

Twitter Report

paul

March 14, 2017

Contents

Twitter API Concepts	1
Twitter Intro:	1
REST API:	2
Search API:	2
Streaming API:	3
Account and Access Token	3
Test Gmail Account:	3
Twitter Test Account:	3
Creating Application Access Token:	3
Rest API	4
Tools and Packages:	4
Evaluation of <i>twitteR</i> and REST API :	4
Tweeter Limits And Information:	4
Example 1: Creating a Word Cloud From Twitter	4
REST API Example 2: Random Exploration	9
Example 3: DataBase Connection Functions of <i>twitteR</i> :	10
REST API Example 4: Tweets maximum:	11
Stream API	11
Summary of streamR:	11
IMPORTANT NOTE:	11
libraries	11
Stream Example 1: Public Streams:	11
Stream API Example 2: UserStream	12

Twitter API Concepts

Twitter Intro:

- Structure of data storage is a graph. The graph contains 4 main objects/nodes:
 - Tweets
 - Users
 - Entities: metadata and contextual information. Often appears as a field in other objects.
 - Places: location information associated with endpoints.
- Each object can be referenced by an unique ID.

- Requests are made using HTTP requests.
 - **Divided into 2 types of API:**
 - *REST API*: which is used for requesting existing objects within Twitter.
 - *Streaming API*: used for streaming *LIVE* data from the twitter API stream.
 - Twitter goes beyond *OAuth* for security purposes, twitter server will negotiate a cipher upon establishing connection. TLS Information
 - This report likely uses pre-made packages and will not delve into the specifics for security.
 - Specific Twitter endpoints support pagination. To request for cursor results, add `&cursor=-1` to the request. If then endpoint/node support cursors, the API will default `cursor` to `-1`. The response value for cursor can be used to navigate.
 - Cursors
-

REST API:

- Only takes Application-Only Authentication (requests made on behalf of the application).
- Request format:
 - `https://api.twitter.com/1.1/{endpoints}/{fields}.json?q={query}`
- Rate limited by 15 minute windows, each endpoint/request has varying limitations. Limitation is a cumulative sum. For more information refer to REST Rate Limit
 - Rate Limit Table
 - GET requests can be made on the behalf of application or user account.
 - HTTP headers are available to request for rate limit information.
- Working with Timelines (like Home page on Facebook):
 - Since timelines are changing in real time, twitter adds parameters to avoid redundant information retrieval.
 - * *max_Id*: specifies the to retrieve posts up to and including the *max_Id*. This will return 1 redundant request. To avoid this, add 1 to the ID of the post (doesn't matter if the post exists or not).
 - * *since_Id*: extract posts after an id.
 - * Details on Working with Timelines
- URLs in twitter are often shortened to "twitter format" but the expanded URL is usually available in the response as well.
- Includes the option to retrieve/post private messages (not very applicable).

Search API:

- Essentially the search engine of twitter and will return information from public feed that matches search string.
 - This is part of the REST API.
 - Provides powerful query formats that can make very interesting queries such as:
 - politics filters: `politics filter:safe`
 - containing media: `puppy filter:media`
 - attitude: `flight:(`
 - hashtags: `#haiku`
 - Example Request: `'https://api.twitter.com/1.1/search/tweets.json?q=%40twitterapi'`
 - Search API
-

Streaming API:

- Twitter Stream API
 - This stream provides access to newly updated public tweets data.
 - Includes two useful types: *Public* and *User* stream API.
 - Public Stream: Live stream of public posts. *GET* for shorter URL requests while *POST* for longer URLs.
 - User Stream used to extract a person's view of twitter: direct messages, replies, following status, etc.
 - General process is to establish a connection to the stream API with a request and save the data into a database for future use. The connection is sustained unless error occurs or the user disconnects.
 - Does not have normal rate limit caps, however connections will be closed if:
 - attempting to establish too many connections.
 - suddenly stops reading data.
 - reads data at a slow pace such that the queue is filled.
 - For more details regarding stalls, reconnecting, etc. Refer to Connecting to Stream API
 - Each JSON return will be separated by `\r\n`
 - Missing fields will be indicated by a “-1”, use REST API to retrieve information.
 - Stream Message Types
 - will contain blank messages (to sustain connection), delete messages notifications, changes to tweets etc..
-

Account and Access Token

Test Gmail Account:

- *Name*: API-Testing(first name) NRC(last name)
 - *Username/Email*: NRC.API.Testing@gmail.com
 - *Password*: NRCTesting123
 - *Birthday*: July 1st 1997
 - *Gender*: Rather not say
-

Twitter Test Account:

- *Username*: NRC API-Testing
 - *Email*: NRC.API.Testing@gmail.com
 - *Password*: NRCTesting123
 - *Twitter Username*: NRC_API_Testing
-

Creating Application Access Token:

- 1: Register on to the Twitter Application Site.
- 2: Create a new Application. Fill in a placeholder for the *Website URL*.

3: After creation, click on the “keys and Token” tab and create tokens.

Rest API

Tools and Packages:

- Apigee Twitter API Console
 - Useful interface to test out queries to the API.
 - *twitteR*
 - *twitteR* Vignette
 - **Highly Recommended to Read**
-

Evaluation of *twitteR* and REST API:

Advantages: * The built-in class wrappers provides convenient and organized information. + functions associated with these class are very useful. like `user$id` etc. * Even though the package does not support custom requests, most information of interest is provided by default. * The built-in function for classes provides substantial details and is shown in an easily accessible way. * Unlike Facebook, many users & tweets are public and thus the account details and tweets are easily accessed.

Disadvantages: * However it is difficult to track a conversation and reactions to a post because it is not directly related to the original tweet. * The token generated by the package is cached somewhere that can be used by the *twitteR* package functions through out the session. However if another package requires access to the token, it is not easily accessible. * Tweets are often truncated in response unless specified. The built-in functions does not seem to support extracting untruncated tweets.

Tweeter Limits And Information:

- Can extract up to a maximum 3200 statuses from a user Timeline.
 - Each page of response can contain up to 200 results.
 - For search/tweets, each page of response can contain up to 100 tweets.
 - The *source owner* is mentioned in the text of the tweet (status in *twitteR* package) by an `_@_` sign followed by the source owner of the tweet.
 - URLs are often reported in twitter short hand, *twitteR* provides functionality to expand into URL and vice-versa.
-

Example 1: Creating a Word Cloud From Twitter

GOAL: To test out the *twitteR* package’s ability to scrape public twitter data.

- Google Building wordCloud

Application Settings

Keep the "Consumer Secret" a secret. This key should never be human-readable in your application.

Consumer Key (API Key)	oQ3PqERg75kPtgBcgOLaFShSC
Consumer Secret (API Secret)	d4cxaKc1Dt3ugagruUNPtWzvmqGHx8WwYAQ8MywUqTIVTTj9O
Access Level	Read and write (modify app permissions)
Owner	NRC_API_Testing
Owner ID	833674399224061952

Application Actions

[Regenerate Consumer Key and Secret](#)[Change App Permissions](#)

Your Access Token

This access token can be used to make API requests on your own account's behalf. Do not share your access token secret with anyone.

Access Token	833674399224061952- tL4gGOyGUrz84IbVlkkAmQzqUPahL1N
Access Token Secret	qkNmKD7TU5uZtIENW3r5K20wkqfbL6w37xyXLweiYBZg6
Access Level	Read and write
Owner	NRC_API_Testing
Owner ID	833674399224061952

Token Actions

[Regenerate My Access Token and Token Secret](#)[Revoke Token Access](#)

Figure 1: Twitter Token

- Building wordCloud
- Using tm Package

1: Import Libraries and getting Access Token

```
rm(list = ls())

library(twitterR)
library(httr)

library(tm)
library(wordcloud)
library(SnowballC)
library(RColorBrewer)

# access Tokens
consumer_Key = "oQ3PqERg75kPtgBcg0LaFShSC"
consumer_Secret = "d4cxaKc1Dt3ugagruUNPtWzvmqGHx8WvwYAQ8MywUqTIVTTj90"
access_Token = "833674399224061952-tL4gG0yGUrz84IbVlkkAmQzqUPahL1N"
access_Secret = "qkNmKD7TU5uZtIENW3r5K20wkqfbL6w37xyXLwelyBZg6"

## Initially, the function asks the user to cache the credentials and
## will be used for another session.
setup_twitter_oauth(consumer_Key,consumer_Secret,access_Token,access_Secret)

## [1] "Using direct authentication"

## Creating a token manually
app <- oauth_app("twitter", key=consumer_Key, secret=consumer_Secret)

token = Token1.0$new(endpoint = NULL, params = list(as_header = TRUE),
                     app = app, credentials = list(oauth_token = access_Token,
                                                    oauth_token_secret = access_Secret))
saveRDS(token,"tokenTest")
test_Token = readRDS("tokenTest")

## token and test_Token are the same object.

## loading cached token from twitterR package:
## It failed, likely because of the output format.
## The file size is 0KB
# oauth_content <- readRDS('.httr-oauth')
```

2: Getting Raw Data

```
## Some parameters
search_String = "NRC+OR+#NRC+OR+@NRC"
lang = "en"
since = "2016-01-01"

## Extracting coordinates for center of Canada:
google_Api_Key = "AIzaSyBGTs-gZCbyP8n0Hvw_VZ76Z6YrST1DNa8"
google_Host = "https://maps.googleapis.com/maps/api"

request = paste(google_Host,"/geocode/json",
```

```

        "?address=Canada&key=",
        google_Api_Key, sep="")
raw= GET(request)

data = jsonlite::fromJSON(
  httr:: content(raw,as="text")
)
lat = data$results$geometry$location["lat"]
lng = data$results$geometry$location["lng"]

geocode =paste(lat,lng,"2000km",sep=",")

## A list of tweets in Ottawa mentioning NRC. Note, the return
## already a "status"
NRC_Search = searchTwitter(search_String, n=200,
                           lang=lang, since=since,geocode =geocode)

## Warning in doRppAPICall("search/tweets", n, params = params,
## retryOnRateLimit = retryOnRateLimit, : 200 tweets were requested but the
## API can only return 165

## built in functions allow the specification of "untruncated tweets"
## This was done manually.

## Many tweets are truncated. Getting a list
## of ids for tweets that have been truncated.

truncated_Id = lapply(NRC_Search, function(x)
{
  if(x$truncated)
    return(x$id)
  else
    return(NA)
})

version = 1.1
cmd = "/statuses/show/"
param = "?tweet_mode=extended"

search_Id = truncated_Id[
  !is.na(truncated_Id)]

long_Tweet = list();

## Getting Untruncated tweets
## Going to use the GET method
for(i in 1:length(search_Id))
{
  url = paste("https://api.twitter.com/",
              version,cmd,

```

```

        search_Id[i], ".json",
        param, sep="")

##getting raw response
raw_Response = GET(url, config=token)

## expanded
long_Tweet[[i]] = jsonlite::fromJSON(
  httr::content(raw_Response, "text")
)
}

truncated_Id[!is.na(truncated_Id)] = long_Tweet

## Organized texts results
for(i in 1:length(NRC_Search)){
  if(NRC_Search[[i]]$truncated){
    NRC_Search[[i]] = truncated_Id[[i]]
  }
}

## removing twitter links
text_NRC = lapply(NRC_Search, function(x)
{
  text = x$text
  ## converting to ASCII
  text= iconv(x=text, from="UTF-8", to="ASCII", sub="")

  ##Clearing out URLs
  text=gsub("http(s?)://t.co/[a-zA-Z0-9]+",
    "", text)
  text=gsub("\nhttps:", "", text)
})

# rm(list = setdiff(ls(), c("token", "text_NRC")))

```

3: Make the word Map

```

require(twitterR)
## Constructing corpus (structure to process text)
cor= Corpus(VectorSource(text_NRC))
cor = tm_map(cor, removePunctuation)
cor = tm_map(cor, removeWords, stopwords('english'))
cor = tm_map(cor, stemDocument)

```


Example 3: DataBase Connection Functions of twitterR:

- The functionality provided by twitterR package behaves as the documentation outlines. To view testing code, refer to the *Twitter Rest API.Rmd* file.

```
library(twitterR)
search_Data = searchTwitter(searchString="sports",n=10,
                             lang="en")

db_Data = twitterR::twListToDF(search_Data)

##sorting text in search_Data
for(i in 1:length(db_Data$text)){
  db_Data[[i]] = gsub(pattern="\n", replacement = " ",
                     db_Data[[i]])
}

if(!file.exists("DB_Data.txt"))
{
  file.create("DB_Data.txt")
}

write.table(db_Data, file="./DB_Data.txt",row.names = F,
            fileEncoding = "UTF-8",sep="\t",col.names=F)
```

- A very simple database containing the 10 tweets from the *db_Data* and stored it in a local sql data base.

```
require(RMySQL)
require(twitterR)
```

```
db_name = "twitterdb"
user = "root"
host = "localhost"
password = "19970728Paul$"

## sets up a connection
DBI = register_mysql_backend(db_name,host,user,password)

## returns a list of twitterR status
loaded_Data = load_tweets_db(table_name = "status")
paste("Length", length(loaded_Data))
```

```
## [1] "Length 74"
```

```
## Trying to store tweets into the same db:
search_Data2 = searchTwitter(searchString="#glee",n=10,
                              lang="en")

## The new data is appended to the bottom
store_tweets_db(search_Data2,table_name="status")
```

```
## [1] TRUE
```

```
loaded_Data = load_tweets_db(table_name = "status")
paste("Length", length(loaded_Data))
```

```
## [1] "Length 75"
```

REST API Example 4: Tweets maximum:

GOAL: Test to see how many tweets can twitter return. * The maximum number of tweets is 3200.

Stream API

Summary of streamR:

- provides easy access to the twitter stream API.
- Handles connection, parsing, disconnecting, reconnecting, backing off and writing to a file all in the background

IMPORTANT NOTE:

- Make sure that the twitter application has “obb” specified in the call back URL, this will take the user to the authorization page to extract the pin to set up twitter handshake.
- It is recommended to store the RAW JSON/XML response into a file and then process it later on to reduce possible delay for streams.
- ___ The streaming functions provided by streamR only stores complete tweets and disregards deletion, updates, incomplete posts etc.___
- User stream returns only data for the authenticated user for this session. Which is the twitter test account for this report. Not much information due to the nature of the account being a test account.

libraries

- streamR: Handles connecting and extracting information from twitter stream apis.
-

Stream Example 1: Public Streams:

```
## The token is stored and read as an R object

if(!file.exists("stream Token")){
  file.create("stream Token")
  saveRDS(object = my_oauth, file="stream Token")
}

token = readRDS("stream Token")
```

```

##Creating a file to store data
if(!file.exists("tweets_CNN.json")){
  file.create("tweets_CNN.json")

  ## Can be controlled by either number of tweets
  ## and maximum connection time (timeout)

  filterStream( file.name="tweets_CNN.json",
    track="CNN", tweets=10, oauth=token)
}

## reading in saved file, convert it to a data frame.
## where each column is a field and each row is a tweet.
tweets_DB = parseTweets(tweets = "tweets_CNN.json")

```

6 tweets have been parsed.

```
names(tweets_DB)
```

```

## [1] "text" "retweet_count"
## [3] "favorited" "truncated"
## [5] "id_str" "in_reply_to_screen_name"
## [7] "source" "retweeted"
## [9] "created_at" "in_reply_to_status_id_str"
## [11] "in_reply_to_user_id_str" "lang"
## [13] "listed_count" "verified"
## [15] "location" "user_id_str"
## [17] "description" "geo_enabled"
## [19] "user_created_at" "statuses_count"
## [21] "followers_count" "favourites_count"
## [23] "protected" "user_url"
## [25] "name" "time_zone"
## [27] "user_lang" "utc_offset"
## [29] "friends_count" "screen_name"
## [31] "country_code" "country"
## [33] "place_type" "full_name"
## [35] "place_name" "place_id"
## [37] "place_lat" "place_lon"
## [39] "lat" "lon"
## [41] "expanded_url" "url"

```

```

## a list where each element is a JSON nested
## tweet
tweets_List = readTweets(tweets="tweets_CNN.json")

```

6 tweets have been parsed.

Stream API Example 2: UserStream

```

## Not much information is provided beacuse
## not many activites of the test account.

if(!file.exists("user stream.json")){

```

```
file.create("user stream.json")
userStream(file.name="user stream.json",
            timeout=120,oauth=token)

}
```