text"], NULL)
##                                                    text
## 8147                              I'm really just tired...
## 2227 Chronic pain sufferers... it's ok to tak...
## 2725 5 Trust in the LORD with all your heart ...
## 2937 Accident, left lane blocked in #Methuen ...
```

Example 6: Sentiment analysis

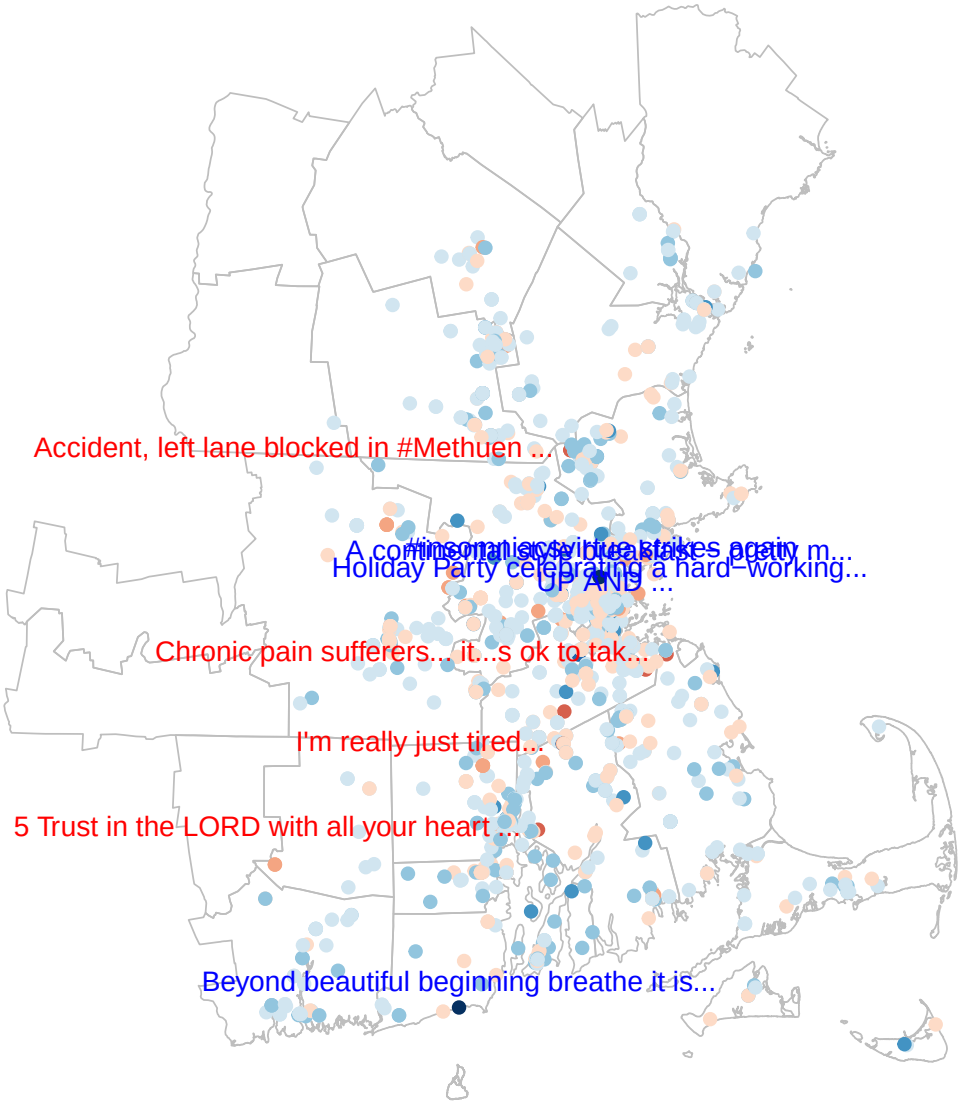


Figure 28: Sentiment score map

LBSN research applications (6)

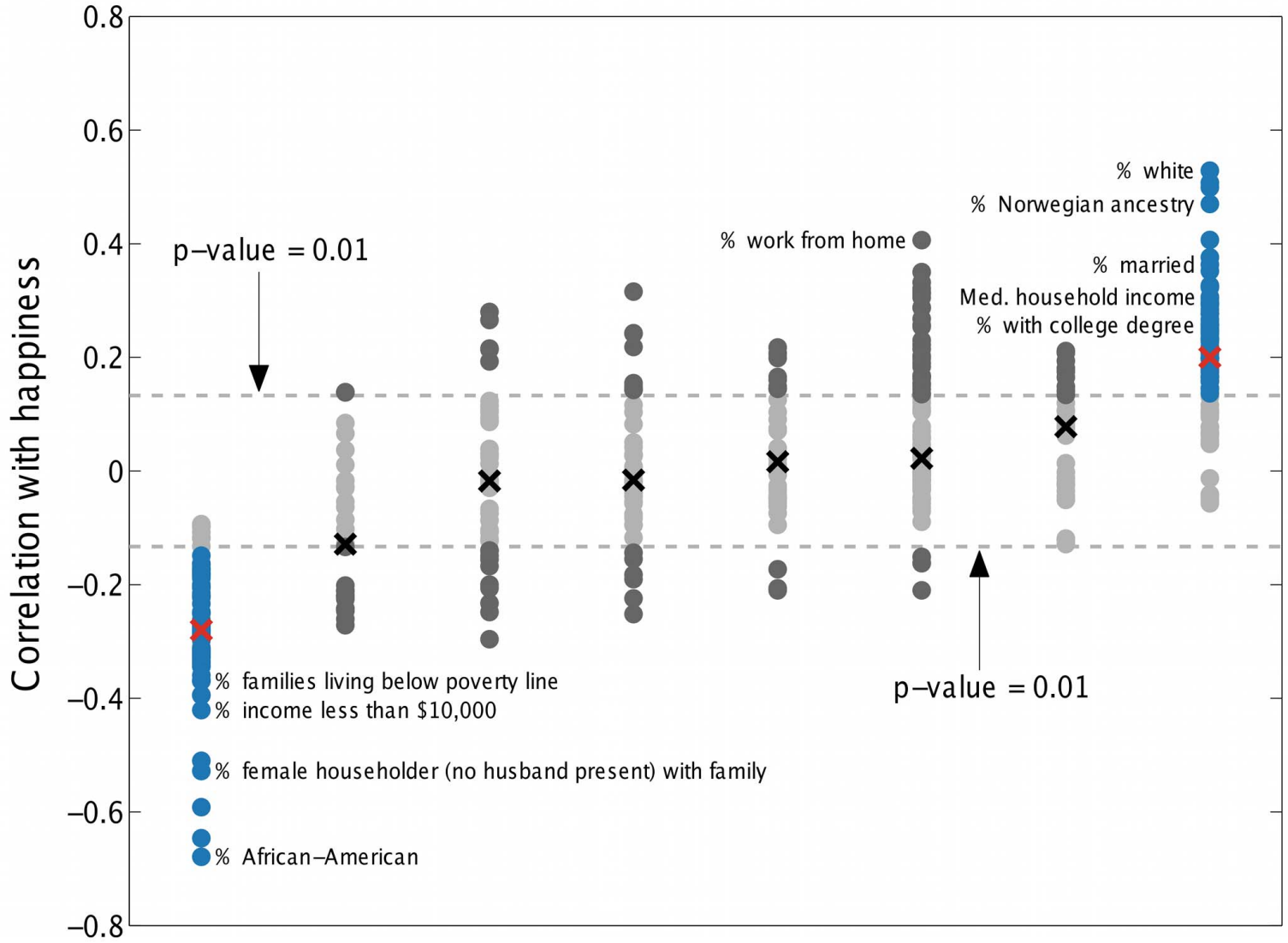


Figure 29: Spearman correlations for 432 demographic attributes with happiness (Mitchell *et al.* 2013)

LBSN research applications (7)

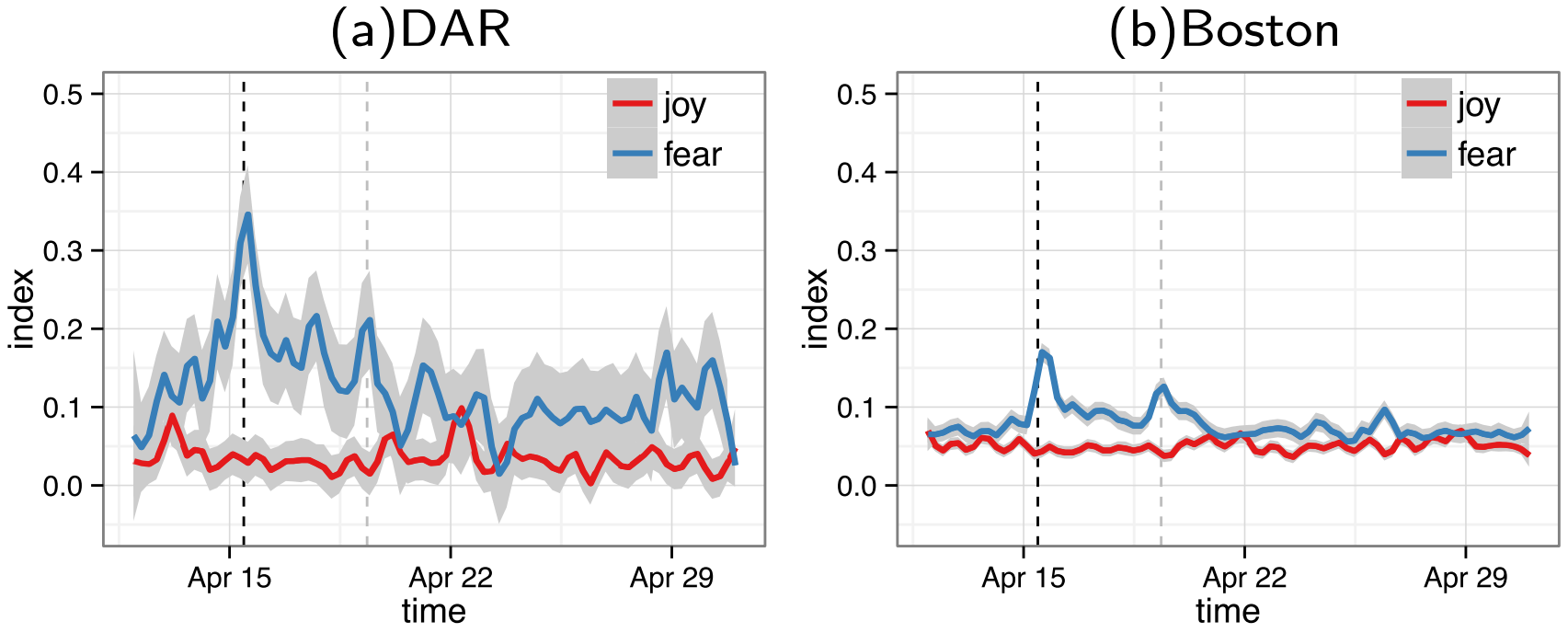


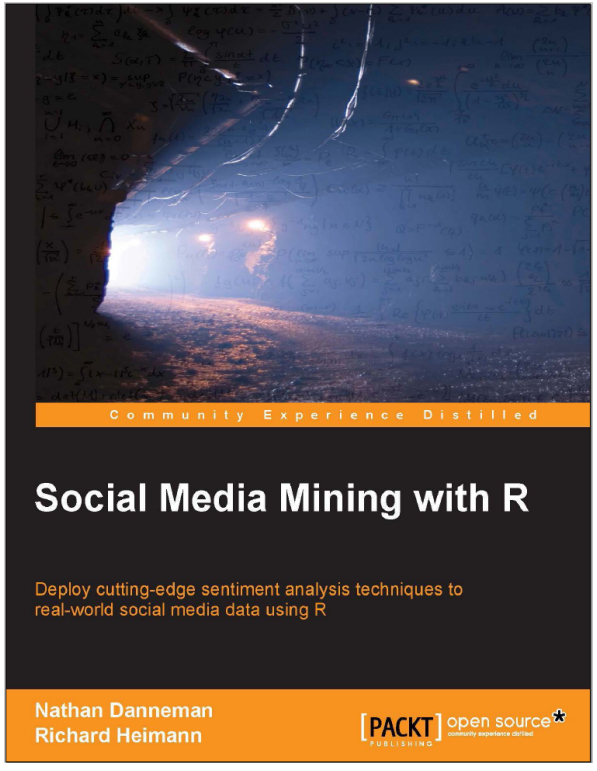
Figure 30: Sentiment indices over time in (a) the direct affected region (DAR) and (b) the City of Boston (Lin & Margolin 2014)

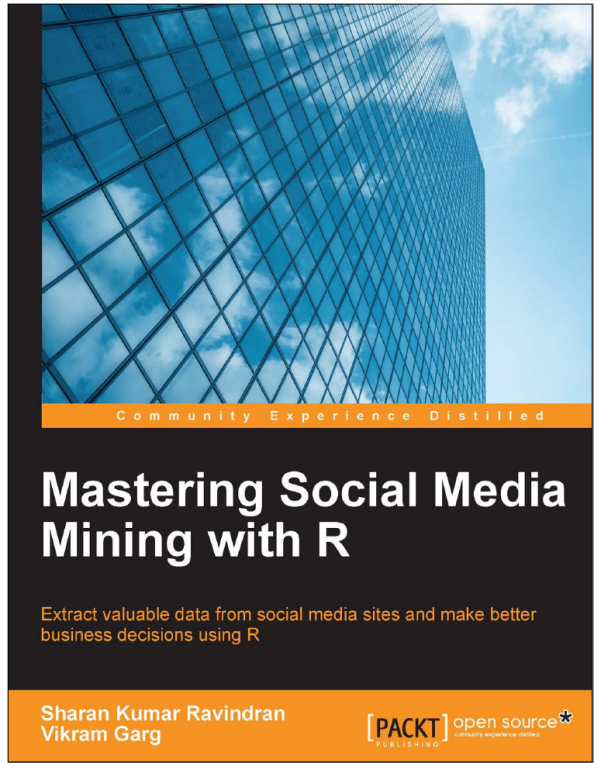
Summary - Software Tools

- Data collection
  - `rtweet` (R)
  - `tweepy` (Python)
- Spatial analysis
  - `sf` (R)
  - `mapsapi` (R)
  - `rworldmap` (R)
- Network analysis
  - `igraph` (R)
- Sentiment Analysis
  - `sentimentr` (R)

More information

- *Mining the Social Web*, 2nd Ed. (2013)
- *Social Media Mining with R* (2014)
- *Mastering Social Media Mining with R* (2015)





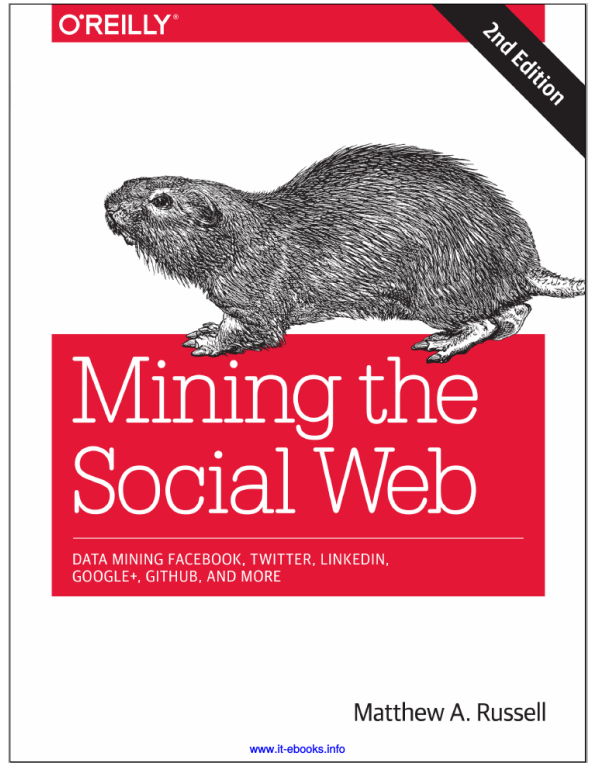


Figure 31: Books on Social Network Analysis

More information

- *Statistical Analysis of Network Data* (2014)
- *A User's Guide to Network Analysis in R* (2015)

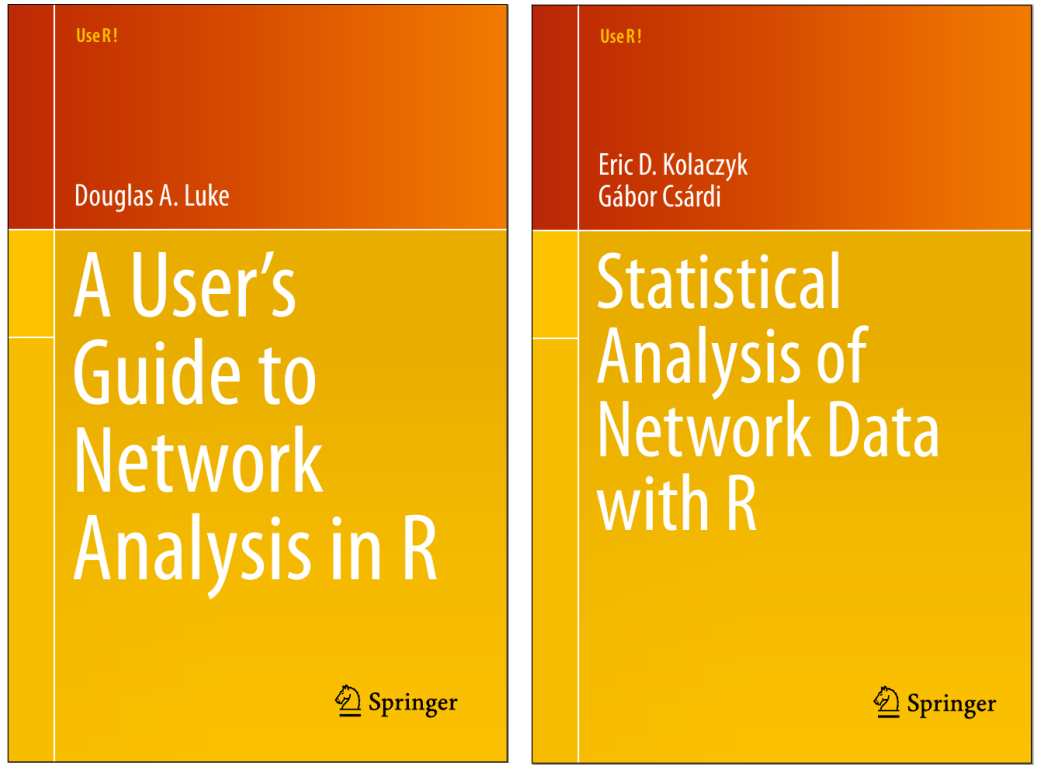


Figure 32: Books on Social Network Analysis

Thank you for listening!