Text analysis workshop: Getting data from Twitter

Casey O'Hara

Overview

Facebook may be the devil right now, but Twitter only seems to topple democracies for good rather than evil. In any case, Twitter can provide a nice snapshot of public perceptions of every topic from climate change to pineapple on pizza - so it's a good place to get data for sentiment analysis.

There is an R package, `rtweet`, that access the Twitter API to automate searches and return data in a nice format. There is an older package, `twitteR`, that is deprecated - it still works but is no longer being updated. Use `rtweet` instead.

Twitter also has a new API that allows users to mine data all the way back to the beginning (https://thenextweb.com/twitter/2018/02/01/twitter-new-api-lets-developers-sift-every-tweet-history/) - previously the standard API would only allow searches within the past seven days, and recently their "premium" API only allowed searches back 30 days. Access to the full timespan was only available through their "enterprise" APIs that are extremely expensive.

Currently it seems like the `rtweet` package doesn't access these new API endpoints, but presumably it will be updated to take advantage eventually.

Packages and setting up Twitter access

To access the Twitter API you need to set up a developer account and an application within your account. Here's info on how to do that: https://medium.com/@GalarnykMichael/accessing-data-from-twitter-api-using-r-part1-b387a1c7

If you haven't already done that, no problem, we'll just use previously collected data. If you have, here's some code to load the packages and set up the OAuth (https://en.wikipedia.org/wiki/OAuth) to access your account.

I've put my various keys into a text file that is not stored on GitHub, so the Russians can't use my Twitter access for nefarious purposes. `scan()` reads the lines one by one into a vector of strings. There are probably more secure ways to do this but this is good enough for me for now.

NOTE on creating an OAuth token in R:

The `rtweet::create_token()` function requires that you have an app already set up in your developer account, that the app has **read-write access**, and that the **callback URL** of the app is set to http://127.0.0.1:1410.

Once this is set up, `create_token()` will open a browser window to allow you to authorize access.

```
library(tidyverse)
library(stringr)

# install.packages('rtweet')
library(rtweet)
```

```
### This chunk is set to not evaluate when knitting...
### Scan the text file containing the keys; don't keep the text file on Github.
twitter_key <- scan('~/github/api_keys/twitter.txt', what = 'character')
```

```
consumer_key    <- twitter_key[1]
consumer_secret <- twitter_key[2]

rtweet::create_token(app = "rtweet_sentiments_testing", consumer_key, consumer_secret,
  set_renv = TRUE)
```

Using the Twitter API

Now we can set up a search; check `?search_tweets` to see the parameters available. Let's test a hypothesis that Apple users (iPhones and iPads) are more favorable toward #aquaculture than Android users, while Android users are more favorable toward #fisheries.

Important parameters:

- `q`: query that you are looking for.
  - In the query field, a space acts as an "AND" - so `'#fisheries #aquaculture'` would only get tweets that included both terms.
  - To get tweets that contain either, use `'#fisheries OR #aquaculture'`.
  - To get tweets with an exact phrase, use double quotes in the query (which is then surrounded by single quotes): `'"match this exact phrase"'`.
- `n`: how many tweets do you want to retrieve? If there aren't that many available in the search period (7-10 days for standard API) then it will give back a smaller number.

Some of the other parameters you can enter could be pretty helpful:

- `type` to focus on "recent" (default), "popular", or "mixed". For our purposes, let's just use recent tweets.
- `include_rts` (T/F) to include or exclude retweets. For our purposes, seems like a retweet is an echo of a sentiment of the original, so we will keep them in.
- `geocode` to limit the geographic extent of the search, e.g. if you are only interested in aquaculture sentiments in California, you can set a center (lat long) and radius to limit searches.
- `retryonratelimit` (T/F) to allow the function to wait and retry later, in case you are running up against your allowable limit of access. If you want to access more than 18000 tweets (the max allowed within a 15 minute window), set this to be `TRUE`.

Setting up the search

Here is the code used to create the saved dataset, using the standard API. Since the access can be slow and I don't want to hit my rate limit, I've set a "if/else" function to only get tweets if there is no saved file available.

Note also: this API returns a ton of information in a data frame (`parse = TRUE` is default), including some cells that may contain a list column with vector elements. So even though it's a dataframe, it won't save to a .csv; instead we get an error message:

```
Error in stream_delim_(df, path, ...) :
  Don't know how to handle vector of type list.
```

Results

So we can save our results, let's identify the ones most interesting to our analysis, and get rid of some of the list columns. The hashtag column can potentially be a list column, so we'll use `unnest()` to separate the vector elements for each observation.

- status_id: tweet specific identifier?
- **created_at**
- user_id, screen_name: user-specific ID and screen name
- **text**: the actual tweet!
- **source**: What was the user using to post the tweet?
- reply_to_status_id, reply_to_user_id, reply_to_screen_name
- **is_quote**, **is_retweet**: true/false
- **favorite_count**, **retweet_count**
- **hashtags**: Note, multiple hashtags are *included as a list column*, so we may need to `unnest()` them.
- symbols
- urls_url, urls_t.co, urls_expanded_url: links within the tweet; *may be reported as list columns.*
- media_url, media_t.co, media_expanded_url, media_type, ext_media_url, ext_media_t.co, ext_media_expanded_url, ext_media_type: media (photos etc) attached to the tweet? *may be reported as list columns.*
- mentions_user_id, mentions_screen_name
- lang
- quoted_status_id, **quoted_text**, retweet_status_id, **retweet_text**: text of the original posts that are being quoted or retweeted in this instance.
- place_url, place_name, place_full_name, place_type, country, country_code: location info by text (or URL)
- geo_coords, coords_coords, bbox_coords: location info by geographic coordinates, *reported as a list column.*

```
### Who loves fisheries and aquaculture more - iphone users or android users?

fish_tweets_file <- 'data/fis_aq_tweets.csv'

if(!file.exists(fish_tweets_file)) {
  ### A bit slow so save it for future use...

  tw_fis_aq_raw <- rtweet::search_tweets('#fisheries OR #aquaculture', n = 10000,
                                include_rts = TRUE)

  tw_fis_aq <- tw_fis_aq_raw %>%
    select(status_id, created_at, text, source,
           is_quote, is_retweet,
           hashtags, favorite_count, retweet_count,
           contains('place'),
           contains('country')) %>%
    unnest(hashtags)

  write_csv(tw_fis_aq, fish_tweets_file)

} else {

  tw_fis_aq <- read_csv(fish_tweets_file)

}
```

Performing the sentiment analysis

Clean up the data

- We can identify user devices from the `source` column, based on whether we find 'iPhone', 'iPad', or 'Android', using `str_match` or `str_detect`.

- We can drop all the irrelevant hashtags using `filter`.
- We should ditch user names and URLs in the main text, so they don't get counted as a sentiment. We can use `str_replace_all` to get rid of these.

```
### Set up some devices and hashtags we want to test:
devices <- c('iphone', 'ipad', 'android') %>%
  paste(collapse = '|')
fis_aq_hash <- c('aquaculture', 'fisheries')

### Note use of POSIX instead of more complicated [^A-Za-z0-9<etc>].
tw_phone <- tw_fis_aq %>%
  select(created_at, text, source,
         # place_full_name, country,
         hashtags) %>%
  filter(tolower(hashtags) %in% fis_aq_hash) %>%
  mutate(text = str_replace_all(text, '[^[:ascii:]]', '_') %>%
           tolower(),
         text_clean = str_replace_all(text, '@[^ ]+', '_usr_'),
         text_clean = str_replace_all(text_clean, 'http[^ ]+', '_url_')) %>%
  mutate(device = str_match(tolower(source), devices))

### A couple more cleanups
tw_phone <- tw_phone %>%
  mutate(hashtags = tolower(hashtags),
         device = ifelse(is.na(device), 'other', device))
```

Attach sentiment words

Load the `tidytext` package, load a sentiment library. Turn each tweet text into individual words (using the text_clean variable we just created), then remove stop words and attach sentiment scores to the remainder.

```
library(tidytext)

sentiments_b <- tidytext::get_sentiments('bing')
sentiments_a <- get_sentiments('afinn')
sentiments_n <- get_sentiments('nrc')
# sentiments_l <- get_sentiments('loughran')

test_df <- tw_phone %>%
  tidytext::unnest_tokens(output = word, input = text_clean, token = 'words') %>%
  anti_join(tidytext::stop_words, by = 'word') %>%
  left_join(sentiments_b, by = 'word')
### Check the sentiment assignments by word - some are pretty funny,
### but also shows limitation of the word bank used

score_df <- test_df %>%
```

```
  count(device, hashtags, sentiment) %>%
  spread(sentiment, n) %>%
  mutate(score = (positive - negative) - mean(positive - negative, na.rm = TRUE))

ggplot(score_df, aes(x = device, fill = hashtags)) +
  theme_classic() +
  geom_bar(aes(y = score), stat = 'identity', position = 'dodge') +
  labs(y = 'sentiment score') +
  coord_flip()
```