Sampling with Twitter following graph with `aPPR`

Alex Hayes

2020-10-08

The Goal: Collect data on a portion of the Twitter following graph

Solution

Download the data from Twitter API using `rtweet` package

Difficulties:

- API is severely rate limited
- Who to actually sample?

Three key types of information we want

- user data (i.e. node attributes)
- who follows a user (i.e. edge data)
- who a user follows (i.e. edge data)

You're gonna need `rtweet`

We don't have time to discuss here in detail

You're adults, read the documentation at https://docs.ropensci.org/rtweet/

Get a developer token. This can take a while, so do this ASAP.

Two ways to identify Twitter users

1. `screen_name` (string): for example `"karlrohe"`
2. `user_id` (64 bit integer encoded as string): for example `"1191642560"`

Getting user information with rtweet

```r
library(rtweet)
library(dplyr)

lookup_users("karlrohe") %>%
  select(screen_name, user_id, friends_count)
## # A tibble: 1 x 3
##   screen_name user_id    friends_count
##   <chr>       <chr>              <int>
## 1 karlrohe    1191642560           306
```

More basic `rtweet`

```
get_friends("karlrohe") %>%
  head(3)
## # A tibble: 3 x 2
##   user      user_id
##   <chr>     <chr>
## 1 karlrohe 148593548
## 2 karlrohe 52675436
## 3 karlrohe 15433452
```

Problem 1: rate limits

See the official rate limits for details

- `lookup_users()` – 900 users / 15 min
- `get_friends()` – 15 neighborhoods / 15 min
- `get_followers()` – 15 neighborhoods / 15 min

Need to plan ahead and be smart about data collection

Problem 2: which users should you sample?

1. Start with *seed nodes* in communities you care about
2. Get other users in these communities:
    - Snowball sampling (whatever, fine)
    - Personalized PageRank (extremely lit)

Why snowball sampling is eh

Basically the graph diameter is too small

1-hop neighborhood: $\sim$1000 users
2-hop neighborhood: $\sim$100,000 to 5,000,000 users
3-hop neighborhood: all of Twitter

More reasons why snowball sampling is eh

1-hop neighborhoods aren't that interesting
2-hop neighborhoods are impossible to collect due to rate limits
3-hop neighborhoods won't fit on any computer you own

I didn't write any code for you so you'll have to do it yourself

Personalized PageRank

General strategy:

1. Pick seeds
2. Find the users with highest Personalized PageRank
3. Use these users

Note: Karl should have taught you about Personalized PageRank, if this doesn't make sense blame him

Why is Personalized PageRank a good idea

Computationally: can *approximate* Personalized PageRank using a random walk that only needs to see a small portion of the graph

Statistically: Fan Chen, Yilin Zhang and Karl showed you can recover clusters from DC-SBMs

Talk to the paper

1. Chen, F., Zhang, Y. & Rohe, K. *Targeted sampling from massive Blockmodel graphs with personalized PageRank*. 2019. pdf
2. Andersen, R., Chung, F. & Lang, K. *Local Graph Partitioning using PageRank Vectors*. 2006. pdf

I coded this shit up

R package `aPPR` currently available on Github

See documentation at https://rohelab.github.io/aPPR/

Easy to extend `aPPR` to new graph objects

Installing aPPR

```
install.packages("devtools")
devtools::install_github("RoheLab/aPPR")
```

How `aPPR` works

You give `aPPR`:

- a graph
- a seed node
- hyperparameters `alpha` and `epsilon`

`aPPR` gives you Personalized PageRanks

Example

```r
library(aPPR)

karlrohe_ppr <- appr(
  rtweet_graph(),       # use the Twitter graph
  "karlrohe",           # seed node
  epsilon = 1e-3,       # convergence criteria
)
```

The results object is a little fancy

```r
# for documentation about results R6 object
?Tracker

# what you actually care about
# is the stats field
karlrohe_ppr$stats
```

The PPR table

```
head(karlrohe_ppr$stats, 5)
## # A tibble: 5 x 7
##   name                    r      p in_degree out_degree 
##   <chr>               <dbl>  <dbl>     <dbl>      <dbl>
## 1 1191642560        0.212   0.118       2270        306
## 2 148593548         0.00219 0         149354       3868
## 3 52675436          0.00219 0           1165        212
## 4 15433452          0.00219 0         334865      41595
## 5 10320366480732~ 0.00219 0             152        483
```

The PPR table columns

- `name`: Node IDs. For the Twitter graph, always user ids, never screen names
- `p`: Personalized PageRank
- `r`: Residual / error bound for `p`
- `in_degree` / `out_degree`: Self-explanatory, hopefully

The PPR table columns

- `degree_adjusted`: `p` divided by `in_degree`
- `regularized`: `p` divided by `in_degree` plus a regularization term `tau` that defaults to the average in-degree

Basically `regularized` is a version of PageRank that is specific to a given community and *excludes high degree nodes from other communities*

Read Fan's (really great) paper for details!!

Yeah okay I don't actually care about `p` though

Correct, we don't want `p`, we want a graph! Crucially, we don't want to have to make a whole bunch of API requests a second time

I build a backend called `twittercache` that saves data from Twitter API requests into a database and avoids wasting API requests

aPPR + twittercache

```r
library(aPPR)

karlrohe_ppr <- appr(
  twittercache_graph(),  # cache the data!! critical!!
  "karlrohe",
  epsilon = 5e-6,
  verbose = TRUE
)
```

Joining additional user data to results

```r
library(dplyr)
library(twittercache)

# doesn't preserve node order!!
screen_names <- cache_lookup_users(
  karlrohe_ppr$stats$name
) %>%
  select(user_id, screen_name)
```

Joining additional user data to results

```r
ppr_with_node_data <- karlrohe_ppr$stats %>%
  left_join(
    screen_names,
    by = c("name" = "user_id")
  ) %>%
  arrange(desc(regularized)) %>%
  select(screen_name, regularized, p, r)
```

Who is Karl's favorite student? It's Sijia!!

```
select(head(ppr_with_node_data), screen_name, regularized
## # A tibble: 6 x 2
##   screen_name       regularized
##   <chr>                   <dbl>
## 1 karlrohe          0.000000646
## 2 Machine_LeanIn  0.00000000303
## 3 Machine_Leaning 0.00000000275
## 4 DS3madison      0.00000000188
## 5 uwMadStat       0.00000000187
## 6 SijiaFang       0.00000000185
```

Getting the graph out of `twittercache`

```
twittercache::get_node_table()
twittercache::get_edge_table()
```

- Gets the whole graph, not necessarily the nodes with high PPR

Getting the graph out of `twittercache`

You'll need to write some more code to turn these tables into something useful. Feel free to contribute this back to `aPPR` as pull requests! I also have (limited) bandwidth to respond to feature requests.

The hyperparameters

Tuning knobs: `epsilon`

Recall we are doing *approximate* Personalized PageRank

`epsilon` is node-wise error tolerance

- `1e-3`, `1e-4`, `1e-5`: Make sure the code works, limited accuracy, $<$1hr runtime normally
- `1e-6`: 6-8 hours runtime, okay sample
- `1e-7`, `1e-8`: 1-3 days runtime, good sample

Node ranking can vary a lot with `epsilon`

Tuning knobs: `alpha`

Just use `alpha = 0.15`.

`alpha` controls the level of personalization, higher `alpha` is more personalized. There is no principled way to select the locality. This is a made up number. That's fine. Don't worry about it.

Tuning knobs: `tau`

In general you don't have to play with this, it only influences the `regularized` column

If `regularized` focuses too much on low degree nodes, increase `tau` to the node degree you care about most.

For Twitter, this is probably somewhere in the range 500 - 5000

Side note: I am doing some work on how to interpret `tau`

Technical caveats

Misc stuff to know

- `aPPR` samples *rectangular* adjacency matrices. i.e. don't observe edges for all nodes in the `ppr_table` (in particular, the ones with `p = 0`)
- If you need more precise results than whatever you currently have, use the `update()`

Getting help with the software

- Read the documentation at https://rohelab.github.io/aPPR/
- STAT 992 students can email me (sparingly please, I'm dying right now)

Limitations

`twittercache` slows down as the number of edges increases

If you get to ~10 million edges, clear out the cache

We're working on switching to a graph database backend but it's currently a low priority

Human, ethical, and legal caveats

There are major privacy considerations here

The social graph is fucking terrifying and reveals immense amounts of personal information

It's easy to identify interests, institutional affiliations, family members, close friends, etc, with `aPPR`. Do not share any of this information without explicit permission

Note: `aPPR` code is *not* licensed

Code is intellectual property that requires permission to use

Normally academic code has a permissive license that lets anyone use it

`aPPR` is *unlicensed*, meaning that we retain all copyright at the moment and it cannot be reproduced, distributed without permission and derivative work cannot be created without permission

Project ideas

PPR Hacking

Maximize your PPR with respect to `@karlrohe` on Twitter by the end of the semester via judicious following and unfollowing

I will sponsor up to $20 in bribes to get people to follow you if you make a sufficiently amusing case

Clustering

Implement new `aPPR` graphs (associated methods) for

- questions on Stack Overflow, or
- the Wikimedia knowledge graph

and see how spectral clustering works on these graphs