pyTweet

This module enables data scientists to build large datasets for graph analytics. It can be very difficult to obtain data sets for big data analysis, along with developing architecture for processing and storage. With pyTweet, a user can easily select a sampling method and have the collection run unsupervised. Profile and timeline metadata are saved in JSON file format. Modules add-ons can process the JSON files into a PostgreSQL database with a graph-like schema.

Noteworthy functionality:

- Create a complete data set for graph analysis using the Official Twitter API
- Collect profile details such as name, location, and images
- Get full network of friends and followers for a Twitter user
- Obtain full timeline details with media and user mention entities
- Pausing for rate limits is built into all API functions so user's do not have to implement it themselves

Contents of Documentation

1. **Getting Started**: This section describes the software requirements of pyTweet. It also provides instruction on how to create a Twitter application, which is needed for the REST API.
2. **Building a Network**: This section documents the available sampling methods in pyTweet as well as the API wrapper functions used in sampling.
3. **API wrapper functions**: In the case that breadth-first search is not appropriate for you, leverage the API wrapper functions to create your own sampling script.

Getting Started

Refer to this section to getting pyTweet's dependencies and accessing the Official Twitter API.

Python 2.7 and Installing pyTweet

The module pyTweet is compatible with Python version 2.7 and above. If installing Python for the first time, it's recommended to download it though Anaconda: https://store.continuum.io/cshop/anaconda/. In addition to the latest stable version of Python, Anaconda installs Numpy, Scipy, and other helpful modules that are not typically included in the regular Python installation. Once your Python environment has been established save the pyTweet module in the site-packages directory, i.e. /path/to/Anaconda/Lib/site-packages/pyTweet. [1]

Additional Python Modules

The module pyTweet is dependent on several Python modules. Many of these come with the standard version of Python, but some are not included in the standard distribution.

Module	Description	Source
requests	Create and handle web requests	http://docs.python-requests.org/en/latest/
ujson	UltraJSON is an ultra fast JSON encoder and decoder written in pure C with bindings for Python 2.5+ and 3.	https://pypi.python.org/pypi/ujson
re	This module provides regular expression matching operations similar to those found in Perl.	https://docs.python.org/2/library/re.html
urllib2	Create and handle web requests	https://docs.python.org/2/library/urllib2.html
requests-oauthlib	OAuth authorization - dependent on oauthlib below	https://github.com/requests/requests-oauthlib
oauthlib	OAuth authorization	https://github.com/idan/oauthlib

Obtaining Twitter API Keys

Twitter's REST API requires API keys for access. To obtain keys you'll need an application in addition to a regular Twitter account. The following steps explain how to create an application and get them:

1. Create a standard Twitter account on https://twitter.com/
2. Go to https://apps.twitter.com/ and login. Click on the *Create New App* button to create an application.
3. Fill in the form, accept the developer agreement, and finish creating your application
4. Go to the 'Permissions' tab, and select read only access
5. Go to the 'Keys and Access Tokens' tab and click the button 'Create my access token'. You may need to refresh the page to view the new tokens. Use the Consumer Key (API Key), Consumer Secret (API Secret), Access Token, and Access Token Secret for authorization in pyTweet.

Obtain CA Certificate for Official Twitter API

In addition to API keys, you'll need a .cert file to access the Official Twitter API. You can obtain this by copying and pasting the content from http://curl.haxx.se/ca/cacert.pem. Save the content to a file named *api.twitter.cer* in the pyTweet directory. [2]

Building a Network

This capability builds a data set based on an initial seed of users or topics. First a user loads a list of seed user names (Twitter @handles) into the sampling method. Then the sampling method *hops* to the next group of users based on a set of rules defined by the sampling parameters.

Breadth-first Search by Explicit Relationships

This capability builds a data set based on an initial seed of user screen names. After the names are read into the program, the sampling method them *hops* to the next group of users based on friendships, followers, mentions, or replies. The data collected is stored as one of two flavors of JSON files: user information JSON files and user timeline JSON files. The user information JSON files contain account information for a single user and are given the name **userInfo_*uuid*.json**. Similarly the timeline JSON files contain a timeline for a single user and are named **timeline_*uuid*.json**, where the UUID matches the one in **userInfo_*uuid*.json**. The directories to store profile and timeline JSON files must be specified before, otherwise default folders *profiles* and *timelines* are created in the current directory. For details about the metadata returned with the JSONs, please see the Developer's site https://dev.twitter.com/overview/api/tweets and https://dev.twitter.com/overview/api/users.

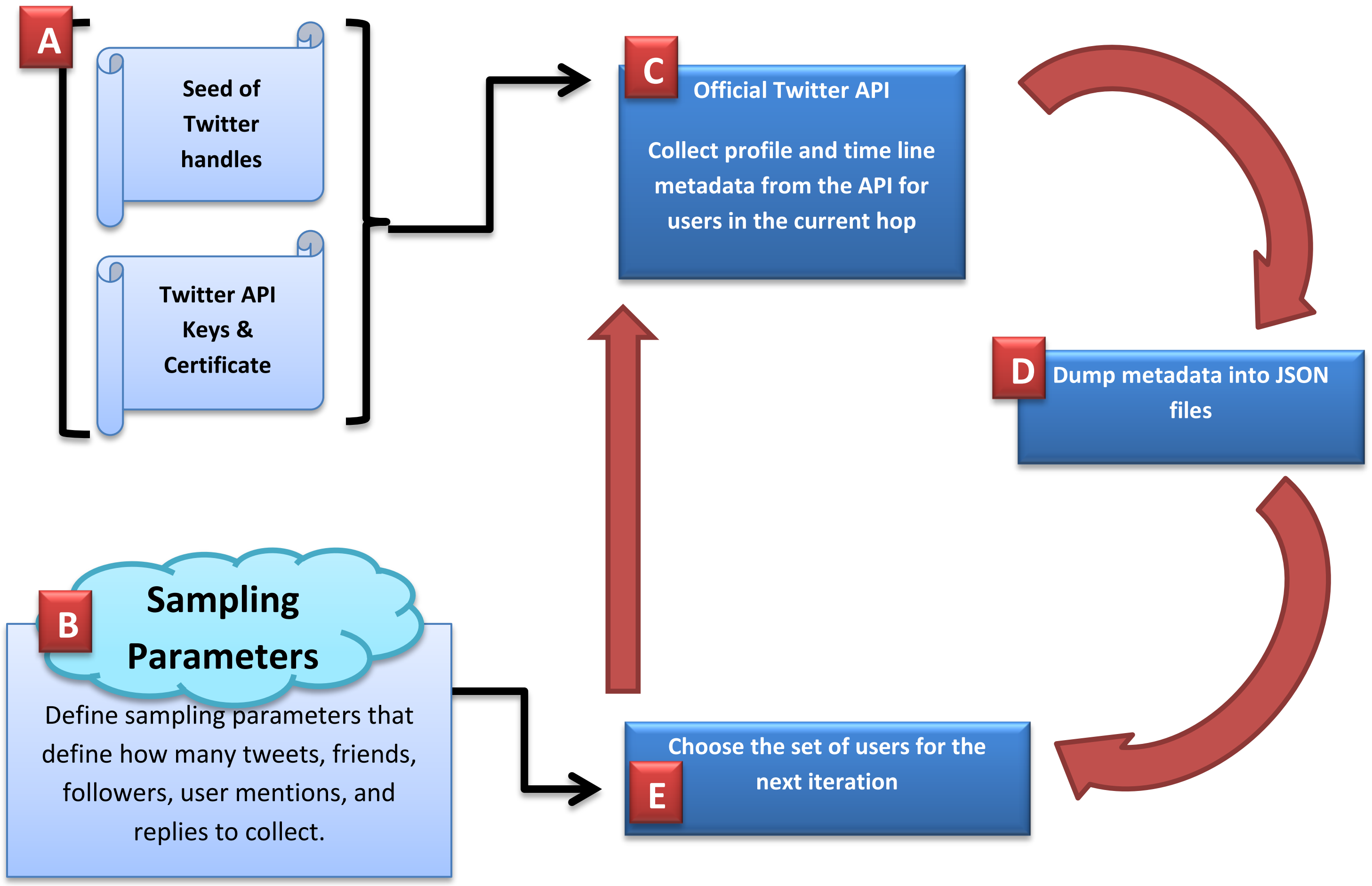
Note that sometimes the timeline JSON contain only the word 'null' if no timeline can be obtained. Keep the null timeline files so that you don't waste API calls looking for a timeline that may not exist. The existing timelines and user JSON files are routinely checked to avoid making redundant calls within the examples listed in the table. Also be aware that some attributes of timelines and profiles can change over time (i.e. retweet count, screenname). You may find it helpful to refer to the field *DOC*, which is the time the profile or timeline was collected.

The diagram in *Figure 1* illustrates the process of breadth-first sampling. Before sampling, the user must first complete **Steps A and B**:

A. Define the initial seed of Twitter handles and API keys. Refer to the *Getting Started* section to learn more.
B. Define the sampling parameters. There are three general objects used for sampling: the date timeline windows start, expansion limits, and collection limits. The expansion limits define how many friends, followers, replies, and user mentions to include in the next hop of users. They also define the stopping criteria of sampling: maximum number of hops and maximum amount of data (in GB). If a user doesn't want to impose a limit on one of these features they can simply define it as a *None* object. The collection limit parameter allows users to collect friends and followers, without expanding them by including them in the next iteration of seed.
C. Connect to the Official Twitter API and request data. We will receive mainly profile and timeline metadata.
D. Save the timeline and profile metadata as JSON files.
E. Based on the expansion limits, define the next set of users to collect metadata from. Redundant calls are avoided.

Steps C-E are iterative, and continue until the data or hop limit has been reached. Refer the file */pyTweet/examples/breadth_first_search_example.py* to see an implementation of this code.

Figure 1: Diagram of breadth-first search by explicit relationships


A
Seed of Twitter handles
Twitter API Keys & Certificate
C
Official Twitter API
Collect profile and time line metadata from the API for users in the current hop
D
Dump metadata into JSON files
B
Sampling Parameters
Define sampling parameters that define how many tweets, friends, followers, user mentions, and replies to collect.
Choose the set of users for the next iteration
E

API Wrapper Functions

In the case that breadth-first does not meet your sampling needs, the API wrappers can be used to create your own sampling methods. Note that rate limit status checking is built into these wrappers. Note that the field *DOC*, known as date of collection, is not returned in objects retrieved from these methods.

get_twitter_certificate()
This function gets the location of Twitter API Certificate if it is stored in pyTweet's directory
@return Filename of Twitter API certificate, including its path

get_authorization(twitter_keys)
This function obtains an authorization object for accessing the Official Twitter API.
@param twitter_keys - Dictionary object containing 'API_KEY', 'API_SECRET', 'ACCESS_TOKEN', 'ACCESS_TOKEN_SECRET'

@return OAUTH - Authorization object required for remaining pyTweet collection functions

get_rate_limit_status(type, proxies, auth)
This function returns the remaining and reset seconds.
@param type - Type of API call: "timeline", "friends", "followers", "search_tweets", "search_users", "retweets", or "users"
@param proxies - proxy dictionary, ex. {'http': 'http://%s:%s' % (HOST, PORT), 'https': 'http://%s:%s' % (HOST, PORT)}
@param auth - Twitter application authentication, see the get_authorization method
@return (reset, remaining)

check_rate_limit_status(min_calls, type, auth, proxies)
This function checks the rate limit for an API call and pauses as specified by the API
@param min_calls - minimum number of calls left before pausing
@param type - type of call: "timeline", "friends", "followers", "search_tweets", "search_users", "retweets", or "users"
@param proxies - proxy dictionary, ex. {'http': 'http://%s:%s' % (HOST, PORT), 'https': 'http://%s:%s' % (HOST, PORT)}
@param auth - Twitter application authentication, see the get_authorization method

user_lookup_usernames(user_list, proxies, auth)
Look up user information for a list of usernames. If a user's account has been deleted then it will not be returned in the .json of user information. We can request information for up to 100 users at a time.
@param user_list - list of usernames
@param proxies - proxy object, ex. {'http': 'http://%s:%s' % (HOST, PORT), 'https': 'http://%s:%s' % (HOST, PORT)}
@param auth - Twitter application authentication, see the get_authorization method
@return user_info - list of JSON user objects that could be returned.

user_lookup_userids(user_list, proxies, auth)
Look up user information for a list of user IDs. If a user's account has been deleted then it will not be returned in the .json of user information. We can request information for up to 100 users at a time.
@param user_list - list of user ids
@param proxies - proxy object, ex. {'http': 'http://%s:%s' % (HOST, PORT), 'https': 'http://%s:%s' % (HOST, PORT)}
@param auth - Twitter application authentication, see the get_authorization method
@return user_info - JSON list of user information that could be retrieved

get_user_friends(user_id, proxies, auth, limit=None)
Look up the IDs of all of a user's friends (people they follow), and return them in a list. Find up to 5000 friends per request.
@param user_id

@param limit - limit to number of friends to collect. Set to None to get all friends - this is the default
@param proxies - proxy object, ex. {'http': 'http://%s:%s' % (HOST, PORT), 'https': 'http://%s:%s' % (HOST, PORT)}
@param auth - Twitter application authentication, see the get_authorization method
@return friends_list - list of user's friends' IDs

get_user_followers(user_id, proxies, auth, limit=None)
Look up the IDs of all of a user's followers (people who follow them), and return them in a list of json objects.
@param user_id - User ID
@param limit - limit to number of friends to collect. Set to None to get all friends - this is the default
@param proxies - proxy object, ex. {'http': 'http://%s:%s' % (HOST, PORT), 'https': 'http://%s:%s' % (HOST, PORT)}
@param auth - Twitter application authentication, see the get_authorization method
@return followers_list - list of user's followers' IDs

convert_twitter_date(twitter_date)
Convert Twitter date
@param twitter_date - date of creation in Twitter's format
@return converted_twitter_date - Python date object

collect_user_timeline(USER, USER_type, start_date, proxies, auth)
Find timeline of a user occuring after start_date.
@param USER - Can be either a Twitter user ID (numeric), Twitter user ID (string), or Twitter screen name.
@param USER_type - specifies whether USER is an user ID or a screen name, enter either 'user_id' or 'screen_name'
@param start_date- start of timeline segment to collect
@param proxies   - proxy dictionary, ex. {'http': 'http://%s:%s' % (HOST, PORT), 'https': 'http://%s:%s' % (HOST, PORT)}
@param auth - Twitter application authentication, see the get_authorization method
@return timeline - timeline dictionary of user, None if one doesn't exist

pull_timeline_entitites(timeline, type, limit=None)
Pull fields out of a timeline array into a list.
@param timeline - A Twitter timeline that has been loaded into Python (typically a dictionary format)
@param type - Specify 'user_mentions' or 'in_reply_to_user_id' to get from the timelines. Example, type = ['text', 'geo']
@param limit - Limit of entities to collect from timeline. Default is None which means to collect all
@return - List that contains the specified field, ex. List of in_reply_to_user_id.

References

[1] "Anaconda." *Scientific Python Distribution*. Web. 26 June 2015.

[2] "Twitter Developers." *Twitter Developers*. Web. 26 June 2015.