Package 'CooRTweet'

January 20, 2025

Version 2.1.0

Date 2024-12-18

Type Package

Title Coordinated Networks Detection on Social Media

Description Detects a variety of coordinated actions on social media 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@uniurb.it>

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

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

License MIT + file LICENSE

Encoding UTF-8

RoxygenNote 7.3.2

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

Suggests knitr, rmarkdown, testthat (>= 3.0.0)

Config/testthat/edition 3

Depends R (>= 3.5.0)

LazyData true

VignetteBuilder knitr

NeedsCompilation no

Repository CRAN

Date/Publication 2025-01-09 00:00:06 UTC

Contents

`account_stats` *account_stats*

Description

Calculate account statistics: total posts shared, average time delta, average edge symmetry score.

Usage

```
account_stats(
  coord_graph,
  result,
  weight_threshold = c("full", "fast", "none")
)
```

Arguments

`coord_graph`	an `igraph` object generated by generate_coordinated_network
`result`	a table generated by detect_groups
`weight_threshold`	The threshold to be used for filtering the graph (options: "full", "fast", or "none").

Details

With this helper function, you can obtain summary statistics for the accounts in the network. When applied to a network for which a narrower `time_window` has been calculated using the flag_speed_share function, the summary statistics are computed separately for the full and faster networks depending on the 'weight_threshold' option. When this option is set to "full", metrics are computed on the set of nodes and edges surpassing the user-defined edge_weight threshold in the generate_coordinated_network function. Also, metrics for nodes and edges in the fastest network are returned, but they are calculated on the specified subgraph. The same applies when the 'weight_threshold' option is set to "fast". In this case, metrics are calculated on the fast subgraph. When the option is set to "null", the entire inputted graph without further subsetting is considered.

The node share count is performed thanks to the table resulting from the detect_groups function. If the user has used the optional flag_speed_share function and decides to calculate statistics on the fastest graph (by setting weight_threshold = "fast"), the share count is calculated considering only shares made in the fastest time window. Alternatively, shares in the largest time window are considered (option weight_threshold = "full" or weight_threshold = "none"). When calculating the share count, all shares made by accounts are considered, regardless of whether they are shares of posts shared in a coordinated fashion or not, according to the edge weight threshold. In other words, this is a measure of an account's activity in the time window under consideration.

Value

a data.table with summary statistics for each account

`detect_groups` *detect_groups*

Description

Function to perform the initial stage in detecting coordinated behavior. It identifies pairs of accounts that share the same objects in a time_window. See details.

Usage

```
detect_groups(
  x,
  time_window = 10,
  min_participation = 2,
  remove_loops = TRUE,
  ...
)
```

Arguments

`x`	a data.table with the columns: `object_id` (uniquely identifies coordinated content), `account_id` (unique ids for accounts), `content_id` (id of account generated content), `timestamp_share` (integer). See also reshape_tweets and prep_data.

`time_window` the number of seconds within which shared contents are to be considered as coordinated (default to 10 seconds).

`min_participation`
The minimum number of actions required for a account to be included in subsequent analysis (default set at 2). This ensures that only accounts with a minimum level of activity in the original dataset are included in subsequent analysis. It is important to distinguish this from the frequency of repeated interactions an account has with another specific account, as represented by edge weight. The edge weight parameter is utilized in the `generate_coordinated_network` function as a concluding step in identifying coordinated behavior.

`remove_loops` Should loops (shares of the same objects made by the same account within the time window) be removed? (default to TRUE).

`...` keyword arguments for backwards compatibility.

Details

This function achieves the initial stage in detecting coordinated behavior by identifying accounts who share identical objects within the same temporal window, and is preliminary to the network analysis conducted using the generate_coordinated_network function. `detect_groups` groups the data by `object_id` (uniquely identifies content) and calculates the time differences between all `content_id` (ids of account 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. For twitter data, best use reshape_tweets.

Value

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

`do_remove_loops` *Remove loops from the result.*

Description

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

Usage

```
do_remove_loops(result)
```

Arguments

`result` The result of the previous filtering steps.

Value

The result with loops removed.

`filter_min_participation` *Filter the result by minimum participation*

Description

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

Usage

```
filter_min_participation(x, result, min_participation)
```

Arguments

`x`	The original data table where a preliminary filter is applied
`result`	A data table containing the result data from calc_group_combinations.
`min_participation`	The minimum activity threshold. accounts with participation count greater than this threshold will be retained in the final 'result' table.

Value

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

`flag_speed_share` *flag_speed_share*

Description

Function to update result based on a narrower time_window.

Usage

```
flag_speed_share(x, result, min_participation, time_window)
```

Arguments

x	The original data used to run the detect_groups function.
result	A data table containing the result data from the detect_groups function.
min_participation	The minimum participation threshold. Accounts with participation count greater than this threshold will be retained (default parameter equal to the one used in the detect_groups function).
time_window	The number of seconds within which shared contents are to be considered as coordinated according to the new time_window (default parameter equal to the one used in the detect_groups function).

Details

This function identifies and marks the subset of results that match a more stringent time window.

Value

A results data table that includes an additional column set to 1 when the share corresponds with the new time_window, and 0 otherwise.

---

generate_coordinated_network *generate_coordinated_network*

Description

This function takes the results of detect_groups and generates a network from the data. It performs the second step in coordinated detection analysis by identifying users who repeatedly engage in identical actions within a predefined time window. The function offers multiple options to identify various types of networks, allowing for filtering based on different edge weights and facilitating the extraction of distinct subgraphs. See details.

Usage

```
generate_coordinated_network(
  x,
  fast_net = FALSE,
  edge_weight = 0.5,
  subgraph = 0,
  objects = FALSE
)
```

Arguments

x	a data.table (result from detect_groups) with the Columns: object_id, account_id, account_id_y, content_id, content_id_y, timedelta
fast_net	If the data.table x has been updated with the flag_speed_share function and this parameter is set to TRUE, two columns weight_full and weight_fast are created, the first containing the edge weights of the full graph, the second those of the subgraph that includes the shares made in the narrower time window.
edge_weight	This parameter defines the edge weight threshold, expressed as a percentile of the edge weight distribution within the network. This applies also to the faster network, if 'fast_net' is set to TRUE (and the data is updated using the flag_speed_share function). Edges with a weight exceeding this threshold are marked as 0 (not exceeding) or 1 (exceeding). The parameter accepts any numeric value between 0 and 1. The default value is set to "0.5", representing the median value of edge weights in the network.
subgraph	Generate and return the following subgraph (default value is 0, meaning that no subgraph is created): • If 1 reduces the graph to the subgraph whose edges have a value that exceeds the threshold given in the edge_weight parameter (weighted subgraph). • If 2 reduces the subgraph whose nodes exhibit coordinated behavior in the narrowest time window (as established with the flag_speed_share function), to the subgraph whose edges have a value that exceeds the threshold given in the edge_weight parameter (fast weighted subgraph). • If 3 reduces the graph to the subgraph whose nodes exhibit coordinated behavior in the narrowest time window established with the flag_speed_share function (fast subgraph), and the vertices adjacent to their edges. In other words, this option identifies the fastest network, along with a contextual set of accounts that shared the same objects but in the wider time window. It also add a vertex attribute color_v to facilitate further analyses or the generation of the graph plot. This attribute is 1 when for the coordinated accounts and 0 for the neighbor accounts.
objects	Keep track of the IDs of shared objects for further analysis with group_stats (default FALSE). There could be a performance impact when this option is set to TRUE, although the actual impact may vary. For smaller datasets, the difference might be negligible. However, for very large datasets, or in scenarios where optimal performance is crucial, you might experience a more significant slowdown.

Details

Two users may coincidentally share the same objects within the same time window, but it is unlikely that they do so repeatedly (Giglietto et al., 2020). Such repetition is thus considered an indicator of potential coordination. This function utilizes percentile edge weight to represent recurrent shares by the same user pairs within a predefined time window. By considering the edge weight distribution across the data and setting the percentile value p between 0 and 1, we can identify edges that fall within the top p percentile of the edge weight distribution. Selecting a sufficiently high percentile

(e.g., 0.99) allows us to pinpoint users who share an unusually high number of objects (for instance, more than 99% of user pairs in the network) in the same time window.

The graph also incorporates the contribution of each node within the pair to the pair's edge weight, specifically, the number of shared `content_id` that contribute to the edge weight. Additionally, an `edge_symmetry_score` is included, which equals 1 in cases of equal contributions from both users and approaches 0 as the contributions become increasingly unequal. The edge_symmetry_score is determined as the proportion of the unique content_ids (unique content) shared by each vertex to the total content_ids shared by both users. This score, along with the value of contributions, can be utilized for further filtering or examining cases where the score is particularly low. Working with an undirected graph, it is plausible that the activity of highly active users disproportionately affects the weight of edges connecting them to less active users. For instance, if user A shares the same objects (`object_id`) 100 times, and user B shares the same object only once, but within a time frame that matches the `time_window` defined in the parameter for all of user A's 100 shares, then the edge weight between A and B will be 100, although this weight is almost entirely influenced by the hyperactivity of user A. The `edge_symmetry_score`, along with the counts of shares by each user `user_id` and `user_id_y` (`n_content_id` and `n_content_id_y`), allows for monitoring and controlling this phenomenon.

Value

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

References

Giglietto, F., Righetti, N., Rossi, L., & Marino, G. (2020). It takes a village to manipulate the media: coordinated link sharing behavior during 2018 and 2019 Italian elections. *Information, Communication & Society*, 23(6), 867-891.

`german_elections` *German 2021 election campaign*

Description

An anonymized multi-platform and multi-modal dataset of social media messages from the 2021 German election campaign. Includes Facebook and Twitter posts.

Usage

```
german_elections
```

Format

`mmmp`**:**

A data.frame with 218,971 rows and 7 columns:

account_id character, with shorthand for platform

post_id integer

url_id integer, anonymized url contained in post

hashtag_id integer, anonymized hashtag contained in post

domain_id integer, anonymized domain of url

phash_id integer, anonymized perceptual hash of shared image

timestamp numeric, timestamp of post

Source

Righetti, N., Giglietto, F., Kakavand, A. E., Kulichkina, A., Marino, G., & Terenzi, M. (2022). Political Advertisement and Coordinated Behavior on Social Media in the Lead-Up to the 2021 German Federal Elections. Düsseldorf: Media Authority of North Rhine-Westphalia.

`group_stats` *group_stats*

Description

With this helper function, you can obtain summary statistics for the objects in the network.

Usage

```
group_stats(coord_graph, weight_threshold = c("full", "fast", "none"))
```

Arguments

`coord_graph`	A result `igraph` generated by generate_coordinated_network
`weight_threshold`	The level of the network for which to calculate the statistic. It can be "full," "fast," or "none." The first two options are applicable only if the data includes information on a faster network, as calculated with the flag_speed_share function. These options preliminarily filter the nodes based on their inclusion in the subgraph filtered by edge weight threshold ("full"), filtered by edges created in the faster time window and surpassing the edge weight threshold in that network ("fast"), or apply to the unfiltered graph ("none").

Value

a `data.table` with summary statistics

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 accounts, 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 accounts). 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 accounts 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.

prep_data *prep_data*

Description

Function to rename columns of a given data.table. This function standardizes column names to "object_id", "account_id", "content_id", and "timestamp_share". It is useful for preparing datasets for further analysis by ensuring consistent column naming.

Usage

```
prep_data(
  x,
  object_id = NULL,
  account_id = NULL,
  content_id = NULL,
  timestamp_share = NULL
)
```

Arguments

x	A data.table or an object that can be converted into a data.table. This is the dataset whose columns will be renamed.
object_id	The current name of the column that should be renamed to "object_id". If NULL, no renaming is performed on this column.
account_id	The current name of the column that should be renamed to "account_id". If NULL, no renaming is performed on this column.
content_id	The current name of the column that should be renamed to "content_id". If NULL, no renaming is performed on this column.
timestamp_share	The current name of the column that should be renamed to "timestamp_share". The data in this column should be either in UNIX format or in a "%Y-%m-%d %H:%M:%S" format. If the data is in a different format or conversion is unsuccessful, the function stops with an error. If NULL, no renaming or conversion is performed on this column.

Details

This function allows the user to specify the current names of columns in their data.table that they wish to rename to a standard format. The function checks for each parameter and renames the corresponding column in the data.table. If the parameter is NULL, no change is made to that column. The function ensures the data input is a data.table; if not, it converts it before renaming. For the 'timestamp_share' column, the function expects the format to be either UNIX format (integer representing seconds since the Unix epoch) or "%Y-%m-%d %H:%M:%S". If the 'timestamp_share' is in a different format, the function attempts to convert it to UNIX format using base R functions.

Value

A data.table with the specified columns renamed according to the input parameters. If no renaming is required, the original data.table is returned unaltered.

Examples

```
dt <- data.table::data.table(old_object_id = 1:3, old_account_id_y = 4:6)
dt <- prep_data(dt, object_id = "old_object_id", account_id = "old_account_id_y")
```

---

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.

`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_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 (accounts posting same text). The output of this function is a reshaped `data.table` that can be passed to detect_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".
account_id ID of the user who tweeted. Twitter API: "author_id"
content_id Tweet ID.
timestamp_share Ingeger. Timestamp (posix time)

Source

Kulichkina, A., Righetti, N., & Waldherr, A. (2024). Protest and repression on social media: Pro-Navalny and pro-government mobilization dynamics and coordination patterns on Russian Twitter. New Media & Society. https://doi.org/10.1177/14614448241254126

`simulate_data` *simulate_data*

Description

Create a simulated input and output of `detect_groups` function.

Usage

```
simulate_data(
  approx_size = 200,
  n_accounts_coord = 5,
  n_accounts_noncoord = 4,
  n_objects = 5,
  min_participation = 3,
  time_window = 10,
  lambda_coord = NULL,
  lambda_noncoord = NULL
)
```

Arguments

approx_size	the approximate size of the desired dataset. It automatically calculates the lambdas passed to rpois(), which is the expected rate of occurrences. It only works when lambda_coord and lambda_noncoord are NULL (default).
n_accounts_coord	the desired number of coordinated accounts.
n_accounts_noncoord	the desired number of non-coordinated accounts.
n_objects	the desired number of objects.
min_participation	the minimum number of repeated coordinated action to define two accounts as coordinated.
time_window	the time window of coordination.
lambda_coord	lambda parameter for coordinated accounts passed to rpois(), which is the expected rate of occurrences (higher lambda means more coordinated shares).
lambda_noncoord	lambda parameter for non-coordinated accounts passed to rpois(), which is the expected rate of occurrences (higher lambda means more non-coordinated shares).

Details

This function generates a simulated dataset with fixed numbers for coordinated accounts, uncoordinated accounts, and shared objects. The user can set minimum participation and time window parameters and the coordinated accounts will "act" randomly within these restrictions.

The size of the resulting dataset can be adjusted using the approx_size parameter, and the function will return approximately a dataset of the required size. Additionally, the size of the dataset can also be adjusted with the lambda_coord and lambda_noncoord parameters. These correspond to the lambda for the rpois Poisson distribution used to populate the coordination matrix. If lambda is between 0.0 and 1.0, the dataset will be smaller compared to choosing lambdas greater than 1. The approx_size parameter also serves to set the lambda of the rpois function in a more intuitive way.

Value

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

Examples

```
# Example usage of simulate_data
## Not run:
set.seed(123) # For reproducibility
simulated_data <- simulate_data(
  n_accounts_coord = 100,
```

```
  n_accounts_noncoord = 50,
  n_objects = 20,
  min_participation = 2,
  time_window = 10
)

# Extract input
input_data <- simulated_data[[1]]

# Extract output and keep coordinated actors.
# This is expected correspond to CooRTweet results from `detect_group`
simulated_results <- simulated_data[[2]]
simulated_results <- simulated_results[simulated_results$coordinated == TRUE, ]
simulated_results$coordinated <- NULL

# Run CooRTweet using the input_data and the parameters used for simulation
results <- detect_groups(
  x = input_data,
  time_window = 10,
  min_participation = 2
)

# Sort data tables and check whether they are identical
data.table::setkeyv(simulated_results, names(simulated_results))
data.table::setkeyv(results, names(simulated_results))

identical(results, simulated_results)

## End(Not run)
```

Index