MY472 – Week 7: APIs

Pablo Barberá & Akitaka Matsuo

MY 472: Data for Data Scientists

November 13, 2018

Course website: lse-my472.github.io

Course outline

1. Introduction to data
2. The shape of data
3. Cloud computing
4. Basics of HTML and CSS
5. Using data from the internet
6. (Reading week)
7. Working with APIs
8. Creating and managing databases
9. Interacting with online databases
10. Exploratory data analysis
11. Parallel computing

Seminar schedule

7. APIs
   - 3rd marked assignment (in groups)
   - Deadline: November 23
8. SQL
9. Online databases
   - 4th marked assignment (in groups)
   - Deadline: December 7th
10. Exploratory data analysis
11. Course wrap-up
    - 5th marked assignment (individual)
    - Deadline: December 21st

Take-home exam due January 18

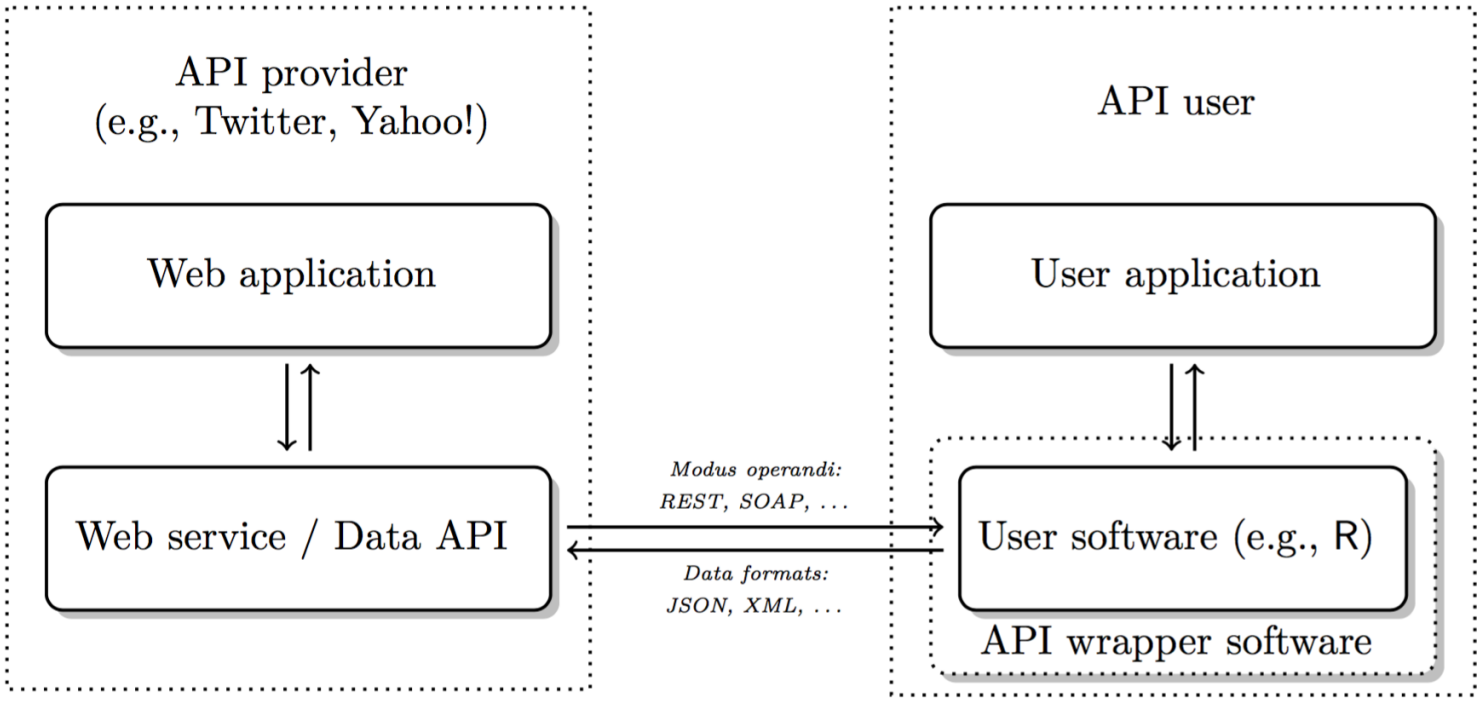
Plan for today

- APIs
  - Definition
  - Types of APIs
  - Constructing an API call
  - Authentication
  - Example: the New York Times API
- Social media data
  - What data is available?
  - Twitter APIs

APIs

API = Application Programming Interface; a set of structured http requests that return data in a lightweight format.

HTTP = Hypertext Transfer Protocol; how browsers and e-mail clients communicate with servers.



Source: Munzert et al, 2014, Figure 9.8

APIs

Types of APIs:

1. RESTful APIs: queries for static information at current moment (e.g. user profiles, posts, etc.)
2. Streaming APIs: changes in users' data in real time (e.g. new tweets, weather alerts...)

APIs generally have extensive documentation:

- Written for developers, so must be understandable for humans
- What to look for: endpoints and parameters.

Most APIs are rate-limited:

- Restrictions on number of API calls by user/IP address and period of time.
- Commercial APIs may impose a monthly fee

Connecting with an API

Constructing a REST API call:

- Baseline URL endpoint:
  `https://maps.googleapis.com/maps/api/geocode/json`
- Parameters: `?address=london`
- Authentication token (optional): `&key=XXXXX`

From R, use `httr` package to make `GET` request:

```
library(httr)
r <- GET(
"https://maps.googleapis.com/maps/api/geocode/json",
query=list(address="london", key="XXXXX"))
```

If request was successful, returned code will be 200, where `4xx` indicates client errors and `5xx` indicates server errors.
If you need to attach data, use `POST` request.

```
{
   "results" : [
      {
         "address_components" : [
            {
               "long_name" : "London",
               "short_name" : "London",
               "types" : [ "locality", "political" ]
            },
            {
               "long_name" : "London",
               "short_name" : "London",
               "types" : [ "postal_town" ]
            }
         ],
         "formatted_address" : "London, UK",
         "geometry" : {
            "bounds" : {
               "northeast" : {
                  "lat" : 51.6723432,
                  "lng" : 0.148271
               },
               "southwest" : {
                  "lat" : 51.38494009999999,
                  "lng" : -0.3514683
               }
            },
            "location" : {
               "lat" : 51.5073509,
               "lng" : -0.1277583
            },
...
}
```

```
{
...
            "location_type" : "APPROXIMATE",
            "viewport" : {
               "northeast" : {
                  "lat" : 51.6723432,
                  "lng" : 0.148271
               },
               "southwest" : {
                  "lat" : 51.38494009999999,
                  "lng" : -0.3514683
               }
            }
         },
         "place_id" : "ChIJdd4hrwug2EcRmSrV3Vo6llI",
         "types" : [ "locality", "political" ]
      }
   ],
   "status" : "OK"
}
```

JSON

Response is often in JSON format (Javascript Object Notation).

- Type: `content(r, "text")`
- Data stored in key-value pairs. Why? Lightweight, more flexible than traditional table format.
- Curly brackets embrace objets; square brackets enclose arrays (vectors)
- Use `fromJSON` function from `jsonlite` package to read JSON data into R
- But many packages have their own specific functions to read data in JSON format; `content(r, "parsed")`

Authentication

- Many APIs require an access key or token
- An alternative, open standard is called OAuth
- Connections without sharing username or password, only temporary tokens that can be refreshed
- `httr` package in R implements most cases (examples)

R packages

Before starting a new project, worth checking if there's already an R package for that API. Where to look?

- CRAN Web Technologies Task View (but only packages released in CRAN)
- GitHub (including unreleased packages and most recent versions of packages)
- rOpenSci Consortium

Also see this great list of APIs in case you need inspiration.

Why APIs?

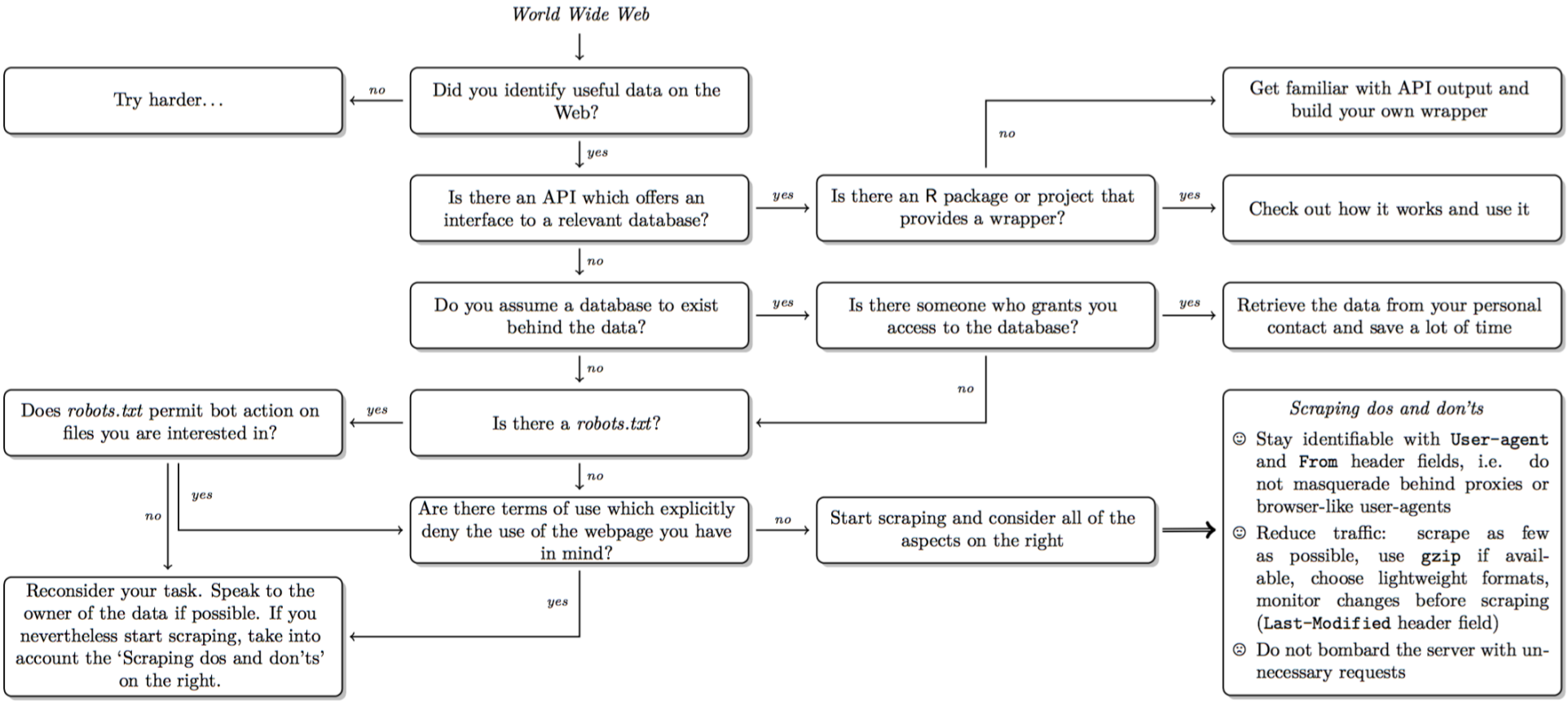
Advantages:

- 'Pure' data collection: avoid malformed HTML, no legal issues, clear data structures, more trust in data collection...
- Standardized data access procedures: transparency, replicability
- Robustness: benefits from 'wisdom of the crowds'

Disadvantages

- They're not too common (yet!)
- Dependency on API providers
- Rate limits

Decisions, decisions...



Example: the New York Times API

see `01-nytimes-api.Rmd`

Twitter data

Twitter APIs

Two different methods to collect Twitter data:

1. REST API:
   - Queries for specific information about users and tweets
   - Search recent tweets
   - Examples: user profile, list of followers and friends, tweets generated by a given user ("timeline"), users lists, etc.
   - R library: tweetscores (also twitteR, rtweet)
2. Streaming API:
   - Connect to the "stream" of tweets as they are being published
   - Three streaming APIs:
     - 2.1 Filter stream: tweets filtered by keywords
     - 2.2 Geo stream: tweets filtered by location
     - 2.3 Sample stream: 1% random sample of tweets
   - R library: streamR

Important limitation: tweets can only be downloaded in real time (exception: user timelines, $\sim$ 3,200 most recent tweets are available)

Anatomy of a tweet

Anatomy of a tweet

Tweets are stored in JSON format:


```
{ "created_at": "Wed Nov 07 04:16:18 +0000 2012",
  "id": 266031293945503744,
  "text": "Four more years. http://t.co/bAJE6Vom",
  "source": "web",
  "user": {
    "id": 813286,
    "name": "Barack Obama",
    "screen_name": "BarackObama",
    "location": "Washington, DC",
    "description": "This account is run by Organizing for Action staff.
        Tweets from the President are signed -bo.",
    "url": "http://t.co/8aJ56Jcemr",
    "protected": false,
    "followers_count": 54873124,
    "friends_count": 654580,
    "listed_count": 202495,
    "created_at": "Mon Mar 05 22:08:25 +0000 2007",
    "time_zone": "Eastern Time (US & Canada)",
    "statuses_count": 10687,
    "lang": "en" },
  "coordinates": null,
  "retweet_count": 756411,
  "favorite_count": 288867,
  "lang": "en"
}
```

Streaming API

- Recommended method to collect tweets
- Potential issues:
  - Filter streams have same rate limit as spritzer: when volume reaches 1% of all tweets, it will return random sample
  - Good to restart stream connections regularly.
- My workflow:
  - Amazon EC2, cloud computing
  - Cron jobs to restart R scripts every hour.
  - Save tweets in .json files, one per day.

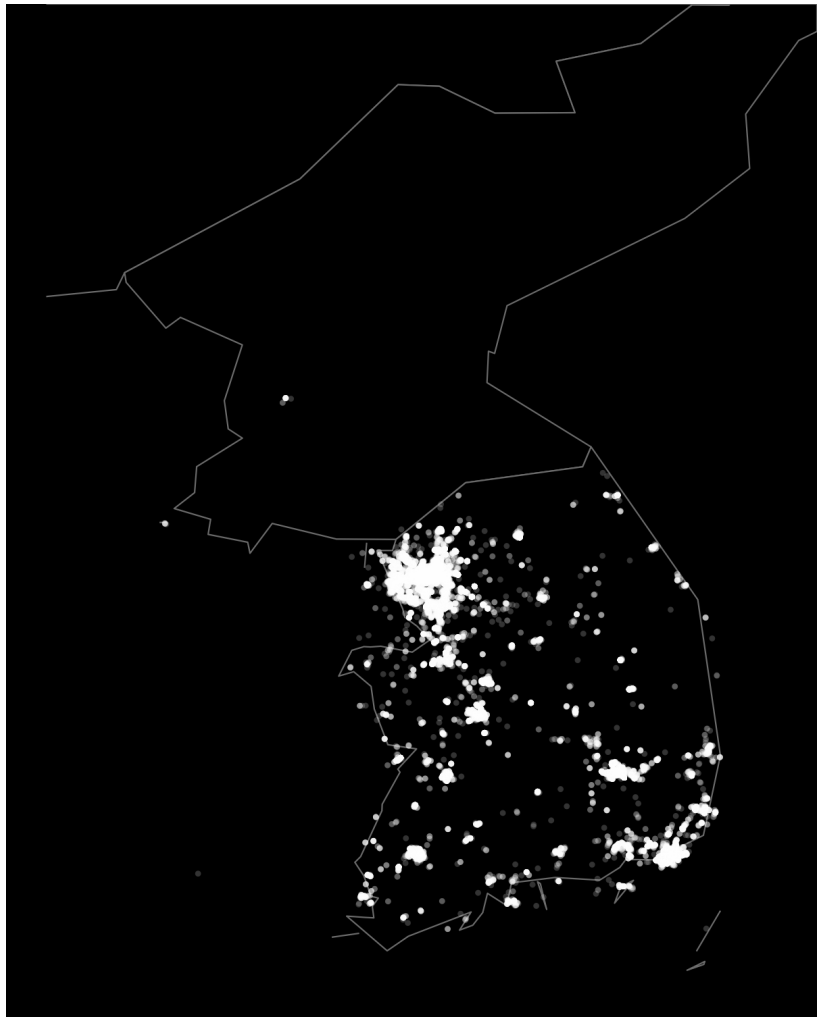
Sampling bias?

Morstatter et al, 2013, *ICWSM*, "Is the Sample Good Enough? Comparing Data from Twitter's Streaming API with Twitter's Firehose":

- 1% random sample from Streaming API is not truly random
- Less popular hashtags, users, topics... less likely to be sampled
- But for keyword-based samples, bias is not as important

González-Bailón et al, 2014, *Social Networks*, "Assessing the bias in samples of large online networks":

- Small samples collected by filtering with a subset of relevant hashtags can be biased
- Central, most active users are more likely to be sampled
- Data collected via search (REST) API more biased than those collected with Streaming API



Tweets from Korea: 40k tweets collected in 2014 (left)
Korean peninsula at night, 2003 (right). Source: NASA.

Who is tweeting from North Korea?

Twitter user: @uriminzok_engl

Facebook data

Collecting Facebook data

Facebook used to allow access to public pages' data through the Graph API:

1. Posts on public pages and groups
2. Likes, reactions, comments, replies...

Currently not available.

Aggregate-level statistics available through the FB Marketing API. See the code by Connor Gilroy (UW)

Access to other (anonymized) data used in published studies requires permission from Facebook or from users.

Social Science One as a new model for academic partnerships with Facebook.

Example: Twitter API

see `02-twitter-streaming-api.Rmd`

see `03-twitter-rest-api.Rmd`