Package 'CooRTweet'

September 5, 2023

Version 1.5.0

Date 2023-08-16

Type Package

Title Coordinated Networks Detection on Social Media

Description Detects a variety of coordinated actions on Twitter and outputs the network of coordinated users along with related information.

Author Nicola Righetti [aut, cre] (<https://orcid.org/0000-0002-9257-5113>),
Paul Balluff [aut] (<https://orcid.org/0000-0001-9548-3225>)

Maintainer Nicola Righetti <nicola.righetti@univie.ac.at>

URL https://github.com/nicolarighetti/CooRTweet

BugReports https://github.com/nicolarighetti/CooRTweet/issues

License MIT + file LICENSE

Encoding UTF-8

RoxygenNote 7.2.3

Imports data.table, tidytable, RcppSimdJson, Matrix, lubridate,
igraph, stringi, textreuse

Suggests knitr, rmarkdown, testthat (>= 3.0.0)

Config/testthat/edition 3

Depends R (>= 2.10)

LazyData true

VignetteBuilder knitr

NeedsCompilation no

Repository CRAN

Date/Publication 2023-09-05 11:40:02 UTC

R topics documented:

`detect_coordinated_groups` *detect_coordinated_groups*

Description

Function to detect coordinated behaviour based on content groups. See details.

Usage

```
detect_coordinated_groups(x, time_window = 10, min_repetition = 2)
```

Arguments

`x`	a data.table with the columns: `object_id` (uniquely identifies coordinated content), `id_user` (unique ids for users), `content_id` (id of user generated content), `timestamp_share` (integer)
`time_window`	the number of seconds within which shared contents are to be considered as coordinated (default to 10 seconds).
`min_repetition`	the minimum number of repeated coordinated actions a user has to perform (default to 2 times)

Details

The function groups the data by `object_id` (uniquely identifies coordinated content) and calculates the time differences between all `content_id` (ids of user generated contents) within their groups. It then filters out all `content_id` that are higher than the `time_window` (in seconds). It returns a `data.table` with all IDs of coordinated contents. The `object_id` can be for example: hashtags, IDs of tweets being retweeted, or URLs being shared.

Value

a data.table with ids of coordinated contents. Columns: `object_id`, `id_user`, `id_user_y`, `content_id`, `content_id_y`, `timedelta`. The `id_user` and `content_id` represent the "older" data points, `id_user_y` and `content_id_y` represent the "newer" data points. For example, User A retweets from User B, then User A's content is newer (i.e., `id_user_y`).

`detect_similar_text` *detect_similar_text*

Description

This function detects coordinated cotweets, i.e. pairs of social media posts that are similar in terms of their text and were posted within a short time window.

Usage

```
detect_similar_text(
  x,
  min_repetition = 2,
  time_window = 10,
  min_similarity = 0.8,
  similarity_function = textreuse::jaccard_similarity,
  tokenizer = textreuse::tokenize_ngrams,
  minhash_seed = NULL,
  minhash_n = 200
)
```

Arguments

`x`	A data.table with the following columns: • content_id: The ID of the content (e.g. a tweet ID) • object_id: The text of the social media post • id_user: The ID of the user who shared the content • timestamp_share: The timestamp when the content was shared
`min_repetition`	the minimum number of repeated coordinated actions a user has to perform (defaults to 2 times)
`time_window`	The maximum time difference between two posts in order for them to be considered coordinated cotweets (defaults to 10 seconds).

`min_similarity` The minimum similarity score between two posts in order for them to be considered coordinated cotweets (defaults to 0.8).

`similarity_function` The function that is used to calculate the similarity between two tweets. The default function is Jaccard Similarity (see: jaccard_similarity).

`tokenizer` The function that is used to tokenize the text of the tweets. The default function is the tokenize_ngrams function.

`minhash_seed` The seed that is used to generate the minhash signatures. If NULL, a random seed will be used.

`minhash_n` The number of minhash signatures that are used (see `textreuse` package for details).

Details

Uses the textreuse package to compare each post with each other and determine their text similarity. Use the `reshape_tweets()` function with `intent = "cotweet"` parameter to prepare your data.

Value

A data.table with the following columns:

- content_id: The ID of the first post
- content_id_y: The ID of the second post
- id_user: The ID of the user who shared the first post
- id_user_y: The ID of the user who shared the second post
- timestamp_share: The timestamp when the first post was shared
- timestamp_share_y: The timestamp when the second post was shared
- similarity_score: The similarity score between the two posts
- time_delta: The time difference between the two posts

`filter_min_repetition` *Filter the result by minimum repetition.*

Description

This private function filters the result by the minimum number of repetitions required.

Usage

```
filter_min_repetition(x, result, min_repetition)
```

Arguments

x	A data table from a coordination detection function
result	A data table containing the result data.
min_repetition	The minimum repetition threshold. Users with repetition count greater than this threshold will be retained.

Value

A data table with filtered rows based on the specified minimum repetition.

---

generate_network *generate_network*

Description

Take the results of coordinated content detection and generate a network from the data. This function generates a two-mode (bipartite) incidence matrix first, and then projects the matrix to a weighted adjacency matrix.

Usage

```
generate_network(x, intent = c("users", "content", "objects"))
```

Arguments

x	a data.table (result from detect_coordinated_groups) with the Columns: object_id, id_user, id_user_y, content_id, content_id_y, timedelta
intent	the intended network. The option "users" generates a network of users who are connected over the same content that they share (default). Option "content" generates a network based on content ids. Option "objects" generates a network of the coordinated content (object_id) that is connected via the users.

Value

A weighted, undirected network (igraph object) where the vertices (nodes) are users (or content_ids) and edges (links) are the membership in coordinated groups (object_id)

group_stats *group_stats*

Description

Calculate coordinated group statistics: total unique users per group, total posts in per group, average time delta per group

Usage

```
group_stats(x)
```

Arguments

x	a result `data.table` generated by `detect_coordinated_groups`

Details

This helper function gives you a summary of the coordinated groups.

Value

a `data.table` with summary statistics for each group

load_many_tweets_json *load_many_tweets_json*

Description

EXPERIMENTAL. Batched version of load_tweets_json with control over retained columns. Not as efficient as load_tweets_json but requires less memory. Wrapper of the function fload

Usage

```
load_many_tweets_json(
  data_dir,
  batch_size = 1000,
  keep_cols = c("text", "possibly_sensitive", "public_metrics", "lang",
    "edit_history_tweet_ids", "attachments", "geo"),
  query = NULL,
  query_error_ok = TRUE
)
```

Arguments

data_dir	string that leads to the directory containing JSON files
batch_size	integer specifying the number of JSON files to load per batch. Default: 1000
keep_cols	character vector with the names of columns you want to keep. Set it to NULL to only retain the required columns. Default: keep_cols = c("text", "possibly_sensitive", "public_metrics", "lang", "edit_history_tweet_ids", "attachments", "geo")
query	(string) JSON Pointer query passed on to fload (optional). Default: NULL
query_error_ok	(Boolean) stop if query causes an error. Passed on to fload (optional). Default: FALSE

Details

Unlike load_tweets_json this function loads JSON files in batches and processes each batch before loading the next batch. You can specify which columns to keep, which in turn requires less memory. For example, you can decide not to keep the "text column, which requires quite a lot of memory.

Value

a data.table with all tweets loaded

---

load_tweets_json *load_tweets_json*

Description

Very efficient and fast way to load tweets stored in JSON files. Wrapper of the function fload

Usage

```
load_tweets_json(data_dir, query = NULL, query_error_ok = TRUE)
```

Arguments

data_dir	string that leads to the directory containing JSON files
query	(string) JSON Pointer query passed on to fload (optional). Default: NULL
query_error_ok	(Boolean) stop if query causes an error. Passed on to fload (optional). Default: FALSE

Details

This function is optimized to load tweets that were collected using the academicTwittr Package (Twitter API V2). It uses RcppSimdJson to load the JSON files, which is extremely fast and efficient. It returns the twitter data as is. The only changes are that the function renames the id of tweets to tweet_id, and it also deduplicates the data (by tweet_id). The function expects that the individual JSON files start with data.

Value

a data.table with all tweets loaded

`load_twitter_users_json` *load_twitter_users_json*

Description

Very efficient and fast way to load user information from JSON files. Wrapper of the function fload

Usage

```
load_twitter_users_json(data_dir, query_error_ok = TRUE)
```

Arguments

`data_dir`	string that leads to the directory containing JSON files
`query_error_ok`	(Boolean) stop if `query` causes an error. Passed on to fload (optional). Default: `TRUE`

Details

This function is optimized to load user data JSON files that were collected using the academicTwittr Package (Twitter API V2). It uses RcppSimdJson to load the JSON files, which is extremely fast and efficient. It returns the user data as is. The only changes are that the function renames the `id` of tweets to `user_id`, and it also deduplicates the data (by `user_id`). The function expects that the individual JSON files start with `user`.

Value

a data.table with all users loaded

`normalize_text` *Normalize text*

Description

Utility function that normalizes text by removing mentions of other users, removing "RT", converting to lower case, and trimming whitespace.

Usage

```
normalize_text(x)
```

Arguments

x	The text to be normalized.

Value

The normalized text.

`preprocess_tweets` *preprocess_tweets*

Description

Reformat nested Twitter data (retrieved from Twitter V2 API). Spreads out columns and reformats nested a `data.table` to a named list of unnested data.tables. All output is in long-format.

Usage

```
preprocess_tweets(
  tweets,
 tweets_cols = c("possibly_sensitive", "lang", "text", "public_metrics_retweet_count",
    "public_metrics_reply_count", "public_metrics_like_count",
    "public_metrics_quote_count")
)
```

Arguments

`tweets`	a data.table to unnest. Twitter data loaded with load_tweets_json‘.
`tweets_cols`	a character vector specifying the columns to keep (optional).

Details

Restructure your nested Twitter data that you loaded with load_tweets_json. The function unnests the following columns: `public_metrics` (likes, retweets, quotes), `referenced_tweets` (IDs of "replied to" and "retweet"), `entities` (hashtags, URLs, other users). Returns a named list with several `data.tables`, each `data.table` represents one aspect of the nested data. The function also expects that the following additional columns are present in the `data.table`: `created_at`, `tweet_id`, `author_id`, `conversation_id`, `text`, `in_reply_to_user_id`. Implicitely dropped columns: `edit_history_tweet_ids`

Value

a named `list` with 5 data.tables: tweets (contains all tweets and their meta-data), referenced (information on referenced tweets), urls (all urls mentioned in tweets), mentions (other users mentioned in tweets), hashtags (hashtags mentioned in tweets)

preprocess_twitter_users *preprocess_twitter_users*

Description

Reformat nested twitter user data (retrieved from Twitter v2 API). Spreads out columns and reformats nested `data.table` to long format.

Usage

```
preprocess_twitter_users(users)
```

Arguments

`users`	a data.table with unformatted (nested user data).

Details

Take the Twitter user data that you loaded with load_twitter_users_json and unnests the following columns: `public_metrics` and `entities`.

Value

a data.table with reformatted user data.

---

remove_hashtags *Remove hashtags*

Description

Utility function that removes hashtags from tags.

Usage

```
remove_hashtags(x)
```

Arguments

`x`	The text to be processed.

Value

The text without hashtags.

remove_loops *Remove loops from the result.*

Description

This function is a private utility function that removes loops (i.e., users sharing their own content) from the result.

Usage

```
remove_loops(result)
```

Arguments

result	The result of the previous filtering steps.

Value

The result with loops removed.

reshape_tweets *reshape_tweets*

Description

Reshape twitter data for coordination detection.

Usage

```
reshape_tweets(
  tweets,
  intent = c("retweets", "hashtags", "urls", "urls_domains", "cotweet"),
  drop_retweets = TRUE,
  drop_replies = TRUE,
  drop_hashtags = FALSE
)
```

Arguments

tweets	a named list of Twitter data (output of preprocess_tweets)
intent	the desired intent for analysis.
drop_retweets	Option passed to `intent = "cotweet"`. When analysing tweets based on text similarity, you can choose to drop all tweets that are retweets. Default: TRUE

`drop_replies` Option passed to `intent = "cotweet"`. When analysing tweets based on text similarity, you can choose to drop all tweets that are replies to other tweets. Default: TRUE

`drop_hashtags` Option passed to `intent = "cotweet"`. You can choose to remove all hashtags from the tweet texts. Default: FALSE

Details

This function takes the pre-processed Twitter data (output of preprocess_tweets) and reshapes it for coordination detection (detect_coordinated_groups). You can choose the intent for reshaping the data. Use `"retweets"` to detect coordinated retweeting behaviour; `"hashtags"` for coordinated usage of hashtags; `"urls"` to detect coordinated link sharing behaviour; `"urls_domain"` to detect coordinated link sharing behaviour at the domain level. `"cotweet"` to detect coordinated cotweeting behaviour (users posting same text). The output of this function is a reshaped `data.table` that can be passed to detect_coordinated_groups.

Value

a reshaped data.table

`russian_coord_tweets` *Pro-Government Russian Tweet Dataset*

Description

A anonymized dataset of Tweets. All IDs have been obscured using sha256 algorithm.

Usage

```
russian_coord_tweets
```

Format

`russian_coord_tweets`**:**

A data frame with 35,125 rows and 4 columns:

object_id ID of retweeted content. Twitter API calls this "referenced_tweet_id".

id_user ID of the user who tweeted. Twitter API: "author_id"

content_id Tweet ID.

timestamp_share Ingeger. Timestamp (posix time)

Source

Kulichkina (in Press).

simulate_data *simulate_data*

Description

Create a simulated input and output of detect_coordinated_groups function.

Usage

```
simulate_data(
  n_users_coord = 5,
  n_users_noncoord = 4,
  n_objects = 5,
  min_repetition = 3,
  time_window = 10
)
```

Arguments

n_users_coord	the desired number of coordinated users.
n_users_noncoord	the desired number of non-coordinated users.
n_objects	the desired number of objects.
min_repetition	the minimum number of repeated coordinated action to define two user as coordinated.
time_window	the time window of coordination.

Details

This function generates a simulated dataset with fixed numbers for coordinated users, uncoordinated users, and shared objects. You can set minimum repetition and time window and the coordinated users will "act" randomly within these restrictions.

Value

a list with two data frames: a data frame with the columns required by the function detect_ coordinated_groups (`object_id`, `id_user`, `content_id`, `timestamp_share`) and the output table of the same detect_coordinated_groups function and columns: `object_id`, `id_user`, `id_user_y`, `content_id`, `content_id_y`, `time_delta`.

user_stats *user_stats*

Description

Calculate user statistics: total posts shared, average time delta.

Usage

```
user_stats(x)
```

Arguments

x	a result data.table generated by detect_coordinated_groups

Details

With this helper function you get a summary of the users, who share coordinated content. High number of posts shared and low average time delta are indicators for highly coordinated (potentially automated) user behaviour.

Value

a data.table with summary statistics for each user

Index