Package 'SocialMediaLab'

November 14, 2015

Type Package

Title Tools for collecting social media data and generating networks for analysis

Imports
tm,stringr,twitteR,RCurl,bitops,rjson,plyr,igraph,Rfacebook,Hmisc,data.table,httpuv,instaR,methods,httr

Version 0.18.0

Date 2015-11-14

Author Tim Graham & Robert Ackland

Maintainer Tim Graham <timothy.graham3@uq.net.au>

Description A suite of tools for collecting and constructing networks from social media data. Provides easy-to-use functions for collecting data across popular platforms (Instagram, Facebook, Twitter, and YouTube) and generating different types of networks for analysis.

R topics documented:

SocialMediaLab-package

Collection and network analysis of social media data

Description

The goal of the SocialMediaLab package is to provide a suite of easy-to-use tools for collecting data from social media sources (Instagram, Facebook, Twitter, and Youtube) and generating different types of networks suited to Social Network Analysis (SNA) and text analytics. It offers tools to create unimodal, multimodal, semantic, and dynamic networks. It draws on excellent packages such as **twitteR**, **instaR**, **Rfacebook**, and **igraph** in order to provide an integrated 'work flow' for collecting different types of social media data and creating different types of networks out of these data. Creating networks from social media data is often non-trivial and time consuming. This package simplifies such tasks so users can focus on analysis.

SocialMediaLab uses a straightforward S3 class system. Data collected with this package produces `data.table` objects (extension of class `data.frame`), which are assigned the class `dataSource`. Additionally, `dataSource` objects are assigned a class identifying the source of data, e.g. `facebook` or `youtube`. In this way, `dataSource` objects are fast, easy to work with, and can be used as input to easily construct different types of networks. For example, the function `CollectDataTwitter` is used to collect Twitter data, and the resulting object can be supplied to a number of network generation functions (e.g. `CreateBimodalNetwork`).

Author(s)

Timothy Graham & Robert Ackland

Maintainer: Timothy Graham <timothy.graham3@uq.net.au>

AuthenticateWithFacebookAPI

Facebook API Authentication

Description

OAuth token based authentication with the Facebook API, with caching options for automatic authentication (i.e. avoid using the browser).

Usage

```
AuthenticateWithFacebookAPI(appID, appSecret, extended_permissions, useCachedToken)
```

Arguments

`appID`	character string specifying the 'App ID' of the Facebook app used for authentication.
`appSecret`	character string specifying the 'API Secret' associated with the Facebook App used for authentication.

`extended_permissions`
: logical. If `TRUE` then behaves as described in package 'Rfacebook': the token will give access to some of the authenticated user's private information (birthday, hometown, location, relationships) and that of his/her friends, and permissions to post status updates as well as to access checkins, likes, and the user's newsfeed. If FALSE, token will give access only to public information. Note that updateStatus will only work for tokens with extended permissions.

`useCachedToken`
: logical. If `TRUE` then this function will look for a saved token in the current working directory (name of token file must be `fb_oauth`). If `fb_oauth` token is not found, then it will create a token and save it to current working directory (i.e. for future use).

Details

In order to collect data from Facebook, the user must first authenticate with Facebook's Application Programming Interface (API). Furthermore, the user must create a Facebook 'app' and get an 'app secret'.

To get a Facebook 'app ID' and 'API secret', the excellent tutorial at http://thinktostart.com/analyzing-facebook-with-r/ provides more information.

One problem with Facebook authentication through R is that it normally requires the user to authenticate using their browser each time they wish to collect data. The `useCachedToken` argument provides a way to circumvent this, by saving and loading an authenticated 'token' file stored in the working directory. If the `useCachedToken` argument is set to `TRUE`, then the browser is not necessary for future sessions.

Value

An OAuth access token that enables R to make authenticated calls to the Facebook API.

Author(s)

Timothy Graham <timothy.graham3@uq.net.au> & Robert Ackland <robert.ackland@anu.edu.au>

See Also

`AuthenticateWithTwitterAPI` and `AuthenticateWithYouTubeAPI` for other ways to collect social media data.

Examples

```
## Not run:
  ## Use your own values for myAppID and myAppSecret
  myAppID <- "123456789098765"
  myAppSecret <- "abc123abc123abc123abc123abc123ab"

  # Authenticate with the Facebook API using `AuthenticateWithFacebookAPI`
  fb_oauth <- AuthenticateWithFacebookAPI(appID=myAppID, appSecret=myAppSecret,extended_permissions=FALSE, u

## End(Not run)
```

`AuthenticateWithInstagramAPI` *Instagram API Authentication*

Description

OAuth token based authentication with the Instagram API, with caching options for automatic authentication (i.e. avoid using the browser).

Usage

```
AuthenticateWithInstagramAPI(appID, appSecret, useCachedToken)
```

Arguments

`appID`	character string specifying the 'App ID' of the Instagram app used for authentication.
`appSecret`	character string specifying the 'API Secret' associated with the Instagram App used for authentication.
`useCachedToken`	logical. If `TRUE` then this function will look for a saved token in the current working directory (name of token file must be `fb_oauth`). If `fb_oauth` token is not found, then it will create a token and save it to current working directory (i.e. for future use).

Details

In order to collect data from Instagram, the user must first authenticate with Instagram's Application Programming Interface (API). Furthermore, the user must create a Instagram 'app' and get an 'app secret'.

To get a Instagram 'app ID' and 'API secret', please see the Instagram document at: https://instagram.com/developer/auth

One problem with Instagram authentication through R is that it normally requires the user to authenticate using their browser each time they wish to collect data. The `useCachedToken` argument provides a way to circumvent this, by saving and loading an authenticated 'token' file stored in the working directory. If the `useCachedToken` argument is set to `TRUE`, then the browser is not necessary for future sessions.

Value

An OAuth access token that enables R to make authenticated calls to the Instagram API.

Author(s)

Timothy Graham <timothy.graham3@uq.net.au> & Robert Ackland <robert.ackland@anu.edu.au>

See Also

`AuthenticateWithTwitterAPI` and `AuthenticateWithYouTubeAPI` and `AuthenticateWithFacebookAPI` for other ways to collect social media data.

Examples

```
## Not run:
  ## Use your own values for myAppID and myAppSecret
  app_id <- "123456789098765"
  app_secret <- "abc123abc123abc123abc123abc123ab"

  # Authenticate with the Instagram API using `AuthenticateWithInstagramAPI`
  instagram_oauth_token <- AuthenticateWithInstagramAPI(appID=app_id, appSecret=app_secret, useCachedToken=F

## End(Not run)
```

AuthenticateWithTwitterAPI

Twitter API Authentication

Description

Oauth based authentication with the Twitter API

Usage

```
AuthenticateWithTwitterAPI(api_key, api_secret, access_token, access_token_secret, createToken)
```

Arguments

api_key	character string specifying the 'API key' used for authentication.
api_secret	character string specifying the 'API secret' used for authentication.
access_token	character string specifying the 'access token' used for authentication.
access_token_secret	character string specifying the 'access token secret' used for authentication.
createToken	logical. !! NOT PROPERLY IMPLEMENTED YET.

Details

In order to collect data from Twitter, the user must first authenticate with Twitter's Application Programming Interface (API).

This requires setting up an App on Twitter. An excellent guide to achieving this can be found at: http://thinktostart.com/twitter-authentification-with-r/

Value

This is called for its side effect.

Author(s)

Timothy Graham <timothy.graham3@uq.net.au> & Robert Ackland <robert.ackland@anu.edu.au>

See Also

AuthenticateWithFacebookAPI and AuthenticateWithYouTubeAPI for other ways to collect social media data.

Examples

```
## Not run:
  # Firstly specify your API credentials
  my_api_key <- "1234567890qwerty"
  my_api_secret <- "1234567890qwerty"
  my_access_token <- "1234567890qwerty"
  my_access_token_secret <- "1234567890qwerty"

  AuthenticateWithTwitterAPI(api_key=my_api_key, api_secret=my_api_secret, access_token=my_access_token, acc

## End(Not run)
```

`AuthenticateWithYoutubeAPI` *YouTube API Authentication*

Description

OAuth based authentication with the Google API

Usage

```
AuthenticateWithYoutubeAPI(apiKeyYoutube)
```

Arguments

`apiKeyYoutube`	character string specifying your Google Developer API key.

Details

In order to collect data from YouTube, the user must first authenticate with Google's Application Programming Interface (API). Users can obtain a Google Developer API key at: https://console.developers.google.com

Value

This is called for its side effect.

Note

In the future this function will enable users to save the API key in working directory, and the function will automatically look for a locally stored key whenever it is called without `apiKeyYoutube` argument.

Author(s)

Timothy Graham <timothy.graham3@uq.net.au> & Robert Ackland <robert.ackland@anu.edu.au>

See Also

`AuthenticateWithFacebookAPI` and `AuthenticateWithTwitterAPI` for other ways to collect social media data.

Examples

```
## Not run:
  # Replace with your Google Developer API Key:
  my_apiKeyYoutube <- "314159265358979qwerty"

  apiKeyYoutube <- AuthenticateWithYoutubeAPI(my_apiKeyYoutube)

## End(Not run)
```

`CollectDataFacebook` *Collect data from Facebook pages for generating different types of networks*

Description

This function collects data from Facebook pages (i.e. post data and comments/likes data within posts), and structures the data into a data frame of class `dataSource.facebook`, ready for creating networks for further analysis.

Usage

```
CollectDataFacebook(pageName, rangeFrom, rangeTo, verbose, n, writeToFile, dynamic)
```

Arguments

`pageName`	character string, specifying the name of the Facebook page. For example, if page is: https://www.facebook.com/StarWars, then `pageName="StarWars"`.
`rangeFrom`	character string, specifying a 'start date' for data collection, in the format YYYY-MM-DD. For example, to collect data starting from July 4th 2015, `rangeFrom` would be "2015-07-04". Default value is current system date minus one week (i.e. the date 7 days ago).
`rangeTo`	character string, specifying an 'end date' for data collection, in the format YYYY-MM-DD. For example, to collect data until December 25th 2015, `rangeFrom` would be "2015-12-25". Default value is the current system date.
`verbose`	logical. If `TRUE` then this function will output runtime information to the console as it computes. Useful diagnostic tool for long computations. Default is `FALSE`.
`n`	numeric, maximum number of comments and likes to return (see `getPost` in Rfacebook package). Default value is 1000.
`writeToFile`	logical. If `TRUE` then the data is saved to file in current working directory (CSV format), with filename denoting `rangeFrom`, `rangeTo`, and `pageName`.
`dynamic`	logical. If `TRUE` then temporal data will be collected, which can be used to create dynamic networks (using function `CreateDynamicNetwork`). Note: Facebook API does not currently provide timestamp data for 'likes' on posts. Therefore, if dynamic is set to `TRUE`, then only 'comments' data are collected. In this way, an edge from user i to post j represents whether (and how many times) user i has commented on post j.

Details

CollectDataFacebook collects public 'post' data from a given Facebook page, including comments and 'likes' from within each post.

The function then finds and maps the relationships between users and posts, and structures these relationships into a format suitable for creating bimodal networks using CreateBimodalNetwork.

A date range must be specified for collecting post data using rangeFrom and rangeTo (i.e. data will be collected from posts posted within the date range). If no date range is supplied, then the default is the current system date minus one week (i.e. 7 days leading up to current system date).

Value

A data frame object of class dataSource.facebook that can be used with CreateBimodalNetwork.

Note

Currently supported network types:

- bimodal networks; CreateBimodalNetwork - dynamic bimodal networks; CreateDynamicNetwork

Note: dynamic networks created using Facebook data are bimodal. This means that there are two types of vertices present in the network (i.e. Facebook users and Facebook posts), with edges representing the time(s) when user i commented on post j. Currently, timestamp data is not available through the Facebook API for 'likes' data (i.e. when user i 'likes' post j), so relationships based on 'likes' are excluded from dynamic Facebook data (and therefore networks generated using CreateDynamicNetwork).

Author(s)

Timothy Graham <timothy.graham3@uq.net.au> & Robert Ackland <robert.ackland@anu.edu.au>

See Also

AuthenticateWithFacebookAPI must be run first or no data will be collected, collectTemporalDataFacebook is used to collect temporal data suitable for dynamic networks (i.e. Facebook network that changes as a function of time).

Examples

```
## Not run:
  ## Use your own values for myAppID and myAppSecret
  myAppID <- "123456789098765"
  myAppSecret <- "abc123abc123abc123abc123abc123ab"

  # Authenticate with the Facebook API using `AuthenticateWithFacebookAPI`
  fb_oauth <- AuthenticateWithFacebookAPI(appID=myAppID, appSecret=myAppSecret,extended_permissions=FALSE, u

  # Run the `CollectDataFacebook` function and store the results in variable `myFacebookData`
  myFacebookData <- CollectDataFacebook(pageName="StarWars", rangeFrom="2015-05-01",rangeTo="2015-06-03",wri

  # Create a 'bimodal' network using \code{CreateBimodalNetwork}
  g_bimodal_facebook <- CreateBimodalNetwork(myFacebookData)

  # View descriptive information about the bimodal network
  g_bimodal_facebook

## End(Not run)
```

`CollectDataInstagram` *Collect data from Instagram for generating different types of networks*

Description

This function collects data from Instagram using either hashtag (e.g. #obama) or search coordinates (latitude and longitude), and structures the data into a data frame of class `dataSource.Instagram`, ready for creating networks for further analysis. This function draws heavily on the 'searchInstagram' function (from the 'instaR' package) and includes all the arguments for collecting Instagram data using that function, as well as additional arguments listed below.

Usage

```
CollectDataInstagram(tag, n, lat, lng, distance, folder, mindate, maxdate, verbose,
  sleep, writeToFile, waitForRateLimit)
```

Arguments

`tag`	character string. Hashtag used to filter media. It is only possible for a single hashtag.
`n`	numeric. Maximum number of media to return.
`lat`	numeric. Latitude of the center search coordinate.
`lng`	numeric. Longitude of the center search coordinate.
`distance`	numeric. Default is 1km (distance=1000), max distance is 5km.
`folder`	character string. If different than `NULL`, will download all pictures to this folder.
`mindate`	character string. Minimum date for search period.
`maxdate`	character string. Maximum date for search period.
`verbose`	logical. If `TRUE` then this function will output runtime information to the console as it computes. Useful diagnostic tool for long computations. Default is `FALSE`.
`sleep`	numeric, Number of seconds between API calls (default is 0).
`writeToFile`	logical. If `TRUE` then the data is saved to file in current working directory (CSV format), with filename denoting the current time/date.
`waitForRateLimit`	logical. If `TRUE` then it will try to observe the API rate limit by ensuring that no more than 5000 API calls are made per hour (the current rate limit). If more than 5000 calls are made within a 60 minute window, then all operates will suspend for 60 minutes, and resume afterwards.

Details

`CollectDataInstagram` collects public pictures and videos from Instagram, and also collects the maximum amount of comments and 'likes' for each post. It draws on and extends the 'searchInstagram' function from the 'instaR' package.

As the 'instaR' documentation describes, it is only possible to apply one filter at a time: either search by hashtag OR search by coordinates. The 'mindate' and 'maxdata' search parameters only work when searching by location, not when searching by tag.

After the data are collected, the function finds and maps the relationships between users and posts, and structures these relationships into a format suitable for creating bimodal networks using `CreateBimodalNetwork`.

Value

A data frame object of class `dataSource.Instagram` that can be used with `CreateBimodalNetwork`.

Note

The current implementation only supports creating bimodal networks. Other network types will be added in the near future.

- bimodal networks; `CreateBimodalNetwork`

A bimodal (or two-mode) network means that there are two types of vertices present in the network (i.e. Instagram users and Instagram posts), with edges representing user i 'commenting' or 'liking' post j.

Author(s)

Timothy Graham <timothy.graham3@uq.net.au> & Robert Ackland <robert.ackland@anu.edu.au>

See Also

`AuthenticateWithInstagramAPI` must be run first or no data can be collected through the API.

Examples

```
## Not run:
## Use your own values for myAppID and myAppSecret
myAppID <- "123456789098765"
myAppSecret <- "abc123abc123abc123abc123abc123ab"

# Authenticate with the Instagram API using `AuthenticateWithInstagramAPI`
instagram_oauth_token <- AuthenticateWithInstagramAPI(appID=app_id, appSecret=app_secret,
  useCachedToken=TRUE)

# EXAMPLE 1

# Run the `CollectDataInstagram` function and store the results in variable `myInstagramData`
# (searching by hashtag)
myInstagramData <- CollectDataInstagram(tag="obama", distance=5000, n=100, folder=NULL,
verbose=TRUE, waitForRateLimit=TRUE)

# Create a 'bimodal' network using \code{CreateBimodalNetwork}
g_bimodal_instagram_obama <- CreateBimodalNetwork(myInstagramData,writeToFile=F)

# View descriptive information about the bimodal network
g_bimodal_instagram_obama

# EXAMPLE 2

# Run the `CollectDataInstagram` function and store the results in variable `myInstagramData`
# (searching by coordinates in Brisbane (Australia) with a radius of 5km)
myInstagramData <- CollectDataInstagram(lat=-27.4701, lng=153.0220, distance=5000, n=100,
  folder=NULL, verbose=TRUE, waitForRateLimit=TRUE)

# Create a 'bimodal' network using \code{CreateBimodalNetwork}
g_bimodal_instagram_brisbane <- CreateBimodalNetwork(myInstagramData,writeToFile=F)

# View descriptive information about the bimodal network
```

```
g_bimodal_instagram_brisbane

## End(Not run)
```

`CollectDataTwitter` *Collect data from Twitter for generating different types of networks*

Description

This function collects data from Twitter based on hashtags or search terms, and structures the data into a data frame of class `dataSource.twitter`, ready for creating networks for further analysis.

Usage

```
CollectDataTwitter(searchTerm, numTweets, verbose, writeToFile, language)
```

Arguments

`searchTerm`	character string, specifying a search term or phrase (e.g. "Australian politics") or hashtag (e.g. "#auspol"). Many query operators are available - see the Twitter documentation for more information: https://dev.twitter.com/rest/public/search
`numTweets`	numeric integer, specifying how many tweets to be collected. Defaults to 1500. Maximum tweets for a single call of this function is 1500.
`verbose`	logical. If `TRUE` then this function will output runtime information to the console as it computes. Useful diagnostic tool for long computations. Default is `FALSE`.
`writeToFile`	logical. If `TRUE` then the data is saved to file in current working directory (CSV format), with filename denoting current system time and `searchTerm`. Default is `FALSE`.
`language`	character string, restricting tweets to the given language, given by an ISO 639-1 code. For example, "en" restricts to English tweets. Defaults to NULL.

Details

`CollectDataTwitter` collects public 'tweets' from Twitter using the Twitter API.

The function then finds and maps the relationships of entities of interest in the data (e.g. users, terms, hashtags), and structures these relationships into a data frame format suitable for creating unimodal networks (`CreateActorNetwork`), bimodal networks (`CreateBimodalNetwork`), and semantic networks (`CreateSemanticNetwork`).

The maximum number of tweets for a single call of `CollectDataTwitter` is 1500.

Language support is available, using the `language` argument. The user can restrict tweets returned to a particular language, using the ISO 639-1 code. For example, restricting to English would use `language="en"`. The full list of codes is available here: https://en.wikipedia.org/wiki/List_of_ISO_639-1_codes.

A variety of query operators are available through the Twitter API. For example, "love OR hate" returns any tweets containing either term (or both). For more information see the Twitter API documentation (under the heading 'Query Operators'): https://dev.twitter.com/rest/public/search

Value

A data frame object of class `dataSource.twitter` that can be used for creating unimodal networks (`CreateActorNetwork`), bimodal networks (`CreateBimodalNetwork`), and semantic networks (`CreateSemanticNetwork`).

Note

Data generated using this function is *not* suitable for dynamic networks. Dynamic Twitter networks are not currently implemented in the SocialMediaLab package. This will be implemented in a future release.

Author(s)

Timothy Graham <timothy.graham3@uq.net.au> & Robert Ackland <robert.ackland@anu.edu.au>

See Also

`AuthenticateWithTwitterAPI` must be run first or no data will be collected.

Examples

```
## Not run:
  # Firstly specify your API credentials
  my_api_key <- "1234567890qwerty"
  my_api_secret <- "1234567890qwerty"
  my_access_token <- "1234567890qwerty"
  my_access_token_secret <- "1234567890qwerty"

  # Authenticate with the Twitter API using \code{AuthenticateWithTwitterAPI}
  AuthenticateWithTwitterAPI(api_key=my_api_key, api_secret=my_api_secret, access_token=my_access_token, acc

  # Collect tweets data using \code{myTwitterData}
  myTwitterData <- CollectDataTwitter(searchTerm="#auspol",
    numTweets=150,writeToFile=FALSE,verbose=FALSE)

  # Create an 'actor' network using \code{CreateActorNetwork}
  g_actor_twitter <- CreateActorNetwork(myTwitterData)

  # Create a 'bimodal' network using \code{CreateBimodalNetwork}
  g_bimodal_twitter <- CreateBimodalNetwork(myTwitterData)

  # Create a 'semantic' network using \code{CreateSemanticNetwork}
  g_semantic_twitter <- CreateSemanticNetwork(myTwitterData)

## End(Not run)
```

`CollectDataYoutube` *Collect YouTube comments data for generating different types of networks*

Description

This function collects YouTube comments data for one or more YouTube videos. It structures the data into a data frame of class `dataSource.youtube`, ready for creating networks for further analysis.

Usage

```
CollectDataYoutube(videoIDs, apiKeyYoutube, verbose, writeToFile,maxComments)
```

Arguments

`videoIDs`	character vector, specifying one or more YouTube video IDs. For example, if the video URL is 'https://www.youtube.com/watch?v=W2GZFeYGU3s', then use `videoIDs='W2GZFeYGU3s'`. For multiple videos, the function `GetYoutubeVideoIDs` can be used to create a vector object suitable as input for `videoIDs`.
`apiKeyYoutube`	character string, specifying the Google Developer API Key used for authentication.
`verbose`	logical. If `TRUE` then this function will output runtime information to the console as it computes. Useful diagnostic tool for long computations. Default is `FALSE`.
`writeToFile`	logical. If `TRUE` then the data is saved to file in current working directory (CSV format), with filename denoting current system time. Default is `FALSE`.
`maxComments`	numeric integer, specifying how many 'top-level' comments to collect from each video. This value *does not* take into account 'reply' comments (i.e. replies to top-level comments), therefore the total number of comments collected may be higher than `maxComments`. By default this function attempts to collect all comments.

Details

`CollectDataYoutube` collects public comments from YouTube videos, using the YouTube API.

The function then finds and maps the relationships of YouTube users who have interacted with each other (i.e. user i has replied to user j or mentioned user j in a comment) and structures these relationships into a data frame format suitable for creating unimodal networks (`CreateActorNetwork`).

For multiple videos, the user may wish to use the function `GetYoutubeVideoIDs`, which creates a character vector of video IDs from a plain text file of YouTube video URLs, which can then be used for the `videoIDs` argument of the function `CollectDataYoutube`.

Value

A data frame object of class `dataSource.youtube` that can be used for creating unimodal networks (`CreateActorNetwork`).

Note

Currently supported network types:

- unimodal 'actor' network; `CreateActorNetwork`

Data generated using this function is *not* suitable for dynamic networks. Dynamic YouTube comments networks are not currently implemented in the SocialMediaLab package. This will be implemented in a future release.

Note on maxComments argument: Due to quirks/specifications of the Google API, it is currently not possible to specify the exact number of comments to return from the API using maxResults argument (i.e. including comments that are replies to top-level comments). Therefore, the number of comments collected is usually somewhat greater than maxComments, if a value is specified for this argument. For example, if a video contains 10 top-level comments, and one of these top-level comments has 5 'child' or reply comments, then the total number of comments collected will be equal to 15. Currently, the user must 'guesstimate' the maxResults value, to collect a number of comments in the order of what they require.

Author(s)

Timothy Graham <timothy.graham3@uq.net.au> & Robert Ackland <robert.ackland@anu.edu.au>

See Also

AuthenticateWithYoutubeAPI must be run first or no data will be collected.

Examples

```
## Not run:
  # Use your own Google Developer API Key here:
  myApiKey <- "1234567890"

  # Authenticate with the Google API
  apiKeyYoutube <- AuthenticateWithYoutubeAPI(apiKeyYoutube=myApiKey)

  # Generate a vector of YouTube video IDs to collect data from
  # (or use the function `GetYoutubeVideoIDs` to automatically generate from a plain text file of video URLs)
  videoIDs <- c("W2GZFeYGU3s","mL27TAJGlWc")

  # Collect the data using function `CollectDataYoutube`
  myYoutubeData <- CollectDataYoutube(videoIDs,apiKeyYoutube,writeToFile=FALSE)

  # Create an 'actor' network using the function `CreateActorNetwork`
  g_actor_youtube <- CreateActorNetwork(myYoutubeData)

## End(Not run)
```

CreateActorNetwork *Create 'actor' networks from social media data*

Description

This function creates a unimodal 'actor' network from social media data (i.e. from data frames of class dataSource, or for Twitter data it is also possible to provide a *list* of data frames). In this actor network, edges represent relationships between actors of the same type (e.g. interactions between Twitter users). For example, with Twitter data an interaction is defined as a 'mention' or 'reply' or 'retweet' from user i to user j, given 'tweet' m. With YouTube comments, an interaction is defined as a 'reply' or 'mention' from user i to user j, given 'comment' m.

Usage

```
CreateActorNetwork(x, writeToFile)
```

Arguments

x	a data frame of class dataSource. For Twitter data, it is also possible to provide a *list* of data frames (i.e. data frames that inherit class dataSource and twitter). Only lists of Twitter data frames are supported at this time. If a list of data frames is provided, then the function binds these row-wise and computes over the entire data set.
writeToFile	logical. If TRUE then the network is saved to file in current working directory (GRAPHML format), with filename denoting the current date/time and the type of network.

Details

This function creates a (weighted and directed) unimodal 'actor' network from a data frame of class dataSource (which are created using the 'CollectData' family of functions in the SocialMediaLab package), or a *list* of Twitter data frames collected using CollectDataTwitter function.

The resulting network is an igraph graph object. This graph object is unimodal because edges represent relationships between vertices of the same type (read: 'actors'), such as replies/retweets/mentions between Twitter users. Edges are directed and weighted (e.g. if user i has replied n times to user j, then the weight of this directed edge equals n).

Value

An igraph graph object, with directed and weighted edges.

Note

Not all data sources in SocialMediaLab can be used for creating actor networks.

Currently supported data sources are:

- YouTube - Twitter

Other data sources (e.g. Facebook) will be implemented in the future. The user is notified if they try to create actor networks for incompatible data sources.

For Twitter data, actor networks can be created from multiple data frames (i.e. datasets collected individually using CollectDataTwitter). Simply create a list of the data frames that you wish to create a network from. For example, `myList <- list(myTwitterData1, myTwitterData2, myTwitterData3)`.

Author(s)

Timothy Graham <timothy.graham3@uq.net.au> & Robert Ackland <robert.ackland@anu.edu.au>

See Also

See CollectDataYoutube and CollectDataTwitter to collect data sources for creating actor networks in SocialMediaLab.

Examples

```
## Not run:
  ## This example shows how to collect YouTube comments data and create an actor network

  # Use your own Google Developer API Key here:
  myApiKey <- "1234567890"
```

```
  # Authenticate with the Google API
  apiKeyYoutube <- AuthenticateWithYoutubeAPI(apiKeyYoutube=myApiKey)

  # Generate a vector of YouTube video IDs to collect data from
  # (or use the function `GetYoutubeVideoIDs` to automatically generate from a plain text file of video URLs)
  videoIDs <- c("W2GZFeYGU3s","mL27TAJGlWc")

  # Collect the data using function `CollectDataYoutube`
  myYoutubeData <- CollectDataYoutube(videoIDs,apiKeyYoutube,writeToFile=FALSE)

  # Create an 'actor' network using the function `CreateActorNetwork`
  g_actor_youtube <- CreateActorNetwork(myYoutubeData)

  # Description of actor network
  g_actor_youtube

## End(Not run)
```

`CreateBimodalNetwork` *Create bimodal networks from social media data*

Description

This function creates a bimodal network from social media data (i.e. from data frames of class `dataSource`, or for Twitter data it is also possible to provide a *list* of data frames), with edges representing relationships between actors of two different types (e.g. Facebook users and Facebook posts, with edges representing whether a user has commented or 'liked' a post).

Usage

```
CreateBimodalNetwork(x, writeToFile, removeTermsOrHashtags)
```

Arguments

`x`	a data frame of class `dataSource`. For Twitter data, it is also possible to provide a *list* of data frames (i.e. data frames that inherit class `dataSource` and `twitter`). Only lists of Twitter data frames are supported at this time. If a list of data frames is provided, then the function binds these row-wise and computes over the entire data set.
`writeToFile`	logical. If `TRUE` then the network is saved to file in current working directory (GRAPHML format), with filename denoting the current date/time and the type of network.
`removeTermsOrHashtags`	character vector. Default is none. Otherwise this argument specifies which terms or hashtags (i.e. vertices with matching 'name') should be removed from the bimodal network. This is useful to remove the search term or hashtag that was used to collect the data (i.e. remove the corresponding vertex in the graph). For example, a value of "#auspol" means that if there is a vertex with the exact name "#auspol" then this vertex will be removed.

Details

This function creates a (directed and weighted) bimodal network from a data frame of class `dataSource` (which are created using the 'CollectData' family of functions in the SocialMediaLab package), or a *list* of Twitter data frames collected using `CollectDataTwitter` function.

The resulting network is an igraph graph object. This graph object is bimodal because edges represent relationships between vertices of two different types. For example, in a bimodal Facebook network, vertices represent Facebook users or Facebook posts, and edges represent whether a user has commented or 'liked' a post. Edges are directed and weighted (e.g. if user i has commented n times on post j, then the weight of this directed edge equals n).

Value

An igraph graph object, with weighted and directed edges.

Note

Not all data sources in SocialMediaLab can be used for creating bimodal networks.

Currently supported data sources are:

- Facebook - Twitter

Other data sources (e.g. YouTube) will be implemented in the future. Additionally, the user is notified if they try to create bimodal networks for incompatible data sources.

For Twitter data, bimodal networks can be created from multiple data frames (i.e. datasets collected individually using CollectDataTwitter). Simply create a list of the data frames that you wish to create a network from. For example, `myList <- list(myTwitterData1, myTwitterData2, myTwitterData3)`.

Author(s)

Timothy Graham <timothy.graham3@uq.net.au> & Robert Ackland <robert.ackland@anu.edu.au>

See Also

See `CollectDataFacebook` and `CollectDataTwitter` to collect data for creating bimodal networks in SocialMediaLab.

Examples

```
## Not run:
  ## This example shows how to collect Facebook page data and create a bimodal network

  # Use your own values for myAppID and myAppSecret
  myAppID <- "123456789098765"
  myAppSecret <- "abc123abc123abc123abc123abc123ab"

  # Authenticate with the Facebook API using `AuthenticateWithFacebookAPI`
  fb_oauth <- AuthenticateWithFacebookAPI(appID=myAppID, appSecret=myAppSecret,extended_permissions=FALSE, u

 # Run the `CollectDataFacebook` function and store the results in variable `myFacebookData`
 myFacebookData <- CollectDataFacebook(pageName="StarWars", rangeFrom="2014-05-15",rangeTo="2014-06-03",wri

  # Create a 'bimodal' network using \code{CreateBimodalNetwork}
  g_bimodal_facebook <- CreateBimodalNetwork(myFacebookData)

  # View descriptive information about the bimodal network
```

```
  g_bimodal_facebook

## End(Not run)
```

`CreateDynamicNetwork` *Create dynamic networks from social media data (networks that vary over time)*

Description

This function creates a dynamic network from social media data (i.e. from data frames of class `dataSource` and class `temporal`).

Usage

```
CreateDynamicNetwork(x, writeToFile)
```

Arguments

`x`	a data frame of class `dataSource` and class `temporal`.
`writeToFile`	logical. If `TRUE` then the network is saved to file in current working directory (GRAPHML format), with filename denoting the current date/time and the type of network.

Details

This function creates a directed network from a data frame of class `dataSource` and `temporal` (which are created using the 'CollectTemporalData' family of functions in the SocialMediaLab package).

The resulting dynamic network is an igraph graph object. This graph object is dynamic because all edges in the network have a timestamp attribute expressing their existence as a function of time. Edges are directed and non-weighted (i.e. for each 'interaction' between two vertices there exists an edge with a timestamp attribute). For example, a Facebook user i may 'comment on' post j at time T1 and T2, which is represented by two edges with timestamp attribute T1 and T2, respectively.

Value

An igraph graph object, with directed and non-weighted edges.

Note

Not all data sources in SocialMediaLab can be used for creating dynamic networks.

Currently supported data sources are:

- Facebook

There are three types of edge attributes for timestamp data. The `timestamp` edge attribute is the timestamp in human readable format. For example, "2015-05-29 19:54:39", in the format YYYY-MM-DD HH:MM:SS. The `timestampNumeric` edge attribute utilises an 'epoch' starting from the timestamp of the oldest comment in the data set, which is given the value 0 (zero). Therefore, the oldest comment in the dataset will become 0 (zero), and all comments thereafter will be assigned a number that represents the number of seconds each comment occurred *after* the 'epoch'

comment (i.e. from the zero starting time). The `timestampUnixEpoch` edge attribute utilises the standard Unix epoch, indicating the number of seconds since January 1, 1970 (i.e. how many seconds after the Unix epoch each commented was posted). This provides the ability to compare different dynamic datasets, perform unions on dynamic networks, etc.

Dynamic networks created using Facebook data are bimodal. This means that there are two types of vertices present in the network (i.e. Facebook users and Facebook posts), with edges representing the time when user i commented on post j. Currently timestamp data is not available through the Facebook API for 'likes' data (i.e. when user i 'likes' post j), so relationships based on 'likes' are excluded from the dynamic network.

Other data sources (e.g. YouTube and Twitter) will be implemented in the future. Additionally, the user is notified if they try to create dynamic networks for incompatible data sources.

Author(s)

Timothy Graham <timothy.graham3@uq.net.au> & Robert Ackland <robert.ackland@anu.edu.au>

References

Kolaczyk, E. D., Csardi, G. (2014). Statistical analysis of network data with R. New York: Springer, Chapter 10.

See Also

See `CollectDataFacebook` to collect data for creating dynamic networks in SocialMediaLab.

Examples

```
## Not run:
  ## This example shows how to collect Facebook page data and create a bimodal network

  # Use your own values for myAppID and myAppSecret
  myAppID <- "123456789098765"
  myAppSecret <- "abc123abc123abc123abc123abc123ab"

  # Authenticate with the Facebook API using `AuthenticateWithFacebookAPI`
  fb_oauth <- AuthenticateWithFacebookAPI(appID=myAppID, appSecret=myAppSecret,extended_permissions=FALSE, u

  # Run the `CollectTemporalDataFacebook` function and store the results in variable `myTemporalFacebookData`
  myTemporalFacebookData <- CollectTemporalDataFacebook(pageName="StarWars",
    rangeFrom="2015-05-14",rangeTo="2015-06-04",verbose=FALSE,writeToFile=FALSE)

  # Create a dynamic 'bimodal' Facebook network using `CreateDynamicNetwork`
  g_bimodal_dynamic_facebook <- CreateDynamicNetwork(myTemporalFacebookData)

  # View descriptive information about the bimodal network
  g_bimodal_facebook

## End(Not run)
```

`CreateSemanticNetwork` *Create semantic networks from social media data (semantic relationships between concepts)*

Description

This function creates a semantic network from social media data (i.e. from data frames of class `dataSource`, or for Twitter data it is also possible to provide a *list* of data frames). In such semantic networks, concepts are words/terms extracted from the text corpus of social media data (e.g. tweets on Twitter).

Usage

```
CreateSemanticNetwork(x, writeToFile, termFreq, hashtagFreq, removeTermsOrHashtags, stopwordsEngl
```

Arguments

`x`	a data frame of class `dataSource`. For Twitter data, it is also possible to provide a *list* of data frames (i.e. data frames that inherit class `dataSource` and `twitter`). Only lists of Twitter data frames are supported at this time. If a list of data frames is provided, then the function binds these row-wise and computes over the entire data set.
`writeToFile`	logical. If `TRUE` then the network is saved to file in current working directory (GRAPHML format), with filename denoting the current date/time and the type of network.
`termFreq`	numeric integer, specifying the percentage of most frequent TERMS to include. For example, a value of 20 means that the 20
`hashtagFreq`	** NOT IMPLEMENTED YET - DEFAULTS TO ALL HASHTAGS **. numeric integer, specifying the percentage of most frequent HASHTAGS to include. For example, a value of 80 means that the 80
`removeTermsOrHashtags`	character vector. Default is none. Otherwise this argument specifies which terms or hashtags (i.e. vertices with matching 'name') should be removed from the semantic network. This is useful to remove the search term or hashtag that was used to collect the data (i.e. remove the corresponding vertex in the graph). For example, a value of "#auspol" means that if there is a vertex with the name "#auspol" then this vertex will be removed.
`stopwordsEnglish`	logical. If `TRUE` then English stopwords are removed from the tweets (e.g. words such as 'the' or 'and'). Using `FALSE` may be helpful non-English data sets. The default is `TRUE` (i.e. stopwords will be removed).

Details

This function creates a weighted network from a data frame of class `dataSource` (which are created using the 'CollectData' family of functions in the SocialMediaLab package), or a *list* of Twitter data frames collected using `CollectDataTwitter` function.

The resulting semantic network is an igraph graph object. This graph object is semantic because vertices represent unique concepts (in this case unique terms/words extracted from a social media

text corpus), and edges represent the co-occurrence of terms for all observations in the data set. For example, for a Twitter semantic network, vertices represent either hashtags (e.g. "#auspol") or single terms ("politics"). If there are 1500 tweets in the data set (i.e. 1500 observations), and the term "#auspol" and the term "politics" appear together in every tweet, then this will be represented by an edge with weight equal to 1500.

Value

An igraph graph object, with weighted edges.

Note

Not all data sources in SocialMediaLab can be used for creating semantic networks.

Currently supported data sources are:

- Twitter

Other data sources (e.g. YouTube and Facebook) will be implemented in the future. Additionally, the user is notified if they try to create semantic networks for incompatible data sources.

For Twitter data, semantic networks can be created from multiple data frames (i.e. datasets collected individually using CollectDataTwitter). Simply create a list of the data frames that you wish to create a network from. For example, `myList <- list(myTwitterData1, myTwitterData2, myTwitterData3)`.

Author(s)

Timothy Graham <timothy.graham3@uq.net.au> & Robert Ackland <robert.ackland@anu.edu.au>

See Also

See `CollectDataTwitter` to collect data for creating semantic networks in SocialMediaLab.

Examples

```
## Not run:
  ## This example shows how to collect Twitter data and create a semantic network

  # Firstly specify your API credentials
  my_api_key <- "1234567890qwerty"
  my_api_secret <- "1234567890qwerty"
  my_access_token <- "1234567890qwerty"
  my_access_token_secret <- "1234567890qwerty"

  # Authenticate with the Twitter API using \code{AuthenticateWithTwitterAPI}
  AuthenticateWithTwitterAPI(api_key=my_api_key, api_secret=my_api_secret, access_token=my_access_token, acc

  # Collect tweets data using \code{myTwitterData}
  myTwitterData <- CollectDataTwitter(searchTerm="#auspol",
    numTweets=200,writeToFile=FALSE,verbose=FALSE)

  # Create a 'semantic' network using \code{CreateSemanticNetwork}
  g_semantic_twitter <- CreateSemanticNetwork(myTwitterData,writeToFile=FALSE,termFreq=20,hashtagFreq=80)

## End(Not run)
```

GetYoutubeVideoIDs *Extract/scrape the IDs from a set of YouTube video URLs*

Description

This function reads a list of YouTube video URLs from a text file and converts them to a vector object. For example, "https://www.youtube.com/watch?v=73I5dRucCds" has the ID "73I5dRucCds". This function can be used to create an object for the argument `videoIDs` in the function `CollectDataYoutube`, that is, by extracting the IDs for a set of YouTube videos and compiling them into a vector, ready for collecting data with `CollectDataYoutube`.

Usage

```
GetYoutubeVideoIDs(file)
```

Arguments

`file`	The connection to read from. This can be a local file, or a http or ftp connection. It can also be a character string with the file name or URI. The file must be plain text format with the URL of each YouTube video specified on a new line (separated by character return). For example, the first line might contain https://www.youtube.com/watch?v=73I5dRucCds, and the second line might contain https://www.youtube.com/watch?v=6S9r_YbqHy8.

Value

a character vector representing a set of YouTube video IDs, each with number of characters equal to 11 (e.g. "73I5dRucCds").

Note

This function is useful for lots of videos. However, many videos may take a *long* time to collect data from. In such cases it is recommended to use the `verbose=TRUE` argument for the function `CollectDataYoutube`, in order to keep track of progress during computation.

Author(s)

Timothy Graham <timothy.graham3@uq.net.au> & Robert Ackland <robert.ackland@anu.edu.au>

See Also

Use `CollectDataYoutube` for collecting YouTube comments data.

Examples

```
## Not run:
  ## This example shows how to use `GetYoutubeVideoIDs` to extract video IDs from YouTube video URLs, and then

  # Use your own Google Developer API Key here:
  myApiKey <- "1234567890"

  # Authenticate with the Google API
  apiKeyYoutube <- AuthenticateWithYoutubeAPI(apiKeyYoutube=myApiKey)
```

```
# Use the function `GetYoutubeVideoIDs` to automatically generate vector of IDs from a plain text file of vi
videoIDs <- GetYoutubeVideoIDs(file="youtube_to_scrape.txt")

# Collect the data using function `CollectDataYoutube`
myYoutubeData <- CollectDataYoutube(videoIDs,apiKeyYoutube,writeToFile=FALSE)

## End(Not run)
```

Index