Package 'gatherTweet'

Claudia Kann

November 29, 2022

Version 1.0
Date November 29, 2022
Title A Python Package for Collecting Social Media Data on Online Events
Author Claudia Kann (ckann@caltech.edu)
Maintainer Claudia Kann (ckann@caltech.edu)
Depends
Description
License GPL-3
Repository `https://github.com/ckann10/gatherTweet`
Date/Publication Nov. 29, 2022

Contents

Introduction 2

Generating Events 2
- From Excel File . . . 2
  - upload_from_excel . . . 3
  - upload_from_file_structure . . . 3
- Manually . . . 3
  - TwitterActivity . . . 3
  - TwitterKeyPair . . . 4
  - add_activity . . . 4
  - add_key . . . 4
  - add_keyWords . . . 4
  - add_timing . . . 4
  - add_location . . . 5
  - print_protests . . . 5

Pulling Data from Social Media 5
- Main Functions . . . 5
  - get_tweets . . . 5
- Manually . . . 7
  - version_2_setup . . . 7
  - get_core_users . . . 7
  - get_core_timeline . . . 7
  - get_echo_users . . . 8
  - get_echo_timeline . . . 8
  - get_influence_users . . . 8
  - get_influence_timeline . . . 8
  - get_ids . . . 8
  - pull_tweets . . . 8

Checking Data 8
  - number_of_tweets . . . 8

Reading Data 9
read_tweets 9

Introduction

This manual will walk you through the basic functionality of the package 'gatherTweet' designed to collect Twitter networks of dynamic events. Throughout, the code included should be sufficient to get you started. This package is created to be used with the Twitter Version 2 API as of September 2022.

Generating Events

There are two main ways to import a series of activities that make up a event. The first, which is recommended for beginners is using the excel sheet "event_template.xlsx" found in the repo. The second is to manually import the information into the event structure. Regardless, the first step is to instantiate your event object:

```
import gatherTweet as tw
event = tw.TwitterEvent(name,
                        base_directory = '',
                        separator = 'CityTown')
```

`name`	name for the event
`base_directory`	path to where all of the data will be stored
`separator`	describes how the activities are split in the data saving structure, it can be any of the activity objects. For geographical differences 'CityTown' makes the most sense.

From Excel File

To fill out the excel sheet, you must include the values below. The bolded values are optional:

- activities:
  - ID: unique identification string
  - starting_date: beginning date of the activity within the event in the format dd mm yyyy hh:mm:ss
  - ending_date: ending date of the activity within the event in the format dd mm yyyy hh:mm:ss
  - CityTown: City or Town where you want to search for individuals in the core
  - StateTerritory: State or Territory where you want to search for individuals in the nucleaus
  - Date: Date of the activity in dd mm yyyy format
  - BestGuess: best guess of the size of the activity, if unknown, write 0
  - **period_start**: start of the event (when you want the timeline gathering to start) if different from the rest of the activities in the event. In the format dd mm yyyy hh:mm:ss
  - **period_end**: end of the event (when you want the timeline gathering to end) if different from the rest of the activities in the event. In the format dd mm yyyy hh:mm:ss
- keywords:
  - keywords: list of keywords you want to use to identify the core. Can be written individually or in Twitter accepted format. For example, it is equivalent to have:

    George Floyd
    GeorgeFloyd
    vs.
    (GeorgeFloyd OR (George Floyd))

- time span:
  - start time: begining of the timeline gathering period in dd mm yyyy hh:mm:ss format
  - end time: end of the timeline gathering period in dd mm yyyy hh:mm:ss format
- keys: Twitter key and secret keys, make sure not to share these with other individuals
- location: coordinates to find bounding box for each location, a row needs to be added for each unique CityTown-StateTerritory pair found in the activities tab

upload_from_excel

Once the excel file is filled in, it should be saved in the directory `dirr` as specified above. The built in function can then be used to populate the event object:

```
event.upload_from_excel(path = 'event_template.xlsx')
```

`path`	this should be the path to the excel file. If the path given is not an .xlsx file it will replace it with `base_directory + 'event_template.xlsx'`. If the path is `'event_template.xlsx'`, it also assumes its in the `base_directory`

upload_from_file_structure

If the data has already been pulled from Twitter but postprocessing needs to occur, the entire event may not be necessary. In this case, the function `upload_from_file_structure` can be used in order to populate the event sufficiently to work with the pulled data

```
 event.upload_from_file_structure()
```

Manually

The event can also be added into the object manually. In this case, all of the information must be added through a series of functions.

TwitterActivity

TwitterActivity objects must be created for each activity within the event:

```
activity = tw.TwitterActivity(ID, starting_date, ending_date,
                              CityTown, StateTerritory, Date, BestGuess)
```

`ID`	unique identification string
`starting_date`	beginning date of the activity within the event in the format dd mm yyyy hh:mm:ss
`ending_date`	ending date of the activity within the event in the format dd mm yyyy hh:mm:ss
`CityTown`	City or Town where you want to search for individuals in the core
`StateTerritory`	State or Territory where you want to search for individuals in the core
`Date`	Date of the activity in dd mm yyyy format
`BestGuess`	best guess of the size of the activity, if unknown, write 0

If you want to pull a separate full timeline for individuals associated with this activity, that information must be added as:

```
activity1.add_timing(period_start, period_end)
```

`period_start`	start of the event (when you want the timeline gathering to start) if different from the rest of the activities in the event. In the format dd mm yyyy hh:mm:ss
`period_end`	end of the event (when you want the timeline gathering to end) if different from the rest of the activities in the event. In the format dd mm yyyy hh:mm:ss

TwitterKeyPair

For each set of Twitter keys, a TwitterKeyPair object must be created:

```
key1 = tw.TwitterKeyPair(key, secret)
```

`key`	Twitter developer key as a string
`secret`	Twitter developer secret key as a string

More information for getting these credentials can be found in the Twitter Documentation

add_activity

```
event.add_activity(activities)
```

`activities`	either a TwitterActivity object or a list of TwitterActivity objects

add_key

```
event.add_key(keypair)
```

`keypair`	either a TwitterKeyPair object or a list of TwitterKeyPair objects

add_keyWords

```
event.add_keyWords(words)
```

`words`	list of keywords you want to use to identify the core. Can be written individually or in Twitter accepted format. For example, it is equivalent to have: George Floyd / GeorgeFloyd / vs. / (GeorgeFloyd OR (George Floyd))

add_timing

```
event.add_timing(start, end)
```

`start`	begining of the timeline gathering period in dd mm yyyy hh:mm:ss format
`end`	end of the timeline gathering period in dd mm yyyy hh:mm:ss format

add_location

```
event.add_location(CityTown, StateTerritory, west, south, east, north)
```

`CityTown`	list of CityTown entries that occur in the activities
`StateTerritory`	list of StateTerritories associate with CityTown list
`west`	list of west latitude for bounding box of CityTowns
`south`	list of south longitude for bounding box of CityTowns
`east`	list of east latitude for bounding box of CityTowns
`north`	list of north longitude for bounding box of CityTowns

print_protests

```
protest_ids = event.print_protests()
```

`protest_ids`	a dataframe of the activities with relevant information. Run after generating the event in order to remove incomplete entries and create a check of whether the event is presenting as expected.

Pulling Data from Social Media

Once the event is created, the user can begin pulling the data from Twitter. The main function used here is `get_tweets`, the rest are used within it but can be accessed by the precocious user. In order to use any of these functions you must have a event object, the function is then called as `event.pull.function()`.

Main Functions

get_tweets

```
event.pull.get_tweets(event,
            types = [],
            max_results = 500,
            tweets_per_file = 1000,
            expansions = ['author_id', 'in_reply_to_user_id'],
            tweetfields = ['author_id','created_at', 'geo',
            'entities','public_metrics', 'text','referenced_tweets'])
```

`event`	TwitterEvent object types: a list of the types of Tweets and users you want to collect, options are: [‘Core’, ‘CoreTimeline’, ‘Echos’, ‘EchosTimeline’, ‘Influences’, ‘InfluencesTimeline’]. If left as an empty list, all will be evaluated. In order to run the Timeline versions, all activities are checked to make sure they have the base version. For ‘Echos’ and ‘Influences’ it checks that a core exists. Everything but the ‘Core’ can restart after being interrupted with minimal redundancy.
`max_results`	number of tweets to attempt to pull in each query, must be an integer between 1 and 500
`tweets_per_file`	number of tweets to save per file before beginning a new file
`expansions`	refers to which of the tweetfields the user would like more information on, taken from Twitter Documentation are:

author_id	*Returns a user object representing the Tweet's author*
referenced_tweets.id	*Returns a Tweet object that this Tweet is referencing (either as a Retweet, Quoted Tweet, or reply)*
in_reply_to_user_id	*Returns a user object representing the Tweet author this requested Tweet is a reply of*
attachments.media_keys	*Returns a media object representing the images, videos, GIFs included in the Tweet*
attachments.poll_ids	*Returns a poll object containing metadata for the poll included in the Tweet*
geo.place_id	*Returns a place object containing metadata for the location tagged in the Tweet*
entities.mentions.username	*Returns a user object for the user mentioned in the Tweet*
referenced_tweets.id.author_id	*Returns a user object for the author of the referenced Tweet*

`tweetfields` the values within each tweet to be returned from each call are taken from Twitter Documentation:

id (default)	*The unique identifier of the requested Tweet.*
text (default)	*The actual UTF-8 text of the Tweet. See twitter-text for details on what characters are currently considered valid.*
attachments	*Specifies the type of attachments (if any) present in this Tweet.*
author_id	*The unique identifier of the User who posted this Tweet.*
context_annotations	*Contains context annotations for the Tweet.*
conversation_id	*The Tweet ID of the original Tweet of the conversation (which includes direct replies, replies of replies).*
created_at	*Creation time of the Tweet.*
entities	*Entities which have been parsed out of the text of the Tweet. Additionally see entities in Twitter Objects.*
geo	*Contains details about the location tagged by the user in this Tweet, if they specified one.*
in_reply_to_user_id	*If the represented Tweet is a reply, this field will contain the original Tweet's author ID. This will not necessarily always be the user directly mentioned in the Tweet.*
lang	*Language of the Tweet, if detected by Twitter. Returned as a BCP47 language tag.*
non_public_metrics	*Non-public engagement metrics for the Tweet at the time of the request.*
organic_metrics	*Engagement metrics, tracked in an organic context, for the Tweet at the time of the request.*

possibly_sensitive	*This field only surfaces when a Tweet contains a link. The meaning of the field doesn't pertain to the Tweet content itbut instead it is an indicator that the URL contained in the Tweet may contain content or media identified as sensitive content.*
promoted_metrics	*Engagement metrics, tracked in a promoted context, for the Tweet at the time of the request.*
public_metrics	*Public engagement metrics for the Tweet at the time of the request*
referenced_tweets	*A list of Tweets this Tweet refers to. For example, if the parent Tweet is a Retweet, a Retweet with comment (also known as Quoted Tweet) or a Reply, it will include the related Tweet referenced to by its parent.*
reply_settings	*Shows you who can reply to a given Tweet. Fields returned are "everyone", "mentioned_users", and "followers".*
source	*The name of the app the user Tweeted from.*
withheld	*When present, contains withholding details for withheld content.*

Manually

All of these functions use the inputs in the Main Function section, refer above for help.

version_2_setup

```
APIs, PARAMS = event.pull.version_2_setup(event,
                    max_results = 500,
                    expansions = ['author_id', 'in_reply_to_user_id'],
                    tweetfields = ['author_id','created_at', 'geo',
                              'entities','public_metrics', 'text',
                              'referenced_tweets'])
```

`APIs`	list of connections to Twitter using TwitterAPI and the key and secret keys provided
`PARAMS`	dictionary of parameters that are fed to

get_core_users

```
event.pull.get_core_users(event,
                                PARAMS,
                                APIs,
                                tweets_per_file = 1000)
```

get_core_timeline

```
event.pull.get_core_timeline(event,
                                   PARAMS,
                                   APIs,
                                   tweets_per_file = 1000)
```

get_echo_users

```
event.pull.get_echo_users(event,
                          PARAMS,
                          APIs,
                          tweets_per_file = 1000):
```

get_echo_timeline

```
event.pull.get_echo_timeline(event,
                             PARAMS,
                             APIs,
                             tweets_per_file = 1000):
```

get_influence_users

```
event.pull.get_influence_users(event,
                               PARAMS,
                               APIs,
                               tweets_per_file = 1000):
```

get_influence_timeline

```
event.pull.get_influence_timeline(event,
                                  PARAMS,
                                  APIs,
                                  tweets_per_file = 1000):
```

get_ids

```
event.pull.get_ids(path)
```

pull_tweets

```
event.pull.pull_tweets(PARAMS,
                       datestart,
                       APIs,
                       api_i,
                       file_prefix,
                       tweets_per_file,
                       echo = False,
                       influence = False):
```

Checking Data

number_of_tweets

```
event.check.number_of_tweets(activities,
                             users,
                             base_directory = "",
                             separator = 'CityTown',
                             timeline = False)
```

`activities`	a list of the activities that you want to count the tweets in
`users`	the types of users you want to count tweets from, options are [‘Core’, ‘Echo’, ‘Influence’]
`base_directory`	where to begin searching for the data
`separator`	describes how the activities are split in the data saving structure, it can be any of the activity objects. For geographical differences ‘CityTown’ makes the most sense.
`Timeline`	Boolean of whether to count tweets in the timeline or not, if not, only Nucleaus original tweets will be counted

Reading Data

read_tweets

```
event.read.read_tweets(event, users = "core")
```

`event`	TwitterEvent object
`users`	which tweets do you want to look at, can be a list or an individual, options are: “Core”, “CoreTimeline”, “EchoTimeline”, “InflunceTimeline”

A csv of all the relevant tweets will be added to the `event.base_directory`