Using US Facebook Data on Server

Keng-Chi Chang

2017-10-18

Contents

1 Quick Start & Basic Structure

Due to the sheer size of Facebook data (`build` is now 5.6 TB), we recommend the following workflow:

1. Copy `analysis-[some-project]` folder to wherever you want.
2. Rename the `[some-project]` part to your project name.
3. Put your code inside `analysis-[xxx]/code`.
4. When you code:
    - Set `analysis-[xxx]` as your working directory (for example, `setwd()` in R, `cd` in Stata, and `os.chdir()` in Python).
    - Use relative paths in your code. Read data from `input`, export generated tables or figures to `output`, read and write other temporary files from `temp`.

Here are the things under the hood. `build` is for the process of data cleaning, and `analysis` is for data analysis. Raw data downloaded from Google BigQuery will be stored in `build/input`. Data after basic cleaning it will be put in `build/output`. A symbolic link is made to point from `analysis/input` to `build/output`. So every users can just copy an example `analysis` folder and read the `build/output` (cleaned) data *as if* it is in side their folder without making duplicated copies of data. For a general view of how this should work, see Gentzkow and Shapiro (2014).

2 Directories of Cleaned Data

Cleaned data are stored in `build/output`. A symbolic link is made to point from `analysis/input` to `build/output`. Hence these two folders contain the same documents. Please see Section 1 for how this is worked. To get a gist about what's inside, below presents the tree file structure for these folders:

```
analysis/input or build/output
├── page
│   ├── 1000-page-info.csv
│   ├── politician-info.csv
│   └── 1000-page-and-politician-info.csv
├── post
│   ├── 1000-page
│   │   └── 2015-01-01-to-2017-04-08.csv
│   └── politician
│       └── 2015-01-01-to-2016-11-30.csv
├── reaction
│   ├── 1000-page
│   │   ├── 2015-01-01-to-2016-11-30
│   │   │   └── us-political-user
│   │   │       └── by-reaction-type
│   │   │           └── LIKE
│   │   │               └── by-post-date
│   │   │                   ├── 2015-01-01.csv
│   │   │                   ├── ...(omitted)...
│   │   │                   └── 2016-11-30.csv
│   │   └── 20-min
│   │       └── by-reaction-type
│   │           ├── ANGRY.csv
│   │           ├── ...(omitted)...
│   │           ├── WOW.csv
│   │           └── LIKE
│   │               └── by-page-id
│   │               │   ├── 10018702564.csv
│   │               │   ├── ...(omitted)...
│   │               │   └── 99881661864.csv
│   │               └── by-time-stamp
│   │                   ├── 1475161200.csv
```

```
│   │                   ├── ...(omitted)...
│   │                   └── 1479740400.csv
│   └── politician
│       └── 2015-05-01-to-2016-11-30
│           └── us-political-user
│               └── by-reaction-type
│                   └── LIKE
│                       └── by-post-date
│                           ├── 2015-05-01.csv
│                           ├── ...(omitted)...
│                           └── 2016-11-29.csv
├── comment
│   └── 2015-01-01-to-2016-11-30
└── tree.txt
```

3 Directories of Raw Data

Inside `build/input` are the files downloaded directly from Google BigQuery. Since Google will automatically split large files into hundreds or thousands of small ones, it is not recommended to use these files directly.

```
build/input
├── page
│   ├── 1000-page-and-politician-info.csv
│   ├── 1000-page-info.csv
│   └── politician-info.csv
├── post
│   ├── 1000-page
│   │   └── 2015-01-01-to-2017-04-08.csv
│   └── politician
│       └── 2015-01-01-to-2016-11-30.csv
├── reaction
│   ├── 1000-page
│   │   ├── 2015-01-01-to-2016-11-30
│   │   │   └── us-political-user
│   │   │       └── by-reaction-type
│   │   │           └── LIKE
```

```
│   │   │               ├── 000000000000.csv
│   │   │               ├── ...(omitted)...
│   │   │               └── 000000004999.csv
│   │   └── 20-min
│   │       ├── 000000000000.csv
│   │       ├── ...(omitted)...
│   │       └── 000000000499.csv
│   └── politician
│       └── 2015-05-01-to-2016-11-30
│           └── us-political-user
│               └── by-reaction-type
│                   └── LIKE
│                       ├── 000000000000.csv
│                       ├── ...(omitted)...
│                       └── 000000000026.csv
├── comment
│   └── 2015-01-01-to-2016-11-30
│       ├── 000000000000.csv
│       ├── ...(omitted)...
│       └── 000000000499.csv
└── tree.txt
```

4 Introduction to Facebook Data

Only data related to fan pages are publicly available, which includes:

- Posts on fan pages
- Reactions to / comments of / public shares of these posts
- User id of those who do reactions, comments and shares
- Fan page likes other fan pages

A full list of variables can be found from Graph API Documentation.

4.1 Page Level Variables

Take the fan page Donald J. Trump for example:

- `page_id`: “153080620724”

  - You can see the page by visiting `www.facebook.com/[page_id]`
- `page_name`: "Donald J. Trump"
- `page_fan_count`: 22,813,525
- `page_url`: "www.facebook.com/DonaldTrump"
- `page_category`: "Public Figure"

4.2 Post Level Variables

There are 5 types of posts: link, status, photo, video, offer. Link is the most common type. Take this post Official White House Photographer Reveals His Favourite Photos Of Obama from 9GAG for example:

- `post_id`: "153080620724" or "21785951839_153080620724"
  - The latter format is basically `[page_id]_[post_id]`, which is preferred to the former (although it is said that the former is also uniquely defined)
  - You can see the post by visiting `www.facebook.com/[post_id]`
- `post_name`: "Official White House Photographer Reveals His Favourite Photos Of Obama"
  - This is the title of the link
- `post_message`: "Obama is the coolest president in history..."
  - This is the part the page manager types in
- `post_description`: "Click to see the pic and write a comment..."
  - This is the small font under the link title (`post_name`)
- `post_caption`: "9gag.com"
  - This is the small font under `post_description`
- `post_type`: "link"
- `post_link`: "http://9gag.com/gag/ajqEV90?ref=fbp"
  - This is the url that leads to when you click on the link
- `post_reactions`: 1297326
- `post_likes`: 1149630
- `post_comments`: 20093
- `post_shares`: 209506
- `post_created_time`: "2016-11-11T07:35:00+0000"
  - Original format is ISO 8601
  - I added a converted time `post_created_time_CT` using timezone America/Chicago, for which `CT` refers to Central Time

4.3 Reaction Level Variables

There are 6 types of reactions: LIKE, WOW, HAHA, SAD, ANGRY, THANKFUL. There is no id or time for reactions. So the way to define a unique reaction is to include both `post_id` and `user_id` as key. For example:

- `post_id`: "57972945858_10154109988750859"
- `user_id`: "766918176681835"
  - You can see the user's public profiile by visiting `www.facebook.com/[user_id]`
- `reaction_type`: "LIKE"
- `user_name`: "Trent Porter"

4.4 Comment Level Variables

Take this comment on "Conway on Trump's Hamilton feud: 'Who is to say that he can't do that' " as example:

- `comment_id`: "10154022206161680" or "10154022159491680_10154022206161680"
  - The latter format is basically `[post_id]_[comment_id]`, which is preferred to the former (although it is said that the former is also uniquely defined)
  - You can see the comment by visiting `www.facebook.com/[comment_id]`
- `comment_message`: "I'm getting really sick of seeing Ann Coulter-Lite's crazed, glassy-eyed face plastered all over the place, and Trump hasn't even been sworn in yet."
- `post_id`: "62317591679_10154022159491680"
- `comment_created_time`: "2016-11-22T05:47:18+0000"
- `user_id`: "100011100251277"
- `parent_id`: `null`
  - This is the `comment_id` when this comment is under another comment, otherwise `null`

We still have not download `user_id` and `parent_id` to our server.

5 Data on Server

5.1 Page

For definitions of variables, see Section 4.1. Merge with post and reaction by `page_id`, which is basically the part before underscore _ in `post_id`.

5.1.1 1000 Page

We select top 1000 pages related to 2016 US Presidential elections. The procedures of selection is as follows:

1. Find all the pages ever mentioned Donald Trump and Hillary Clinton in August 2016.
2. Calculate the total number of likes, comments, and shares of candidate-related posts in these pages, and weight them by factors 1:7:14 (a weight suggested by social media consultant, see Calero (2013)), respectively. Changing the weight does not change the list too much.

This will result in a list of pages that includes major news outlets, presidential primaries/general election candidates, and interest groups.

5.1.2 Politician

We find all US national politicians listed on Wikipedia, namely:

1. Senators: Current members and all 2016 Senate election candidates
2. House of Representatives: Current members and all 2016 House election candidates
3. Governors: Current and former Governor (last one).

Quite a few politicians own multiple pages. We include all of them. This will result in a total of 1475 pages.

5.1.3 Notes on Page Data

1. There are 9 pages overlapped between these two sets of pages: Tim Kaine, Bernie Sanders, U.S. Senator Bernie Sanders, Elizabeth Warren, U.S. Senator Elizabeth Warren, Ted Cruz, Rand Paul, Governor Jan Brewer, and Al Franken. Hence, if the data you want to merge includes both 1000 page and politician page, use `page/1000-page-and-politician-info.csv` where these duplicated pages are combined in one row.
2. There are a total of 366,840,068 unique users ever liked a post from the above two sets of pages in 2015 and 2016.
3. There are a total of 29,410,568 unique users ever liked a post from national politicians in 2015 and 2016, we call this kind of users *US political users*.
4. We failed to get the data from some pages in the 1000 page list. This is probably due to some pages are closed while we are scraping. Hence we result in 957 pages. For the list of missing pages, see Section 9.1.

5.2 Post

For definitions of variables, see Section 4.2. Merge with page by `page_id`. Merge with reaction by `post_id`.

5.2.1 1000 Page

All posts of 1000 page (see Section 5.1.1), from 2015-01-01 to 2017-04-08.

5.2.2 Politician

All posts of politician page (see Section 5.1.2), from 2015-01-01 to 2016-11-30.

5.3 Reaction

For definitions of variables, see Section 4.3. Merge with post by `post_id`. Merge with page by `page_id`, which is basically the part before underscore `_` in `post_id`. Merge with user by `user_id`. There is no id for reactions. Combining `user_id` and `post_id` defines a unique reaction.

5.3.1 1000 Page

There are two types of data of reactions on 1000 page.

1. Repeated capture every 20 minutes from 2016-09-29 to 2016-11-21:
    - Since we cannot know the exact time one makes an reaction, we can only approximate the reaction time by comparing the sets of reacted users between two different time for each post. After testing, 20 minutes seems to be the minimum time distance accepted by Facebook server.
    - `reaction_time` is the first time we find a user reacted to that post. Its format is Unix Timestamp, which is the number of seconds since 1970-01-01 00:00:00 at UTC time.
    - `reaction_type` is the type of reaction we found at `reaction_time`.
2. *LIKE* by *US political users* for posts from 2015-01-01 to 2016-11-30:
    - For this part, we use `post_created_time` as a indicator for time.
    - We currently only have *LIKE* (the most popular reaction, and no others) by *US political users* (see Section 5.1.3 for definition) on server.

5.3.2 Politician

Similar to the second type of reaction data of 1000 page.

- *LIKE* by *US political users* for posts from 2015-05-01 to 2016-11-30:
  - We use `post_created_time` as a indicator for time.
  - We currently only have *LIKE* (the most popular reaction, and no others) by *US political users* (see Section 5.1.3 for definition) on server.
  - To prevent likely duplications, we remove likes of overlapping pages from politician folder (see Section 5.1.3 for these 9 overlapping pages). Hence for reactions on these 9 pages, go to 1000 page folder.

5.4 Comment

Comment on post of 1000 page from 2015-01-01 to 2016-11-30. We currently do not have `user_id` and `comment_parent_id`. For definitions of variables, see Section 4.4. Merge with post by `post_id`. Merge with page by `page_id`, which is basically the part before underscore `_` in `post_id`. Merge with user by `user_id`.

6 Relationship Between Data on Server and BigQuery

The true raw data we get is on Google BigQuery. After cleaning and combining using SQL on BigQuery, we download the data to our server to become the so-called raw data on server. Below presents the relationship between these two sets of data. On server part, the path is relative to `build/input/`.

```
Server                                Google BigQuery
page/
  politician-info.csv                 [ntufbdata:politician_info.politician_info]
  1000-page-and-politician-info.csv   [ntufbdata:1000_page_and_politician_info.
                                       1000_page_and_politician_info]
post/
  1000-page/                          [ntufbdata:1000_page_post.20150101_to_
    2015-01-01-to-2017-04-08.csv       20170408_all]
  politician/                         [ntufbdata:politician_post.201501_to_
    2015-01-01-to-2016-11-30.csv       201611_all]
reaction/
  1000-page/
```

```
    2015-01-01-to-2016-11-30/
      us-political-user/
        by-reaction-type/           [ntufbdata:USdata.1000_page_us_user_like_
          LIKE/                      post_201501_to_201611_all]
    20-min/                        [ntue-data-sci:NTUE_TEST.Merge_Type_147]
  politician/
    2015-05-01-to-2016-11-30/
      us-political-user/
        by-reaction-type/           [ntufbdata:USdata.politician_us_user_post_
          LIKE/                      like_all_remove_duplicated_politicians]
comment/                           [ntufbdata:comment.old_posts_2015_2016_
  2015-01-01-to-2016-11-30/         comments]
```

7 Data Cleaning on Google BigQuery

Here outlines the process of cleaning and combining data on BigQuery using SQL. See `build/code/sql` for codes in this process.

7.1 Post

This part is quite complicated, since there are quite a few variables and multiple versions of data. The process mainly involves extracting variables, converting time zone to US Central Time (America/Chicago), combining data from different parsing time, and removed duplicated rows. See `build/code/sql/post.md` for code.

7.2 Reaction

This part involves extracting reactions, selecting likes, and join with post level data. See `build/code/sql/reaction.md` for code.

8 Data Cleaning on Server

After downloading data from BigQuery to server, there are still things to do. Most of the following tasks can also be done by other methods, such as Python, R, or Stata. However, due to the sheer size of our data, reading and writing billions of lines can be slow, not to mention

combining. So we mainly do these tasks using command line, treating these data as plain text format.

8.1 Combining Multiple Tables

Google Cloud Platform will automatic split tables into multiple files if the table is large. So we often have to combine tables.

```
head -n 1 000000000000.csv > ../combined.csv
cat *.csv | sed "1 d" >> ../combined.csv
mv ../combined.csv ../[some_name].csv
```

This code does the following:

1. Copy the first line (header) of `000000000000.csv`, and save it to `../combined.csv`
2. For all `.csv` files in this working directory, copy parts other than the first line, and append to `../combined.csv`
3. Rename the `../combined.csv` to `../[some_name].csv`

8.2 Split Tables by Some Columns

Since it is hard to save tables by grouping on some values, or often when the size of the combined table is too large for the computer's memory to read in, we often also need to split tables by some columns. Examples are tables involve `by-page-id`, `by-time-stamp`, or `by-reaction-type`.

If you have a bunch of `.csv` of the following form:

```
page_id,post_id,user_id,reaction_type,reaction_time
10513336322,10513336322_589193884601712,10153098041052436,SAD,1478708400
43279570822,43279570822_606917526146535,166728873700799,SAD,1475298000
5281959998,5281959998_1778548592362655,10203730351804631,SAD,1478720400
5281959998,5281959998_1778548592362655,10202915192094679,SAD,1478732400
```

You want to group the files by `page_id`, using it as file name. You can do:

```
cat *.csv | awk -F"," '{print $1","$2","$3","$4","$5 >> $1}'
```

That is, for each `.csv` files in this working directory, for each row in the file, split the row by ",". This will result in 5 strings. Print these 5 strings, separated by "," in to the bottom of the file named by the first string (in this case, `page_id`). If the file does not exist, create the

file. Note that headers will also be split to another file. Note also that the file extension also disappears. So we have other things to do.

8.3 Insert Header to File

Like methods in Section 8.2, headers will also be split to a different file, so we need to add them back. This can be done by

```
sed -i '1i userid,postid' *
```

which adds "userid,postid" header to the first line of all the files in current working directory.

8.4 Change Header to File

In Section 8.3, we have the wrong header format. If we want to change the header from "userid,postid" to "user_id,post_id", we can run:

```
sed -i '1s/.*/user_id,post_id/' *
```

to the current working directory.

8.5 Add File Extension

Method in Section 8.2 will also not priting the file extension. The resulting file is a plain text document. If we want to add a `.csv` back to the files, do the following in the current working directory:

```
find -type f -not -name "*.*" -exec mv "{}" "{}".csv \;
```

where it finds the files that does not have extension name, and rename them by adding a `csv` extension name to the file.

9 Appendix

9.1 List of 43 Missing Pages

page_id	page_name	fan_count
1318800798260799	BuzzFeed Video	11205386

page_id	page_name	fan_count
1471162443107177	StreetArtGlobe	3780997
1405630409737397	We are mitú	2575623
1457565357799107	Rare News	1491746
1435071773455316	Daily Wire	1360474
1069936086365702	Fusion	1237807
1476249345949752	RockIt News	1147851
1672814509645693	New Century Times	1007011
1558709644373144	Rare Media	933743
1645816202305254	The People For Bernie Sanders 2016	913870
1374461649456926	Greenville Gazette	834043
1591828521031839	The Conservative Update	679629
1768542800042528	Stewart Humor	570297
10150100007820094	Wendy Davis	537535
1413185958903624	Tactical Shit	531422
1575551819401302	My Favorite F Word Is Feminism	499365
1628417637371411	Consciously Enlightened	295438
1469899943335353	Trumpians	271198
1427314137490753	Sid Miller	259296
1480717605584637	Trump Wall	250199
1089660871073245	Overpasses For America	221331
1578660905749115	Hillary For Prison 2016	214336
1081327175240237	Trump Fan Network	184055
1378158369118238	Democratic Memes	155648
1481073582140028	Bloomberg Politics	145059
1604383669807606	NPR Politics	142312
1129007383778921	Heat Street	89065
1440616002820798	Daniel Scavino Jr.	86175
1458819194388319	Russia Insider	85997
1485438831785542	Real Progressives	84608
1643258135887835	Support Trump/Pence 2016	82606
1468815163437975	Viva Bernie 2016	75622
1397911960432824	Donald Trump Republic 2016	70029
1035839199769306	Progressive Politics	56627
1050515274971135	Never Trump	54745
1493047070916226	Eric Metaxas	53241
1559412207662264	Blue Dem Warriors	52126
1439321242961384	Vidmax.com	49762

page_id	page_name	fan_count
1604423586488781	Let The Madness Begin	45165
1038012599588678	Quotidian Conservative News	44537
1515278958697129	Mindy Fischer, Writer	31372
1420253964879948	Revolt Against Plutocracy	30638
1416409845316333	The Straight, White, Capitalist	24753

References

Calero, Antonio. 2013. "Likes Vs. Comments Vs. Shares." http://www.antoniocalero.com/2013/05/06/facebook-likes-comments-shares/.

Gentzkow, Matthew, and Jesse M. Shapiro. 2014. "Code and Data for the Social Sciences: A Practitioner's Guide." http://web.stanford.edu/~gentzkow/research/CodeAndData.pdf.