NYU Politics Data Lab Workshop: Scraping Twitter and Web Data Using R

Pablo Barberá
Department of Politics
New York University
email: pablo.barbera@nyu.edu
twitter: @p_barbera

March 26, 2013
Scraping the web: what? why?

- Increasing amount of data is available on the web:
  - Election results, budget allocations, legislative speeches
  - Social media data, newspapers articles
  - Geographic information, weather forecasts, sports scores

- These data are provided in an unstructured format: you can always copy & paste, but it’s time-consuming and prone to errors.

- Web scraping is the process of extracting this information automatically and transform it into a structured dataset.

- Two different scenarios:
  1. Web APIs (application programming interface): website offers a set of structured http requests that return JSON or XML files.
  2. Screen scraping: extract data from source code of website, with html parser (easy) or regular expression matching (less easy).

- Why R? It includes all tools necessary to do web scraping, familiarity, direct analysis of data... But python, perl, java are probably more efficient tools. Whatever works for you!
The rules of the game

1. Respect the hosting site’s wishes:
   - Check if an API exists first, or if data are available for download.
   - Some websites “disallow” scrapers on their robots.txt files.

2. Limit your bandwidth use:
   - Wait one second after each hit
   - Try to scrape websites during off-peak hours
   - Scrape only what you need, and just once

3. When using APIs, read terms and conditions.
   - The fact that you can access some data doesn’t mean you should use it for your research.
   - Be aware of rate limits.
   - Ongoing debate on replication of social science research using this source of data.
Outline

- Rest of the workshop: learning with four toy examples.
- Downloading data using Web APIs
  1. Finding influential users using Twitter’s REST API
  2. Capturing and analyzing tweets in realtime using the Streaming API
- Screen scraping of HTML websites
  4. Extracting district-level electoral results in Georgia
  5. Constructing a dataset of bribes paid in India
- Code and data: http://www.pablobarbera.com/workshop.zip
- Fork my repo! http://github.com/pablobarbera/workshop
Introduction to the Twitter API

Why Twitter?
- 140M active Twitter users. 16% online Americans use it.
- Meaningful public conversations; use for political purposes
- Cheap, fast, easy access, contextual information

There are two ways of getting Twitter data:

1. RESTful API:
   - Queries for specific information about users and tweet
   - Examples: user profile, list of followers and friends, tweets generated by a given user, users lists...
   - R library: twitteR

2. Streaming API:
   - Connect to the “stream” of tweets as they are being published
   - Examples: random sample of all tweets, tweets that mention a keyword, tweets from a set of users...
   - R library: streamR

3. More: dev.twitter.com/docs/api/1.1
Most APIs require authentication to limit number of hits per user.

Twitter (and many others) use an open standard called OAuth 1.0, which allows connections without sharing username and password.

Currently all queries to Twitter’s API require a valid OAuth “token”.

How to get yours:

1. Create new application on dev.twitter.com
2. Save consumer key and consumer secret
3. Go to 01_getting_OAuth_token.R and run the code.
4. Save token for future sessions.
1. Finding influential users in a small network

- Twitter as a directed network, where “edges” are following relationships across users
- Common (strong) assumption: more followers implies more influence.
- Who is the most influential Twitter user at the NYU Politics Department?
- We will learn how to:
  1. Download user information
  2. Extract list of followers/friends
  3. Apply snowball sampling to construct dept. network
  4. Quick visualization of the network
- Code: 02_analysis_twitter_nyu.R
# getting data for seed user
seed <- getUser("drewconway")
seed.n <- seed$screenName

## [1] "drewconway"

# saving list of Twitter users he follows
following <- seed$getFriends()
following.n <- as.character(lapply(following, function(x) x$getScreenName()))
head(following.n)

## [1] "MikeGruz" "johnjhorton" "anthlittle" "theumpires"
## [5] "JennyVrentas" "dturkenk"

# creating list to be filled with friends for each NYU user
follow.list <- list()
follow.list[[seed.n]] <- following.n
# extracting description of users

descriptions <- as.character(lapply(following, function(x) x$getDescription()))
descriptions[1]

## [1] "Political science Ph.D student, lover of fine booze and shenanigans, courage under fire."

# function to subset only users from NYU-Politics

extract.nyu <- function(descriptions){
  nyu <- grep("nyu|new york university", descriptions, ignore.case = T)
poli <- grep("poli(tics|tical|sci)", descriptions, ignore.case = T)
others <- grep("policy|wagner|cooperation", descriptions, ignore.case = T)
  nyu.poli <- intersect(nyu, poli)
  nyu.poli <- nyu.poli[nyu.poli %in% others == FALSE]
  return(nyu.poli)
}

# and now subsetting Twitter users from NYU-Politics

nyu <- extract.nyu(descriptions)
nyu.users <- c(seed$screenName, following.n[nyu], "cdsamii")
nyu.users

## [1] "drewconway" "p_barbera" "griverorz" "j_a_tucker"

## [5] "pfernandezvz" "LindseyCormack" "cdsamii"
# loop over NYU users following same steps

```r
while (length(nyu.users) > length(follow.list)) {
    # pick first user not done
    user <- nyu.users[nyu.users %in% names(follow.list) == FALSE][1]
    user <- getUser(user)
    user.n <- user$screenName
    # download list of users he/she follows
    following <- user$getFriends()
    friends <- as.character(lapply(following, function(x) x$getScreenName()))
    follow.list[[user.n]] <- friends
    descriptions <- as.character(lapply(following, function(x) x getDescription()))
    # subset and add users from NYU Politics
    nyu <- extract.nyu(descriptions)
    new.users <- lapply(following[nyu], function(x) x$getScreenName())
    new.users <- as.character(new.users)
    nyu.users <- unique(c(nyu.users, new.users))
    # if rate limit is hit, wait for a minute
    limit <- getCurRateLimitInfo()[44, 3]
    while (limit == "0") {
        Sys.sleep(60)
        limit <- getCurRateLimitInfo()[44, 3]
    }
}
```
nyu.users <- names(follow.list)

# for each user, find which NYU users follow
adjMatrix <- lapply(follow.list, function(x) (nyu.users %in% x) * 1)

# transform into an adjacency matrix
adjMatrix <- matrix(unlist(adjMatrix), nrow = length(nyu.users), byrow = TRUE, 
                      dimnames = list(nyu.users, nyu.users))
adjMatrix[1:5, 1:5]

## drewconway cdsamii p_barbera griverorz j_a_tucker
## drewconway 0 1 1 1 1
## cdsamii 1 0 0 0 1
## p_barbera 1 0 0 1 1
## griverorz 1 0 1 0 0
## j_a_tucker 1 1 1 0 0
library(igraph)
network <- graph.adjacency(adjMatrix)
plot(network)
# computing indegree (followers in NYU dept)
degrees <- degree(network, mode="in")
degrees[1:5]

## drewconway  cdsamii  p_barbera  griverorz  j_a_tucker
## 10 9 10 5 12

# weigh label size by indegree
V(network)$label.cex <- (degrees/max(degrees)*1.25)+0.5

## choose layout that maximizes distances
set.seed(1234)
l <- layout.fruchterman.reingold.grid(network, niter=1000)

# draw nice network plot
pdf("network_nyu.pdf", width=6, height=6)
plot(network, layout=l, edge.width=1, edge.arrow.size=.25,
     vertex.label.color="black", vertex.shape="none",
     margin=-.15)

dev.off()

## pdf
## 2
Scraping Twitter and Web Data Using R

March 26, 2013
How to collect tweets

- For a quick analysis, you could use the `SearchTwitter` function in the `twitteR` package:

```r
searchTwitter("#PoliSciNSF", n = 1)
```

## [1]
"Jen_at_APSA: Is the Republican attack on political science self-defeating? | War of Ideas http://t.co/fvuuw8zMDF #PoliSciNSF"

- Code: 03_tweets_search.R
- Limitations:
  - Not all tweets are indexed or made available via search.
  - Does not contain user metadata
  - Limited to a few thousand most recent tweets
  - Old tweets are not available.
Streaming API

- Recommended method to collect tweets
- Firehose: real-time feed of all public tweets (400M tweets/day = 1 TB/day), but expensive.
- Spritzer: random 1% of all public tweets (4.5K tweets/minute = 8 GB/day), implemented in streamR as `sampleStream`
- Filter: public tweets filtered by keywords, geographic regions, or users, implemented as `filterStream`.
- Issues:
  - Filter streams have same rate limit as spritzer (1% of all tweets)
  - Stream connections tend to die spontaneously. Restart regularly.
  - Lots of invalid content in stream. If it can’t be parsed, drop it.
Anatomy of a tweet

```
{  "created_at":"Wed Nov 07 04:16:18 +0000 2012",
  "id":266031293945503744,
  "id_str":"266031293945503744",
  "text":"Four more years. http://t.co/bAJE6Vom",
  "source":"web",
  "user":  
  {  "id":813286,
    "id_str":"813286",
    "name":"Barack Obama",
    "screen_name":"BarackObama",
    "location":"Washington, DC",
    "url":"http://www.barackobama.com",
    "description":"This account is run by #Obama2012 campaign staff. Tweets from the President are signed -bo.",
    "protected":false,
    "followers_count":23487605,
    "friends_count":670339,
    "listed_count":182313,
    "created_at":"Mon Mar 05 22:08:25 +0000 2007",
    "utc_offset":-18000,
    "time_zone":"Eastern Time (US & Canada)",
    "geo_enabled":false,
    "verified":true,
    "statuses_count":7972,
    "lang":"en" },
  "geo":null,
  "coordinates":null,
  "place":null,
  "retweet_count":816600 }
```

Tweets are encoded in JSON format.

3 types of information:

1. **Tweet information**
2. **User information**
3. **Geographic information**
2. Capturing and analyzing tweets using the Streaming API

- We will learn how to:
  1. Capture tweets that contain a given keyword
  2. Basic sentiment analysis
  3. Capture geo-tagged tweets from a given location
  4. Map tweets by location

- Code: 04_tweets_by_keyword.R and 05_tweets_by_location.R.

- Note that you will need a more robust workflow to do this at a larger scale. I personally use:
  - Amazon EC2 Ubuntu micro instance (free tier)
  - Cron jobs to restart R scripts every hour.
  - Save tweets in .json files or in mySQL tables.
Capturing tweets by keyword

```r
# loading library and OAuth token
library(streamR, quietly = TRUE)

load("my_oauth")
# capturing 3 minutes of tweets mentioning obama or biden
filterStream(file.name = "tweets_keyword.json", track = c("obama", "biden"),
             timeout = 180, oauth = my_oauth)

# parsing tweets into dataframe
tweets <- parseTweets("tweets_keyword.json", verbose = TRUE)

## 317 tweets have been parsed.
```
# preparing words for analysis

```r
clean.tweets <- function(text) {
  # loading required packages
  lapply(c("tm", "Rstem", "stringr"), require, c = T, q = T)
  words <- removePunctuation(text)
  words <- wordStem(words)
  # splitting in words
  words <- str_split(text, " ")
  return(words)
}
```

# classify an individual tweet

```r
classify <- function(words, pos.words, neg.words) {
  # count number of positive and negative word matches
  pos.matches <- sum(words %in% pos.words)
  neg.matches <- sum(words %in% neg.words)
  return(pos.matches - neg.matches)
}
```
# function that applies sentiment classifier
classifier <- function(tweets, pos.words, neg.words, keyword) {
    # subsetting tweets that contain the keyword
    relevant <- grep(keyword, tweets$text, ignore.case = TRUE)
    # preparing tweets for analysis
    words <- clean.tweets(tweets$text[relevant])
    # classifier
    scores <- unlist(lapply(words, classify, pos.words, neg.words))
    n <- length(scores)
    positive <- as.integer(length(which(scores > 0))/n * 100)
    negative <- as.integer(length(which(scores < 0))/n * 100)
    neutral <- 100 - positive - negative
    cat(n, "tweets about", keyword, ":", positive, "% positive," , negative, "% negative," , neutral, "% neutral")
}
# loading lexicon of positive and negative words
lexicon <- read.csv("lexicon.csv", stringsAsFactors = F)
pos.words <- lexicon$word[lexicon$polarity == "positive"]
neg.words <- lexicon$word[lexicon$polarity == "negative"]

# applying classifier function
classifier(tweets, pos.words, neg.words, keyword = "obama")

## 294 tweets about obama : 13 % positive, 16 % negative, 71 % neutral

classifier(tweets, pos.words, neg.words, keyword = "biden")

## 16 tweets about biden : 31 % positive, 0 % negative, 69 % neutral
Capturing tweets by location

```r
# loading library and OAuth token
library(streamR)
load("my_oauth")

# capturing 2 minutes of tweets sent from Africa
filterStream(file.name = "tweets_africa.json", locations = c(-20, -37, 52, 35),
             timeout = 120, oauth = my_oauth)

# parsing tweets into dataframe
tweets.df <- parseTweets("tweets_africa.json", verbose = TRUE)
```
Tweets from Africa:

- 358,374 tweets collected for 24 hours on March 17th.
Tweets from Korea: 41,194 tweets collected on March 18th (left)
Korean peninsula at night, 2003 (right). Source: NASA.
Who is tweeting from North Korea?

Twitter user: @uriminzok_engl
But remember...

PET PEEVE #208:
GEOGRAPHIC PROFILE MAPS WHICH ARE
BASICALLY JUST POPULATION MAPS
Scraping electoral results in Georgia

URL: Central Electoral Comission of Georgia
3. Scraping electoral results in Georgia

- District-level results are not available for direct download
- However, information is structured in a series of HTML tables
- Even original document with electoral returns can be downloaded!
- We will learn how to:
  1. Parse HTML tables
  2. Use regular expressions to “clean” data
  3. Program function that parses list of URLs
  4. Quick visualization of results
- Code: 06_scraping_election_georgia.R
library(XML)
url <- "http://results.cec.gov.ge/index.html"

# how to know which table to extract? run 'str(table)' and look for table of interest. Alternatively, search html code for table ID
table <- readHTMLTable(url, stringsAsFactors = F)

table[1:6, 2:6]

<table>
<thead>
<tr>
<th></th>
<th>V2</th>
<th>V3</th>
<th>V4</th>
<th>V5</th>
<th>V6</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>1</td>
<td>4</td>
<td>5</td>
<td>9</td>
<td>10</td>
</tr>
<tr>
<td>2</td>
<td>222(0.56%)</td>
<td>51(0.13%)</td>
<td>13229(33.44%)</td>
<td>16(0.04%)</td>
<td>413(1.04%)</td>
</tr>
<tr>
<td>3</td>
<td>380(0.54%)</td>
<td>92(0.13%)</td>
<td>16728(23.62%)</td>
<td>44(0.06%)</td>
<td>939(1.33%)</td>
</tr>
<tr>
<td>4</td>
<td>358(0.4%)</td>
<td>123(0.14%)</td>
<td>22539(25.06%)</td>
<td>56(0.06%)</td>
<td>1047(1.16%)</td>
</tr>
<tr>
<td>5</td>
<td>82(0.31%)</td>
<td>27(0.1%)</td>
<td>9991(37.67%)</td>
<td>64(0.24%)</td>
<td>344(1.3%)</td>
</tr>
<tr>
<td>6</td>
<td>164(0.26%)</td>
<td>56(0.09%)</td>
<td>19778(31.06%)</td>
<td>92(0.14%)</td>
<td>1052(1.65%)</td>
</tr>
</tbody>
</table>
# deleting percentages (anything inside parentheses)
library(plyr)
table <- ddply(table, 2:18, function(x) gsub("\\(.*)\\", ", x))
# changing variable names
names(table) <- c("district", paste0("party_", table[1, 2:18]))
# deleting unnecessary row/column (party names and empty column)
table <- table[-1, -18]
# fixing district names
table$district <- as.numeric(gsub("(.*)\\..*", repl = "\\1", table$district))
# fixing variable types
table[, 2:17] <- apply(table[, 2:17], 2, as.numeric)
table[1:5, 1:5]

###
<table>
<thead>
<tr>
<th>district</th>
<th>party_1</th>
<th>party_4</th>
<th>party_5</th>
<th>party_9</th>
</tr>
</thead>
<tbody>
<tr>
<td>2</td>
<td>1</td>
<td>222</td>
<td>51</td>
<td>13229</td>
</tr>
<tr>
<td>3</td>
<td>2</td>
<td>380</td>
<td>92</td>
<td>16728</td>
</tr>
<tr>
<td>4</td>
<td>3</td>
<td>358</td>
<td>123</td>
<td>22539</td>
</tr>
<tr>
<td>5</td>
<td>4</td>
<td>82</td>
<td>27</td>
<td>9991</td>
</tr>
<tr>
<td>6</td>
<td>5</td>
<td>164</td>
<td>56</td>
<td>19778</td>
</tr>
</tbody>
</table>
Scraping district-level electoral results

```r
# list of district numbers
districts <- table$district

# replicate same steps as above for each district
evaluate_results <- function(district) {
    # read and parse html table
    url <- paste0("http://results.cec.gov.ge/olq_", district, ".html")
    results <- readHTMLTable(url, stringsAsFactors = F)
    results <- results$table36

    # variable names = party numbers
    names(results) <- c("section", paste0("party_", results[1, 2:length(results)]))

    # deleting first row and last column
    results <- results[-1, -length(results)]
    results$district <- district
    return(results)
}
```
# empty list to populate with district-level data
results <- list()
# loop over districts
for (district in districts) {
  results[[district]] <- extract.results(district)
}
# convert list to data.frame
results <- do.call(rbind, results)

results[1:5, c(1:5, length(results))]

## section party_1 party_4 party_5 party_9 district
## 2  1   8   2  262   0  1
## 3  2   0   0  239   1  1
## 4  3   4   0  287   2  1
## 5  4   5   4  197   0  1
## 6  5   6   1  284   0  1
# function to extract last digit
last.digit <- function(votes) {
  last.pos <- nchar(votes)
  as.numeric(substring(votes, last.pos, last.pos))
}

# histogram of last digit for two main parties
plot(table(last.digit(results$party_5)))
plot(table(last.digit(results$party_41)))
Scraping bribes data from India

URL: www.ipaidabribe.com
4. Constructing a dataset of bribes paid in India

- Crowdsourcing to combat corruption: ipaidabribe.com
- Data on 18,000 bribes paid in Indian, self-reported
- Information about how much, where, and why.
- We will learn how to:
  - Parse semi-structured HTML code
  - Find “node” of interest and extract it
  - Use regular expressions to clean data
  - Prepare a short script to extract data recursively
- Code: 07_scraping_india_bribes.R
Introduction to regular expressions

- Regular expressions (regex) are patterns used to “match” text strings.
- Used in combination with `grep` (find) and `gsub` (find and replace all)
- Most common expression patterns:
  - . matches any character, ^ and $ match beginning and end of line.
  - Any character followed by {3}, *, + is matched exactly 3 times, 0 or more times, 1 or more times.
  - [0–9], [a–ZA–Z], [:alnum:] matches any digit, any letter, or any digit and letter.
  - To extract pattern (not just replace), use parentheses and option repl="\\1".
  - In order to match special characters (., \, ( ) etc), they need to preceded by a backslash.
  - Type `?regex` for more details.
- Perl regex can also be used in R (option perl=TRUE)
# Starting with the first page
url <- "http://www.ipaidabribe.com/reports/paid"
# read html code from website
url.data <- readLines(url)

# parse HTML tree into an R object
library(XML)
doc <- htmlTreeParse(url.data, useInternalNodes = TRUE)

# extract what we need: descriptions and basic info for each bribe
titles <- xpathSApply(doc, "//div[@class='teaser-title']", xmlValue)
attributes <- xpathSApply(doc, "//div[@class='teaser-attributes']", xmlValue)
# note that we only need the first 10
titles <- titles[1:10]
attributes <- attributes[1:10]

## all those '\t' and '\n' are just white spaces that we can trim
library(stringr)
titles <- str_trim(titles)

# the same for the bribe characteristics
cities <- gsub(".*\[4\]\{5\}.+", attributes, replacement = "\\1")
depts <- gsub(".*\n\t (.*)\t\t.*", attributes, replacement = "\\1")
amounts <- gsub(".*Rs. \([0-9]+\).*", attributes, replacement = "\\1")

# we can put it together in a matrix
page.data <- cbind(titles, cities, depts, amounts)
# let's wrap it in a single function
extract.bribes <- function(url) {
  require(stringr)
  cat("url:", url)
  url.data <- readLines(url)
  doc <- htmlTreeParse(url.data, useInternalNodes = TRUE)
  titles <- xpathSApply(doc, "//div[@class='teaser-title']", xmlValue)[1:10]
  attributes <- xpathSApply(doc, "//div[@class='teaser-attributes']", xmlValue)
  titles <- str_trim(titles)
  cities <- gsub(".*\[4\](.*)\[5\].*", attributes, replacement = "\\1")
  depts <- gsub(".*\n\t\t(\.*\t\t.*)\t\t\t.*", attributes, replacement = "\\1")
  amounts <- gsub(".*Rs. ([0-9]*).*", attributes, replacement = "\\1")
  return(cbind(titles, cities, depts, amounts))
}
## all urls
urls <- paste0("http://www.ipaidabribe.com/reports/paid?page=", 0:50)

## empty array
data <- list()

## looping over urls...
for (i in seq_along(urls)) {
  # extracting information
data[[i]] <- extract.bribes(urls[i])
  # waiting one second between hits
  Sys.sleep(1)
  cat(" done!\n")
}

## transforming it into a data.frame
data <- data.frame(do.call(rbind, data), stringsAsFactors = F)
# quick summary statistics

```r
head(sort(table(data$depts), dec = T))
```

```r
## Railway Police       Police
## 406                   36
## Airports Stamps and Registration
## 13                    11
## Passport Customs, Excise and Service Tax
## 9                     7
```

```r
head(sort(table(data$cities), dec = T))
```

```r
## Bangalore    Mumbai New Delhi Pune Gurgaon
## 157          151       47     23    23  12
```

```r
summary(as.numeric(data$amounts))
```

```r
##     Min.  1st Qu.   Median      Mean  3rd Qu.    Max.  NA's
##      0   200     800   66600   4000   5000000    1
```
References

- Hanretty, Chris. *Scraping the Web for Arts and Humanities*. LINK
- Leipzig, Jeremy and Xiao-Yi Li. *Data Mashups in R*. O’Reilly
- R libraries: scrapeR, XML, twitteR (check vignettes for examples)
- Python libraries: BeautifulSoup, tweepy
- Alex Hanna’s *Tworkshops* LINK