Aug20Ditchley

Release 0.1

Team LEAR

Sep 07, 2020

CONTENTS:

CHAPTER

ONE

CONTRIBUTORS

1.1 Luca Lamoni

1.2 Elizabeth Nicholson

1.3 Adam Hawken

1.4 Robert Webster

CHAPTER

TWO

PROJECT ENVIRONMENT SETUP

2.1 Setting up the Python environment

To create the bare-bones of the Python environment in the Anaconda prompt, run the following:

```
conda create -n lear python=3.6
```

Once the new environment has been created, switch into it using

```
conda activate lear
```

To populate the environment with the required packages, simply run the following in the prompt while in the top level of the repo directory;

```
pip install requirements.txt
```

2.2 Python dependencies

Due to current dependency conflicts with py2neo, the project requires a python version at or below 3.7.1.

The primary module dependencies of this project are:

- twint
- nest_asyncio
- tweepy
- py2neo
- pandas
- numpy
- tqdm
- nltk
- gensim
- spacy
- beautifulsoup
- sklearn
- wordcloud

- wikipedia
- Sphinx
- sphinx_rtd_theme

The full list of package dependencies (including versions) which result from this main set can be found in the *Python Module Requirements*.

2.3 Jupyter Notebooks & Plotly

Plotly is a python module which is excellent for creating interactive plots that allow you to visually inspect data in a way that would be incredibly challenging in matplotlib. To get plotly up-and-running in a Jupyter Notebook (or Jupyter Lab) requires a few extra bits and bobs.

In your conda environment, first install plotly, node.js and the node package manager with:

```
pip install plotly==4.9.0

conda install nodejs
```

Then, to get going in Jupyter Notebook, run:

```
pip install jupyterlab "ipywidgets>=7.5"
```

Or if you are using notebooks in Jupyter Lab:

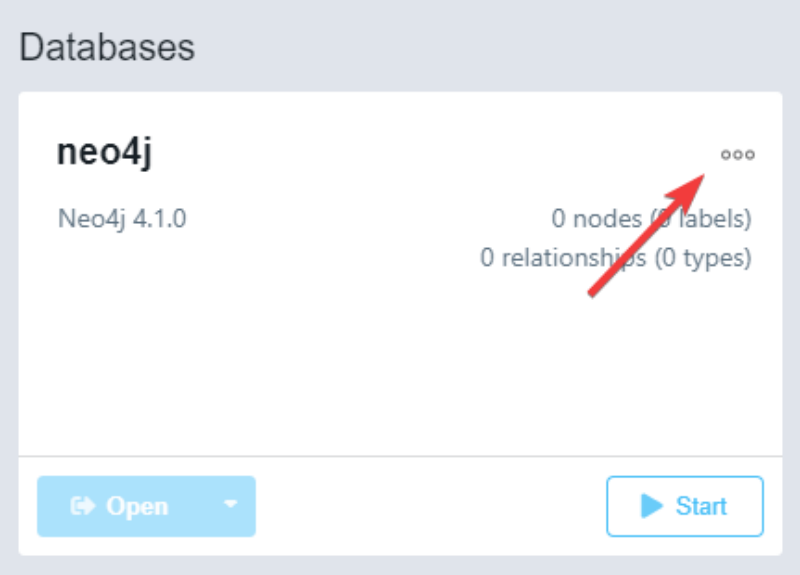
```
# JupyterLab renderer support
jupyter labextension install jupyterlab-plotly@4.9.0

# OPTIONAL: Jupyter widgets extension
jupyter labextension install @jupyter-widgets/jupyterlab-manager plotlywidget@4.9.0
```

2.4 Neo4j Graph Database Directory

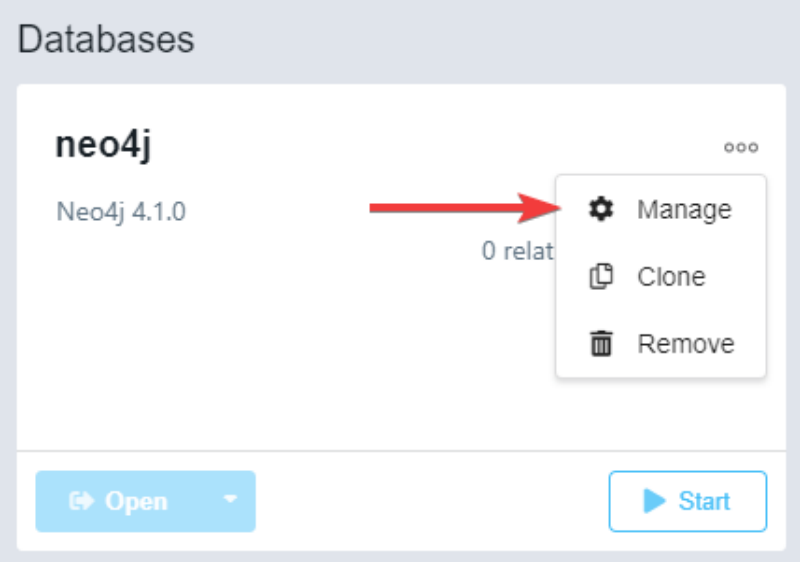
We found that Neo4j requires a few hoops to jump through to get off the ground and working.

One of the biggest sticking points is the need to have data being loaded into the database in the same folder it belongs to.

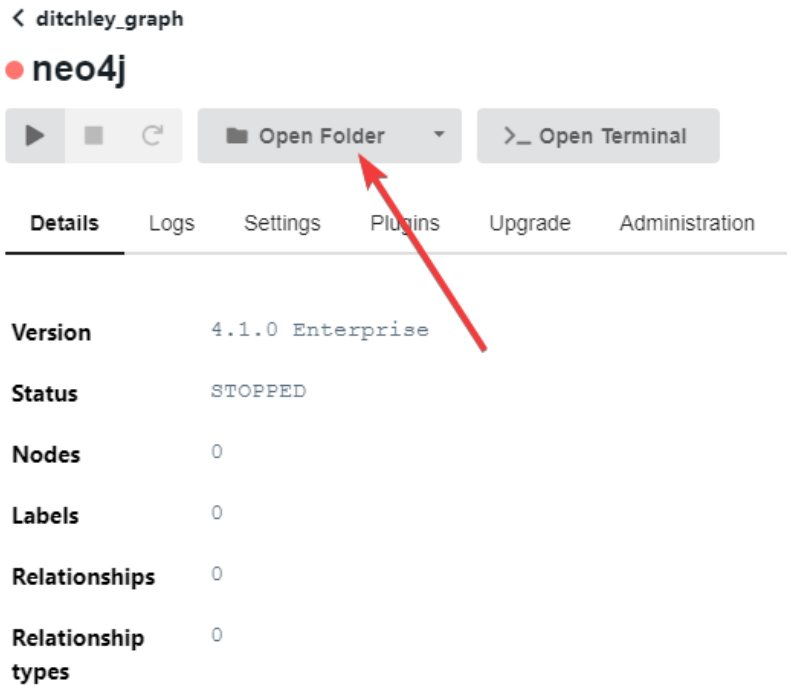
First up, we need to find the directory where the graph lives. This is straightforward to do through the Neo4j desktop environment.



Find your graph and open the `Manage` panel.



Click on `Open Folder` to go straight to the graph directory.



From the directory that opens, you want to copy the filepath, which will look something like this: `D:\neo4j\neo4jDatabases\database-47d324c5-ebe5-4afa-b595-e91969807b1b\installation-4.1.0`

Within this directory, we want to create a symbolic link inside the `imports` folder to our data directory. On Windows 10, this is slightly convoluted, so we give an example below:

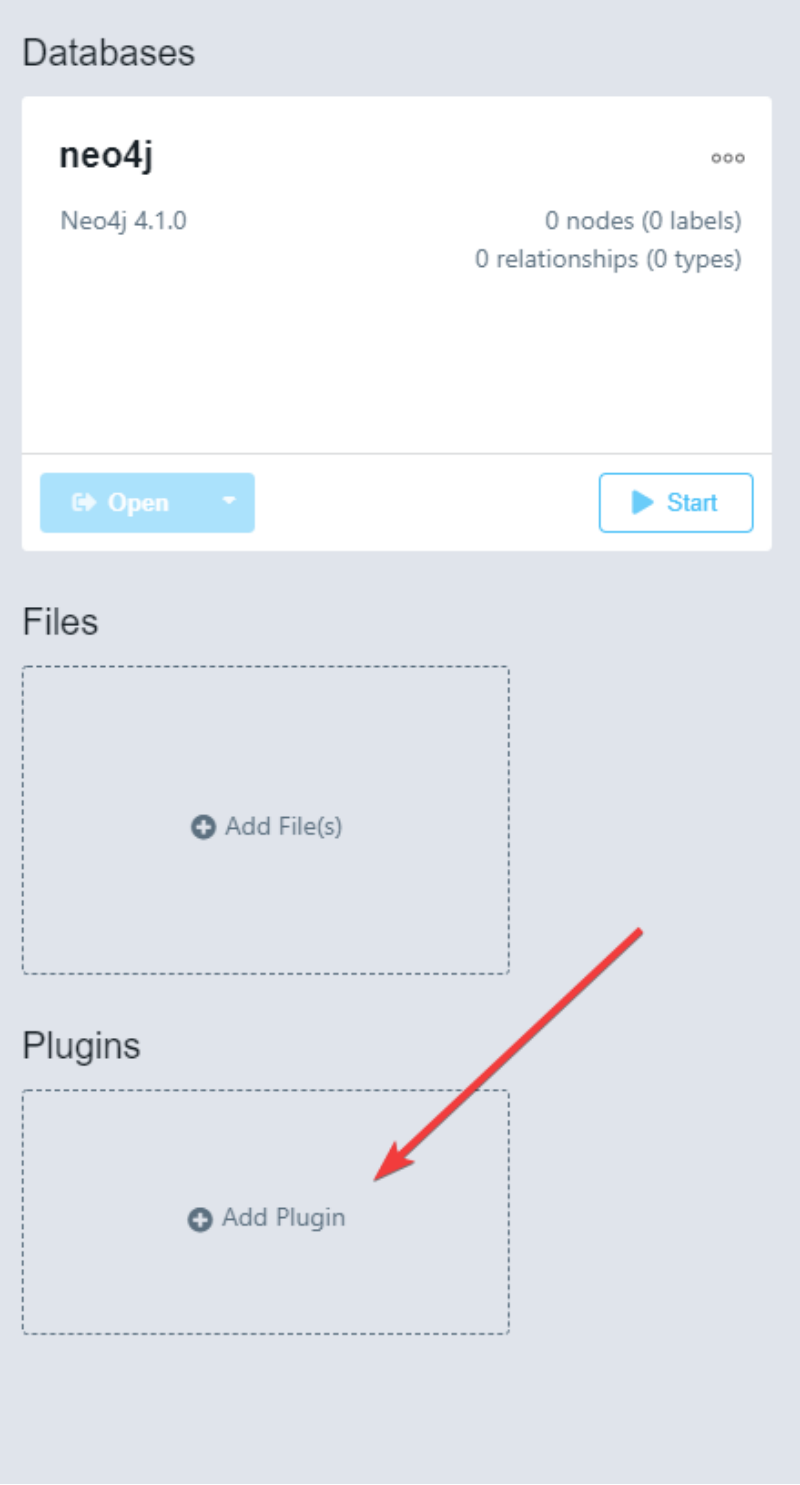
1. Open the command line prompt `cmd` *as an administrator*.
2. Run the command `mklink /D graph-dir/import/data target-data-dir` For example: `mklink /D D:\neo4j\neo4jDatabases\database-47d324c5-ebe5-4afa-b595-e91969807b1b\installation-4.1.0\import\data D:\Aug20_Ditchley\data`

If the link creation is successful, you're now good to go, start your graph and import data!

On Mac/Linux, the command for making the symbolic link would be something like: `ln -s /mnt/neo4j/neo4jDatabases/database-47d324c5-ebe5-4afa-b595-e91969807b1b/installation-4.1.0/import/data /mnt/Aug20_Ditchley/data`

2.5 Neo4j Graph Database Plugins

We also need a couple of plugins (**APOC** and **Graph Data Science Library**) installed on the graph, which can be added from the plugins field in Neo4j Desktop indicated in the image below:



CHAPTER

THREE

THE DATA PIPELINE

There are 4 key components to our analysis pipeline:

- Data collection
- Statistical analysis
- Tweet topic modelling
- Graph database interactions

The workflow to run through each of these components can be found in the `pipeline` directory of the repository. Some broader details are outlined below:

3.1 Data Collection: Scraping friend lists

The friend lists are scraped using Twint (with the _get_friends function) and saved as individual csv for each journalist; the individual csvs are then joined using the function join_friends_csv. The _get_friends function is used in combination with then twint_in_queue function which uses multi-threading on multiple cores (the number of cores need to be assigned as a function argument). This set up speeds up the process (the Twitter API has more limitations for dowloading lists of friends/followers and would therefore be slower). It has to be noted that if you want to abort the process while the twint_in_queue function is running, you will need to interrupt and restart your kernel, and delete from the folder the files that were produced because they might be incomplete lists of friends.

3.2 Data Collection: Requesting user tweets

We included in this section of the pipeline functions that can scrape tweets both with Twint and Twitter API. We did this because the two methods produce slightly different datasets and the pipeline user might be more interested one compared to the other. Main differences between Twint and Twitter API:

Twint:

- it has no limitation on the amount of tweets that can be scraped
- the tweet search can be costomized for a specific time period
- for each tweet, we get the number of replies, times it has been retweeted, and likes
- the tweet dataset does not contain retweets (tweets that have been retweeted without adding any text)
- Twint provides a conversation_id, which is a reference to the first tweet of that particular conversation.
- in general, Twint provide less information for each individual tweet

Twitter API:

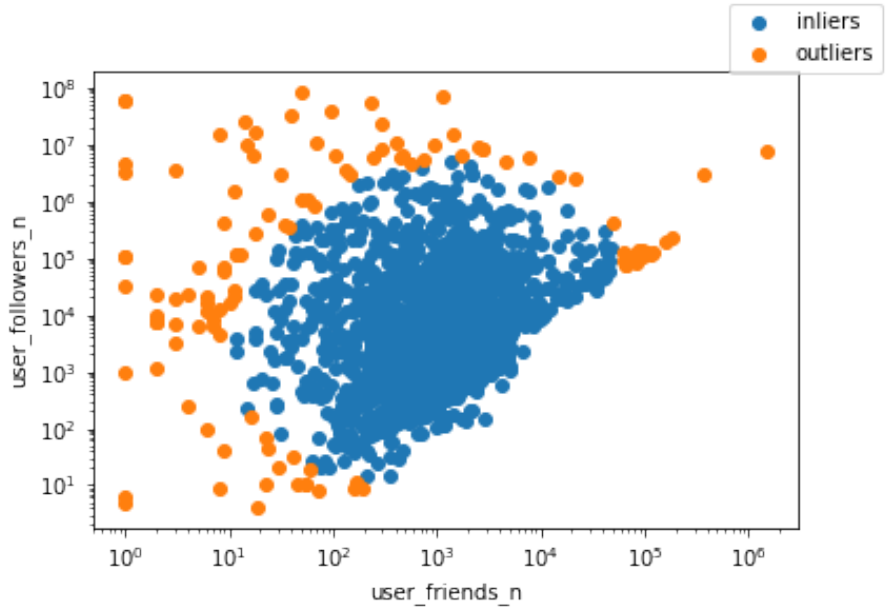
- this method has a maximum of 3200 tweets per user, and these will be the most recent ones
- the number of replies is not available for this type of dataset
- this type of dataset contains retweets and the information regarding who is the author of the tweet
- there is no reference to the conversation_id but we get a in_reply_to_status_id which represent the reference to the tweet the user responded to
- in general, the Twitter API provide a more rich and detailed set of information for each tweet
- it is also the fastest method between the two.

3.3 Statistical Analysis

Part of the process of refining the list of users we want to include involves examining the distribution of user attributes.

Here, two metrics can be drawn from user profiles which can be used to weed out some celebrities and other accounts that are not relevant:

- the number of *friends* a user has and
- the number of *followers*.



Considered as a whole, the distribution of user profiles by friends an followers follows a log-normal distribution, and users at the extrema of this distribution are highly likely to be irrelevant. By fitting a log-normal distribution, we can exclude the 5 percent of users who lie more than 2 standard deviations from the centre of the distribution, which helps filter out accounts of little interest. An example of the code for this below:

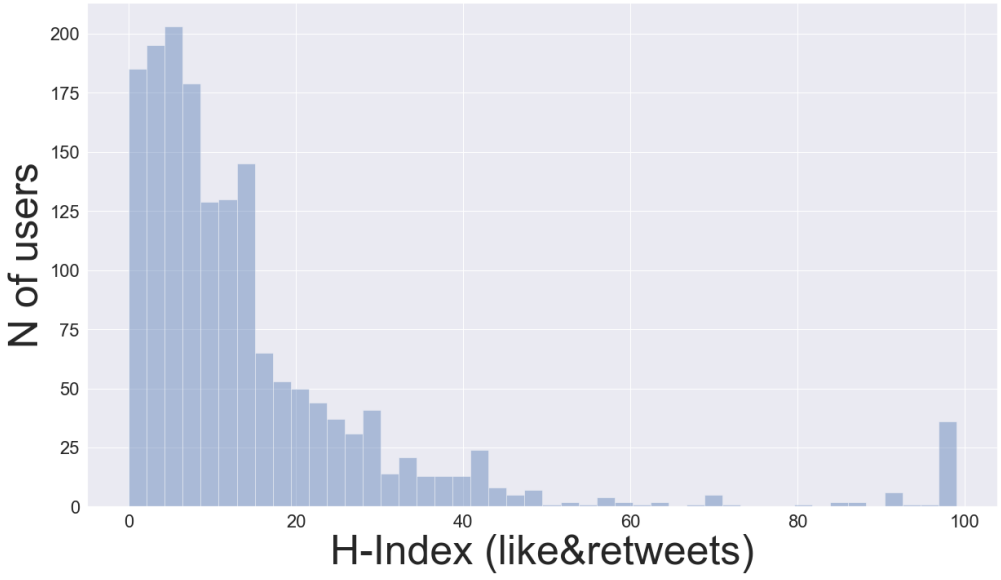
```
from src.graph_database import graphdb as gdb

# This function calculate the chisquare for each user
no_loners = gdb.get_chi2(users_df)

#We can then classify each user as an inlier or outlier based on their chisquare value
inliers = no_loners[no_loners['chi2']<6.18]
outliers = no_loners[no_loners['chi2']>6.18]
```

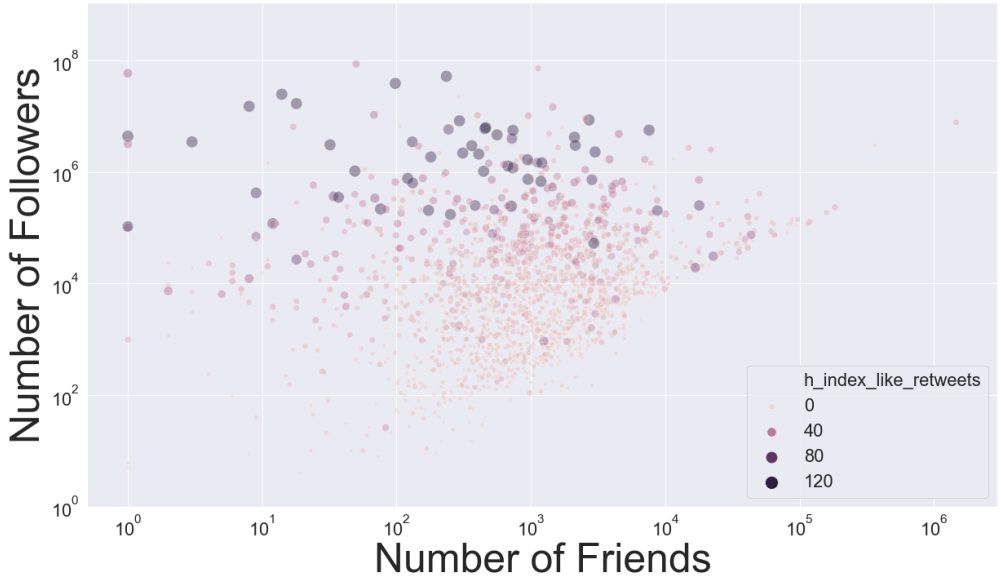
3.4 The Twitter h-index

Another metric that we can use to gain further insight into the users we are investigating derives from their tweets and how well they engage with their audience.

From the world of academic publishing, we have (controversially) adopted the use of the *h-index*, which is a measure of the volume of engagement for each person.

In the context of Twitter, the h-index of a user is the largest number of tweets, N, that have at least N retweets or likes.





3.5 Topic Modelling: Hashtag analysis

The number and name of topics chosen for the word2vec analysis comes from dimensional reduction of the hashtags from the tweets. Looking at the graph, the number of topics can be chosen as where the steepest decline is (the elbow of the graph). After the list of dimensionally reduced hashtags gets produced, it is good to assign hashtags to the topics manually. Once a topic list has been created, these terms can be searched for within the tweet text and keywords are found. Once tweets have been assigned a keyword they can be randomly sampled to balance the topics out. Obviously only sample as many tweets as there are in the topics and balance the classes. It is good to plot the topics using word cloud plots for a sense check.

3.6 Topic Modelling: Doc2vec

There are three models in the notebook:

- Distributed bag of words model (model_dbow), analogous to continuous bag of words in word2vec
- Distributed memory (model_dmm), analogous to the 'skip-gram' model in word2vec
- Combined (model_new), the combined vectors of the above models

Run all three models and see what comes up with the best accuracy or F1 score and use this model for all future exploration.

To test the robustness of the doc2vec network that has been created, you can do some similarity checks and some vector arithmetic. If this doesn't make sense more training data may be required.

3.7 Topic Modelling: Applying to other tweets

Load files that contain all the tweets and apply the model. Some basic analysis can be performed with some simple plots. The output file can then be saved and fed into the graph database for filtering.

3.8 Graph Database: Install and configure Neo4j

Neo4j is a popular graph database, it stores the data in the same way it is layed out on the graph. This makes queerying the database very efficient. Queeries to a neo4j database are made in the cypher language. Cypher is deliberately similar to SQL to make it easy to use for people who are familiar with SQL style relational databases. Neo4j has a desktop development environment. It also has wrappers and interfaces with all the popular scripting languages.

Before running the database notebook one needs to have the Neo4j desktop development environment up and running. The Neo4j development app can be downloaded from here:

https://neo4j.com/download/?ref=try-neo4j-lp

There is also and online "sandbox" version, this is actually quite useful because it contains an example of twitter analytics. Neo4j can also be run from a cypher terminal, which comes with the desktop installation. We need to install the APOC and Graph Data Science Library plugins. These can be done via the desktop app.

3.9 Graph Database: Importing Data

On the desktop, create a new database by clicking "add database" and selecting "create local database". Then click "create", then "start", a new window will open.

Later we will want to be able to read in and write to different file formats. To do this we need to locate the configuration file and add the following couple of lines to location/conf/neo4j.conf

```
# Enable loading of json files
apoc.import.file.enabled=true

# Enable exportation of files
apoc.export.file.enabled=true
```

CHAPTER

FOUR

REBUILD THE DOCS!

To rebuild the docs, first ensure the required packages are installed:

```
pip install Sphinx sphinx_rtd_theme
```

Once these are installed, navigate to the `/docs` folder of the repo and build the docs with commands:

```
sphinx-apidoc -o source/ ../src --force
make html
```

If you have an installation of LaTeX on your machine, you can also generate a pdf with:

```
make latexpdf
```

CHAPTER

FIVE

SRC PACKAGE

5.1 Subpackages

5.1.1 src.data package

Submodules

src.data.H_Index_tools module

src.data.H_Index_tools.**hindex**(*citations*)

Function that calculates an H-Index based on number of retweets and likes for each tweet. The range of values of the H-Index will vary based on the maximum number of tweets dowloaded for each user. For example, if we set the n_tweets to 200, a user that has 200 tweets that have been retweeted and liked 200 times or more will have an H-Index of 199. A user without likes or retweets will have an H-Index of 0.

Parameters **citations** (*list*) – A list of the sum of likes and retweets per tweet per user.

Returns **idx** – H-Index value

Return type int

src.data.H_Index_tools.**loop_csv_H_index**(*src_dir*, *dest_dir*, *keyword*)

Function that loops through the friends tweets downloaded with the API and calculate the H-Index for each user. Retweets originally tweeted by other users are not used in the H-Index calculation

Parameters

- **src_dir** (*str*) – File path where all the friends tweets are stored
- **dest_dir** (*str*) – File path where the like&retweet dataframe count will be stored
- **keyword** (*str*) – Initial keyword used to search for journalists

Returns

- **df_like_count** (*DataFrame object*) – Sum of likes and retweets for each user
- **hdict_df** (*DataFrame object*) – Dataframe of screen_name and H-Index

src.data.api_tweepy module

src.data.api_tweepy.**connect_API**(*keys_file*)
Load twitter API credentials and return a tweepy API instance with which to make API calls.

Parameters **keys_file** (*str*) – The path to a **.json* file containing the twitter API credentials to be used. See the notebook *store_twitter_credentials_as_json.ipynb* for more.

Returns **api**

Return type tweepy API object

src.data.api_tweet_tools module

src.data.api_tweet_tools.**batch_request_user_timeline**(*api*, *user_list*, *filepath*, *chunk_size=500*, *n_tweets=200*, *api_delay=0*)
Function to sample the most recent tweets (up to 200) from the timelines of a list of users.

Parameters

- **api** (*tweepy.API instance*) – The API authorisation hook used to make the requests.
- **user_list** (*list*) – A list of user screen-names to request tweets from.
- **filepath** (*str*) – The location to store csv file outputs.
- **chunk_size** (*int*) – The number of users to collect in each subset of data.
- **n_tweets** (*int*) – The number of tweets to be requested from each user's timeline. Max is 3200.
- **api_delay** (*float*) – Number specifies length of sleep delay to include between requests. Use this to stop the code tripping up on the API rate limits.

src.data.api_tweet_tools.**request_user_timeline**(*api*, *user*, *api_delay=0*, *n_tweets=200*)
Function to make a single API request from a user's timeline. If no keywords are provided, the request will return tweepy defaults, which will be the 20 most recent tweets of the user.

Parameters

- **api** (*tweepy.api.API object*) – The API object to be used to make the request.
- **user** (*str or int*) – Either the screen-name or user-ID for the user whose timeline is being requested.
- **api_delay** (*float*) – Value represents the delay in seconds added after each API request. Is used to pad the processing time to keep within the API rate limits.
- **n_tweets** (*int*) – The number of tweets to be retrieved from a user's timeline. Max is 3200.

Returns

- **TL_tweets** (*list*) – A list where each tweet retrieved is represented by a dict.
- **TODO** (*Check if try/except gets stuck if the api throws an error. I think with*)
- *current code iterating through the timeline would result in trying the same*
- *request over and over again.*

src.data.api_tweet_tools.**wrangle_tweets_into_df**(*tweet_list*)
Takes the output of *request_user_timeline()* or *batch_request_user_timeline()* and parses the results into a Pandas DataFrame.

Parameters **tweet_list** (*A list of dicts*) – The list of tweets. The attributes of each tweet are represented in a dict.

Returns **tweet_df** – The dataframe with the parsed tweet object info.

Return type Pandas DataFrame object

src.data.api_user_tools module

src.data.api_user_tools.**batch_request_user_info**(*api*, *user_list*)
Lookup a batch of users using the lookup method of the API.

Parameters

- **api** (*tweepy API instance*) – The API object to be used to make the request.
- **user_list** (*list*) –
- **list of strings or ints containing either the handles or ID numbers** (*A*) –
- **the users to look up.** (*of*) –

Returns A list of tweepy user objects. See 'INSERT FUNCTION' for turning these objects into a useful dataframe

Return type full_list

src.data.api_user_tools.**query_user_relationship**(*api*, *userA*, *userB*)
Asks the twitter API if user X follows user Y, and vice versa.

Parameters

- **api** (*tweepy API instance*) – The API object to be used to make the request.
- **userA** (*str*) – the first username to check friendship status of
- **userB** (*str*) – the second username to check friendship status with A

Returns **statuses** – contains whether A follows B, and whether B follows A.

Return type dict

src.data.api_user_tools.**request_user_info**(*api*, *screen_name=None*, *user_id=None*, *api_delay=True*)
Request user info from the twitter API using the *get_user* method based on either the user's twitter handle or ID number.

Parameters

- **api** (*tweepy.api.API object*) – The API object to be used to make the request.
- **screen_name** (*str*) – Twitter handle of user to request information about. One of *screen_name* or *user_id* must be provided.
- **user_id** (*int*) – ID number of user to request information about. One of *screen_name* or *user_id* must be provided.
- **api_delay** (*bool*) – If true, add a sleep delay to pace API requests within rate limits. Rate limit info for user profile requests is 900/15 mins

Returns **user_info** – A python dict where user information is stored under named keys.

Return type dict

src.data.api_user_tools.**tweepy_user_to_dataframe**(*user*)
Take in a tweepy User object and parse the data contained within to return a single-row Pandas dataframe containing the same attributes as columns.

Parameters **user** (*tweepy.models.User object*) – The tweepy user information object to be parsed into a dataframe.

Returns

- **user_df** (*DataFrame object*) – A Pandas dataframe with one row and columns for each attribute.
- *TODO*
- —
- *-Rework this function, and maybe move it completely, to work with dicts/*
- *json format*
- *-Abstract the selection of important attributes to the user level and*
- *improve interactivity/customisation.*

src.data.data_cleanup module

src.data.data_cleanup.**clean_API_dataframe**(*API_df*)
A function to take tweet data generated by the twitter API and return a standardised dataframe with a consistent set of fields.

Parameters **API_df** (*Pandas DataFrame*) – The API tweet data to be processed and transferred.

Returns **standard_df** – A dataframe containing tweets with a standardised set of fields.

Return type Pandas DataFrame

src.data.data_cleanup.**clean_text**(*text*)
Function which uses regex on a piece of text to return only alphanumeric characters.

Parameters **text** (*str*) – The text string to be cleaned up.

Returns **cleaned_text** – The text with non-alphanumeric characters removed.

Return type str

src.data.data_cleanup.**clean_twint_dataframe**(*twint_df*)
A function to take tweet data generated by twint and return a standardised dataframe with a consistent set of fields.

Parameters **twint_df** (*Pandas DataFrame*) – The twint tweet data to be processed and transferred.

Returns **standard_df** – A dataframe containing tweets with a standardised set of fields.

Return type Pandas DataFrame

src.data.data_cleanup.**init_cleaned_tweet_df**()
Initialise and return an empty dataframe with a standard, curated set of fields.

Returns **standard_df** – A dataframe containing tweets with a standardised set of fields.

Return type Pandas DataFrame

src.data.data_cleanup.**mentions_to_df**(*df*)
Function to search through twint tweet data for user mentions and return a dataframe containing a tweet id and mentioned username in each row.

Parameters **twint_df** (*Pandas DataFrame*) –

Returns **tweet_mentions**

Return type Pandas DataFrame

src.data.data_cleanup.**populate_user_df**(*user_data*)
Take a series of tweepy user objects and transform them into a dataframe.

Parameters **user_data** (*list*) – List of tweepy.User objects

Returns **user_df** – Cleaned dataframe with user data.

Return type Pandas DataFrame

src.data.data_cleanup.**twint_mentions_to_df**(*twint_df*)
Function to search through twint tweet data for user mentions and return a dataframe containing a tweet id and mentioned username in each row.

Parameters **twint_df** (*Pandas DataFrame*) –

Returns **tweet_mentions**

Return type Pandas DataFrame

src.data.graphdb module

src.data.graphdb.**boost_graph**(*niter*, *nsample*, *fields*, *exponents*, *kwargs*)
Function to randomly subsample users in the graph and to identify who these people are following if they are following other people already in the graph then follower edges are drawn in new users are not added to the graph.

niter [int] number of boosting iterations

nsample [int] number of random samples to take on each iteration

fields [list of str] fields on which sampling should be weighted (must be columns in dataframe)

exponents [list of floats] exponents to determine strength of weighting, +ve => upweighted, -ve => downweighted default quadratic weights is 2

kwargs [dict] keywords for twint

Returns

Return type Void

src.data.graphdb.**excise_outliers**(*outlier_list*, *graph*)
Delete list of users and all their connections from a graph

outlier_list [list of str] list of user names of users to be deleted

graph : graph

Returns

Return type void

src.data.graphdb.**filter_users_by_keywords**(*keywords*, *graph*)
Function to select only twitter users with certain keywords in their bio (or screen name)

keywords [list of str] list of keywords to select users by

graph : graph

Returns **df** – A dataframe containing the screen names of filtered users

Return type pandas dataframe

src.data.graphdb.**get_chi2**(*df*)
Calculate the chi2 of users based on the assumption that follower and friend numbers are lognormally distributed

df [pandas dataframe] contains user information

Returns **no_loners** – contains augmented user information with singularities (users with zero friends) removed should be same dimensions as df

Return type pandas dataframe

src.data.graphdb.**get_graph**(*new_graph=True*)
Load an existing neo4j database or create a new one and return a graph object

new_graph [boolean] True if a fresh graph is wanted, i.e. if the user wants to clear the database

Returns **graph**

Return type a py2neo graph class

src.data.graphdb.**get_multiple_weighted_sample**(*ranked_df*, *sample_size*, *fields*, *weight_exponents*)

src.data.graphdb.**get_posts**(*graph*)
function to draw edges between Tweet and Person nodes based on user id

graph : graph

Returns

Return type Void

src.data.graphdb.**get_talk_about_edges**(*graph*)
Draw TALKS_ABOUT edges on the graph connecting Person nodes. Requires Tweet nodes, POSTS and MENTIONS edges to already be in the graph

graph : graph

Returns

Return type Void

src.data.graphdb.**get_weighted_sample**(*ranked_df*, *sample_size*, *field*, *weight_exponent=2*)
Gets a weighted random sample of users

ranked_df [panadas dataframe] contains the user names and the parameters on which to weight

sample_size [int] desired sample size

field [str] name of column containing weights

weight_exponent [float] the higher the weight exponent the stronger the weighting +ve => upweighting -ve => downweighting default is quadratic

src.data.graphdb.**load_existing_users**(*filename*, *graph*)
function to load user information contained in a '*.csv' file into the graph database to be used when users ARE already in the database

filename [str] name of file containing user information

graph : graph

Returns

Return type Void

src.data.graphdb.**load_friends**(*filename*, *graph*, *new=False*)
Load friend information from a '*.csv' file into the databse, draw FOLLOWS edges

filename [str] location of file with friend information

graph [graph] graph onto which edges will be drawn

new [boolean] if True new nodes will be drawn if friends cannot be found in database if False only edges between existing Person nodes are drawn

Returns

Return type Void

src.data.graphdb.**load_mentions**(*filename*, *graph*)
Loads mention information from a file and draws edges connecting Tweet and Person nodes

filename [str] location of '*.csv' file containing mentions information

graph [graph] graph on which edges will be drawn

Returns

Return type Void

src.data.graphdb.**load_tweets**(*filename*, *graph*)
Load tweets into the graph database as nodes. Uses 'CREATE' therefore duplicate nodes may be created if items are already in the database.

filename [str] The path to a '*.csv' file containing tweet data

graph [graph] The graph database into which the tweet data will be loaded

Returns

Return type void

src.data.graphdb.**load_users**(*filename*, *graph*)
function to load user information contained in a '*.csv' file into the graph database to be used when users are not already in the database

filename [str] name of file containing user information

graph : graph

Returns

Return type void

src.data.graphdb.**run_pagerank**(*nodelist*, *edgelist*, *graph*, *new_native_graph=True*)
Runs the page rank algorithm on the graph network

nodelist [list of str] list of node types to include in projection

edgelist [list of str] list of edge types to include in projection

graph [graph] graph on which algorithm is to be run

new_native_graph [boolean] If True makes a new native projection of the graph If False uses the existing projection

Returns **df** – dataframe containing a list of user names and their rank

Return type pandas DataFrame

src.data.journalists module

src.data.journalists.**get_handles_by_keyword**(*kwd*)
Function which wraps around the webpage retrieval and parsing functions to provide a list of twitter handles for journalists relating to a topic provided by *kwd*.

Parameters **kwd** (*str*) – String containing the search term to be used. This should be a single word at present.

Returns **handle_list** – A list of strings containing the twitter handles scraped over all pages of search results.

Return type list object

src.data.journalists.**get_handles_from_contents**(*contents*)
Use regex to identify twitter handles - which start with "@" - within the HTML of the div contents found using "parse_divs_in_webpage()".

Parameters

- **contents** (*list object*) – The parsed HTML contents of the page. See *parse_divs_in_webpage()* for more.
- **Results** –
- **-------** –
- **handles** (*list object*) – A list of strings. Each entry is the text preceded by an @ symbol, presumed to be a hashtag.

src.data.journalists.**get_webpage**(*url*)
Function wrapped around the urllib method "urlopen" to retrieve webpages. Provides an additional check for instances when urlopen returns None.

Parameters **url** (*string*) – The web address from which to retrieve a page.

Returns **html** – A HTTPResponse object which contains the webpage as well as status response codes and page meta-data.

Return type object

src.data.journalists.**iterate_func_over_pages**(*url*, *func*)
Facilitates execution of regex-based functions over multiple pages of search results.

Parameters

- **url** (*string*) – The url of the search term, the page info will be appended to this.
- **func** (*function*) – Function containing regex used to extract items from parsed HTML. The output of func should be formatted as a list.

- **args** (*tuple*) – The arguments which are taken by func.

Returns **complete_out** – List containing items retrieved from all pages by func.

Return type list

src.data.journalists.**parse_divs_in_webpage**(*webpage*)
Use BeautifulSoup to parse HTML in the webpage, and return all the "div" elements on the page.

Parameters **webpage** (*object*) – A HTTPResponse object

Returns **div_list** – A list of all the div objects found using BeautifulSoup.

Return type list

src.data.pipeline_setup module

src.data.pipeline_setup.**build_data_dir**(*fp*)
Function to which checks for standard-format data directory at the location specified. If directory or any of its sub-folders do not exist, this creates them.

Parameters **fp** (*str*) – The filepath (preferably absolute) to the parent location of the data directory.

src.data.pipeline_setup.**install_nltk_data**()
Download and install/update corpora from nltk which are used in the project.

src.data.twint_tools module

src.data.twint_tools.**join_friends_csv**(*list_journalists*, *keyword*)

src.data.twint_tools.**join_tweet_csv**(*user_list*, *keyword*)

src.data.twint_tools.**twint_in_queue**(*target*, *num_threads*, *queue_items*, *args=()*, *kwargs={}*)
Function for executing twint requests in threads.

Parameters

- **target** (*func*) – The function each thread will be executing.
- **num_threads** (*int*) – The number of threads to distribute the queue items to.
- **queue_items** (*list*) – A list of variables to be passed to the target function via the queue.
- **args** (*tuple*) – A set of arguments for the target function which remain constant between iterations
- **kwargs** (*dict*) – The keyword arguments to be taken by the target function.

Module contents

5.1.2 src.features package

Submodules

src.features.build_features module

Module contents

5.1.3 src.graph_database package

Submodules

src.graph_database.graphdb module

src.graph_database.graphdb.**add_property**(*property_name*, *dataframe*, *graph*)
Routine to add a property to nodes in the graph.

Parameters

- **property_name** (*str*) – The name of the property to be added to the node, should be the name of a column heading in the dataframe.
- **dataframe** (*pandas dataframe*) – Dataframe containing at least the user names and the property to be added to these users
- **graph** (*graph*) –

Returns

Return type Void

src.graph_database.graphdb.**boost_graph**(*niter*, *nsample*, *fields*, *exponents*, *pagerank_params*, *keyword*, *kwargs*)
Function to randomly subsample users in the graph and to identify who these people are following if they are following other people already in the graph then follower edges are drawn in new users are not added to the graph.

Attempting to find all the friends of friends may result in downloading hundreds of thousands or millions of profiles. The network gets exponentially bigger at each level of abstraction. We can avoid this by selecting a random sample of users in our database and seeing if they are following anyone else in our database. We can weight this random selection by, for example, their previously determined rank or the number of friends or followers they have. By repeating this process several times we can build complexity into our graph.

Parameters

- **niter** (*int*) – number of boosting iterations
- **nsample** (*int*) – number of random samples to take on each iteration
- **fields** (*list of str*) – fields on which sampling should be weighted (must be columns in dataframe)
- **exponents** (*list of floats*) – exponents to determine strength of weighting, +ve => upweighted, -ve => downweighted default quadratic weights is 2
- **pagerank_params** (*tuple*) – parameters for pagerank
- **kwargs** (*dict*) – keywords for twint

Returns

Return type Void

src.graph_database.graphdb.**excise_outliers**(*outlier_list*, *graph*)
Delete list of users and all their connections from a graph

Parameters

- **outlier_list** (*list of str*) – list of user names of users to be deleted
- **graph** (*graph*) –

Returns

Return type void

src.graph_database.graphdb.**filter_by_topic**(*keyword*, *graph*)
Function returns a list of individuals associated with a particular topic.

Parameters

- **keyword** (*str*) – The topic one wishes to filter on
- **graph** (*graph*) –

Returns **names** – A list of user names associated with that topic

Return type pandas dataframe

src.graph_database.graphdb.**filter_users_by_keywords**(*keywords*, *graph*, *without=False*)
Function to select only twitter users with certain keywords in their bio (or screen name)

Parameters

- **keywords** (*list of str*) – list of keywords to select users by
- **graph** (*graph*) –
- **without** (*boolean*) – If set to true then returns a list of users without keywords in bio

Returns **df** – A dataframe containing the screen names of filtered users

Return type pandas dataframe

src.graph_database.graphdb.**get_chi2**(*df*)
Calculate the chi2 of users based on the assumption that follower and friend numbers are lognormally distributed

Parameters **df** (*pandas dataframe*) – contains user information

Returns **no_loners** – contains augmented user information with singularities (users with zero friends) removed should be same dimensions as df

Return type pandas dataframe

src.graph_database.graphdb.**get_chi2_H_index**(*df*)
Routine to calculate the chi2 of each user based on the H index

Parameters **df** (*pandas dataframe*) – Dataframe containing le H-index of users

Returns **no_loners** – Dataframe with added column representing the chi2 of users. Users with an H-index of zero are excised from the dataframe.

Return type pandas dataframe

src.graph_database.graphdb.**get_graph**(*new_graph=True*)
Load an existing neo4j database or create a new one and return a graph object

Parameters **new_graph** (*boolean*) – True if a fresh graph is wanted, i.e. if the user wants to clear the database

Returns graph

Return type a py2neo graph class

src.graph_database.graphdb.**get_multiple_weighted_sample**(*ranked_df*, *sample_size*, *fields*, *weight_exponents*)

src.graph_database.graphdb.**get_posts**(*graph*)
function to draw edges between Tweet and Person nodes based on user id

Parameters **graph** (*graph*) –

Returns

Return type Void

src.graph_database.graphdb.**get_talk_about_edges**(*graph*)
Draw TALKS_ABOUT edges on the graph connecting Person nodes. Requires Tweet nodes, POSTS and MENTIONS edges to already be in the graph

Parameters **graph** (*graph*) –

Returns

Return type Void

src.graph_database.graphdb.**get_weighted_sample**(*ranked_df*, *sample_size*, *field*, *weight_exponent=2*)
Gets a weighted random sample of users

Parameters

- **ranked_df** (*panadas dataframe*) – contains the user names and the parameters on which to weight
- **sample_size** (*int*) – desired sample size
- **field** (*str*) – name of column containing weights
- **weight_exponent** (*float*) – the higher the weight exponent the stronger the weighting +ve => upweighting -ve => downweighting default is quadratic

src.graph_database.graphdb.**load_existing_users**(*filename*, *graph*)
function to load user information contained in a '*.csv' file into the graph database to be used when users ARE already in the database

Parameters

- **filename** (*str*) – name of file containing user information
- **graph** (*graph*) –

Returns

Return type Void

src.graph_database.graphdb.**load_friends**(*filename*, *graph*, *new=False*)
Load friend information from a '*.csv' file into the databse, draw FOLLOWS edges

Parameters

- **filename** (*str*) – location of file with friend information
- **graph** (*graph*) – graph onto which edges will be drawn

- **new** (*boolean*) – if True new nodes will be drawn if friends cannot be found in database if False only edges between existing Person nodes are drawn

Returns

Return type Void

src.graph_database.graphdb.**load_mentions**(*filename*, *graph*)
Loads mention information from a file and draws edges connecting Tweet and Person nodes

Parameters

- **filename** (*str*) – location of '*.csv' file containing mentions information
- **graph** (*graph*) – graph on which edges will be drawn

Returns

Return type Void

src.graph_database.graphdb.**load_topics**(*fn_topics*, *graph*, *threshhold*)
Function to read in file containing the results of the topic modelling and to load it onto the graph. Topics are assigned to nodes and edges drawn between users weighted by their association with that topic. (:Person)-[:TWEETS_ABOUT]->(:Topic)

Parameters

- **fn_topics** (*str*) – location of file containing users and their topic weights
- **graph** (*graph*) –
- **threshhold** (*float*) – The minimum weight to assign to a TWEETS_ABOUT edge

Returns

Return type Void

src.graph_database.graphdb.**load_tweets**(*filename*, *graph*)
Load tweets into the graph database as nodes. Uses 'CREATE' therefore duplicate nodes may be created if items are already in the database.

Parameters

- **filename** (*str*) – The path to a '*.csv' file containing tweet data
- **graph** (*graph*) – The graph database into which the tweet data will be loaded

Returns

Return type void

src.graph_database.graphdb.**load_users**(*filename*, *graph*)
function to load user information contained in a '*.csv' file into the graph database to be used when users are not already in the database

Parameters

- **filename** (*str*) – name of file containing user information
- **graph** (*graph*) –

Returns

Return type void

src.graph_database.graphdb.**order_conversations**(*graph*)
Depending on the source of data (the Twitter API or twint) tweets either contain a "conversation_id" or a "reply_to_status_id". These are different but related numbers. The "conversation_id" is the id of the first tweet

in the conversation, the "reply_to_status_id" is the id of the tweet in the conversation preceding the current tweet. Tweets also contain timestamp information, thus even if we don't have the "reply_to_status_id" we can order tweets in the conversation.

Parameters **graph** (*graph*) –

Returns

Return type Void

src.graph_database.graphdb.**run_pagerank**(*nodelist*, *edgelist*, *graph*, *new_native_graph=True*)

Runs the page rank algorithm on the graph network

Parameters

- **nodelist** (*list of str*) – list of node types to include in projection
- **edgelist** (*list of str*) – list of edge types to include in projection
- **graph** (*graph*) – graph on which algorithm is to be run
- **new_native_graph** (*boolean*) – If True makes a new native projection of the graph If False uses the existing projection.

Returns **df** – dataframe containing a list of user names and their rank

Return type pandas DataFrame

Module contents

5.1.4 src.models package

Submodules

src.models.predict_model module

src.models.train_model module

Module contents

5.1.5 src.topics package

Submodules

src.topics.hashtag_analysis module

src.topics.hashtag_analysis.**SVD_on_vectors**(*vectors*, *n_components=100*, *SVD_kwargs={}*, *plot=True*)

Use TruncatedSVD on vectorised documents as dimensionality reduction to generate topics.

Parameters

- **vectors** (*Pandas DataFrame.sparse*) – A sparse, count-vectorized dataframe to be processed.
- **n_components** (*int*) – The number of dimensions to reduce the feature-space to
- **SVD_kwargs** (*dict*) – Apart from n_components, any other kwargs which can be passed to sklearn.decomposition.TruncatedSVD

- **plot** (*bool*) – If True, plots a bar chart showing the variance associated with each component.

Returns

- **vectors_reduced** (*Pandas DataFrame*) – The input sparse matrix, vectors, transformed onto the reduced-dimensions feature-space.
- **components** (*Pandas DataFrame*) – The vectors which map each component onto the original feature-space.

src.topics.hashtag_analysis.**check_for_matches**(*tag_list*, *topic_list*, *number_other_topic*)
Compares a list of hashtags in tweets against those in each topic.

Parameters

- **tag_list** (*Pandas DataFrame column*) – A column containing hashtags
- **topic_list** (*list of lists*) – Each list in the list contains hashtags belonging to one topic.
- **number_other_topic** (*interger*) – The number to declare the 'other' topic, for example, -1, or one plus the number of topics

Returns **matches** – Series of a numbers, where the number corresponds to which topic the hastags fall into

Return type Pandas series

src.topics.hashtag_analysis.**check_keyword_matches**(*text*, *topic_list*)
Compares a list of 'keywords' in tweets.

Parameters

- **text** (*Pandas dataframe column*) – containing the text that the keywords search is to be performed on
- **topic_list** (*a list*) – a list of topics

Returns **matches** – Series of a numbers, where the number corresponds to which topic the keywords fall into

Return type Pandas series

src.topics.hashtag_analysis.**get_ranked_wordcounts**(*words*)
Use the Counter class from Python collections to count the frequency of each word across a set of documents, represented by entries in a Pandas Series.

Parameters **words** (*Pandas Series*) – A Series containing entries which are either strings or lists of strings.

Returns **counts_sorted** – A list of tuples, where tuple[0] is a word and tuple[1] is its' frequency.

Return type list

src.topics.hashtag_analysis.**label_data_by_topic**(*topic_space*, *threshold*)
Takes the SVD transformed data vectors and assigns a label based on most significant axis. Threshold can be used to further filter out noise.

Parameters

- **topic_space** (*Pandas DataFrame*) – DF which contains the results of dimension reduction by SVD.
- **threshold** (*float*) – The value above which label assignment is considered valid. To be used to remove spurious association of tweets with a topic.

Returns **labels** – Has one column which contains the topic label assigned to each tweet.

Return type Pandas DataFrame

src.topics.hashtag_analysis.**make_topic_keywords_from_svd**(*svd_vectors*, *threshold=0.2*)

Takes the results of TruncatedSVD and generates a list of keywords for each SVD component based on the component scores.

Parameters

- **svd_vectors** (*Pandas DataFrame*) – Dataframe containing the scores of each feature in the original space for each SVD component.
- **threshold** (*float*) – A significant threshold to apply on the selection of relevant keywords. Only keywords with a score greater than the threshold are returned for each component.

Returns **topic_kwds** – Dataframe where the index is the topic number, and the chosen keywords are contained as a list in the *keywords* column. Topics for which no keywords were returned are dropped from the returned dataframe.

Return type Pandas DataFrame

src.topics.hashtag_analysis.**produce_random_sample**(*df*, *df_column*, *number_of_samples*)

Pro.

Parameters

- **df** (*Pandas dataframe*) –
- **df_column** (*Pandas dataframe column*) –
- **number_of_samples** (*number to sample in each group*) –

Returns **sample_df** – dataframe containing random samples from original dataframe

Return type Pandas dataframe

src.topics.hashtag_analysis.**vectorize_wordlists**(*words_lists*)

Generate sparse matrix of word-counts from Pandas series where each entry is a list of words.

Parameters **words_lists** (*Pandas Series*) – A series of lists containing words to be one-hot encoded.

Returns **observations** – A sparse dataframe, where each row is a vectorised representation of a document.

Return type Pandas DataFrame.sparse

src.topics.preprocessing module

src.topics.preprocessing.**tokenize_text**(*text*, *x*)

Convert string *text* into a list of word tokens. Removes stopwords and words that are too short.

text [str] The text string to be tokenized.

x [interger or variable] the minimum length of words to be included

tokens : list of strings[InternetShortcut]

URL=https://github.com/S2DSLondon/Aug20_Ditchley/stargazers

text broken up into tokens.

src.topics.topic_modelling module

src.topics.topic_modelling.**clean_text**(*text*)

Parameters **text** (*str*) – test to clean

Returns **text** – cleaned text

Return type srt

src.topics.topic_modelling.**get_vectors**(*model*, *tagged_docs*)
Building feature vectors

Parameters

- **model** (*the model*) – (either: model_dbow, model_dmm, model_new)
- **tagged_docs** (*tagged documents*) –

Returns **tokens**

Return type targets, regressors

src.topics.topic_modelling.**get_vectors_apply**(*model*, *docs_to_classify*)

Parameters

- **model** (*the model*) – (either: model_dbow, model_dmm, model_new)
- **docs_to_classify** (*documents to classify*) –

Returns **tokens**

Return type regressors

src.topics.topic_modelling.**vec_for_learning**(*model*, *tagged_docs*)
Building the final vector feature for the classifier

Parameters

- **model** (*the model*) – (either: model_dbow, model_dmm, model_new)
- **tagged_docs** (*tagged documents*) –

Returns **tokens**

Return type targets, regressors

Module contents

5.2 Submodules

5.3 src.plots module

src.plots.**make_tSNE_projection_of_SVD**(*df*, *kwargs={}*)
Project SVD-transformed data onto two dimensions for visualisation.

Parameters

- **df** (*Pandas DataFrame*) – A dataframe with the reduced-dimensionality vector representation of tweets.

- **kwargs** (*dict*) – Optional keyword arguments which can be fed to sci-kit learn's TSNE model.

Returns **coords** – The x- and y-coordinates of each tweet in the projected 2d-space.

Return type Pandas DataFrame

src.plots.**plot_H_index**(*df*)
Function that produces two subplots to visualise H-Index results: Subplot 1: Distribution of H-Index. The range of x axis depends on the number of tweets dowloaded for each user Subplot 2: Correlation between n. of followers and n. of friends with the H-Index represented by different point size and color

Parameters **df** (*Pandas DataFrame*) – A dataframe which needs to have columns: *h-index_like&retweets*, *user_friends_n* and *user_followers_n*

Returns **fig** – A reference to a Seaborn/Matplotlib figure.

Return type Figure

src.plots.**plot_tsne_projection**(*df*, *label_str*, *plotly=False*, *kwargs={}*)
Function to visualise a tSNE projection in 2 dimensions.

Parameters

- **df** (*Pandas DataFrame*) – A dataframe that must contain columns *tsne_x* and *tsne_y* generated by *make_tSNE_projection_of_SVD()* as well as a column which can be used for labels.
- **label_str** (*str*) – The name of the column in *df* to be used as labels for colouring.
- **plotly** (*bool*) – If True, use plotly for the visualisation. If False, use matplotlib.

Returns **fig** – A reference to either a plotly or a matplotlib figure.

Return type Figure

src.plots.**plot_user_inliers**(*inliers*, *outliers*)
Create a plot of the user distribution in terms of friends and followers, colour-coded by inliers and outliers.

Parameters

- **inliers** (*Pandas DataFrame*) – A dataframe which contains a list of users identified as inliers, and their numbers of friends and followers.
- **outliers** (*Pandas DataFrame*) – A dataframe which contains a list of users identified as outliers, and their numbers of friends and followers.

Returns **fig** – A reference to a matplotlib figure

Return type Figure

src.plots.**wordcloud_plot**(*text*, *filename=None*, *cloudargs={}*)
Displays and saves a wordcloud plot.

Parameters

- **text** (*str*) – The text to be parsed in the wordcloud.
- **filename** (*str*) – If used, also saves a copy of the wordcloud to file as specified in the string.
- **cloudargs** (*dict*) – A list of keywords that can be passed to WordCloud if deviating from defaults.

Returns **fig** – A hook to the figure to make tweaks if wanted.

Return type matplotlib figure

5.4 Module contents

CHAPTER

SIX

PYTHON MODULE REQUIREMENTS

Table 1: Modules

Module		Version
aiodns		2.0.0
aiohttp		3.6.2
aiohttp-socks		0.5.3
alabaster		0.7.12
argon2-cffi		20.1.0
async-timeout		3.0.1
attrs		20.1.0
Babel		2.8.0
backcall		0.2.0
beautifulsoup4		4.9.1
bleach		3.1.5
blis		0.4.1
boto		2.49.0
boto3		1.14.47
botocore		1.17.47
catalogue		1.0.0
cchardet		2.1.6
certifi		2020.6.20
cffi		1.14.2
chardet		3.0.4
click		7.1.2
colorama		0.4.3
cryptography		3.0
cycler		0.10.0
cymem		2.0.3
Cython		0.29.14
decorator		4.4.2
defusedxml		0.6.0
docker		4.3.1
docutils		0.15.2
elasticsearch		7.9.1
english		2020.7.0
entrypoints		0.3
fake-useragent		0.1.11
gensim		3.8.3
geographiclib		1.50

continues on next page

Table 1 – continued from previous page

Module		Version
geopy		2.0.0
googletransx		2.4.2
idna		2.10
idna-ssl		1.1.0
imagesize		1.2.0
importlib-metadata		1.7.0
ipykernel		5.3.4
ipython		7.16.1
ipython-genutils		0.2.0
ipywidgets		7.5.1
jedi		0.17.2
Jinja2		2.11.2
jmespath		0.10.0
joblib		0.16.0
json5		0.9.5
jsonschema		3.2.0
jupyter-client		6.1.7
jupyter-core		4.6.3
jupyterlab		2.2.6
jupyterlab-server		1.2.0
kiwisolver		1.2.0
MarkupSafe		1.1.1
matplotlib		3.3.1
mistune		0.8.4
monotonic		1.5
multidict		4.7.6
murmurhash		1.0.2
nbconvert		5.6.1
nbformat		5.0.7
neotime		1.7.4
nest-asyncio		1.4.0
nltk		3.5
notebook		6.1.3
numpy		1.19.1
oauthlib		3.1.0
packaging		20.4
pandas		1.1.1
pandocfilters		1.4.2
pansi		2020.7.3
parso		0.7.1
pickleshare		0.7.5
Pillow		7.2.0
plac		1.1.3
preshed		3.0.2
prometheus-client		0.8.0
prompt-toolkit		2.0.10
py2neo		2020.0.0
pycares		3.1.1
pycparser		2.20

continues on next page

Table 1 – continued from previous page

Module		Version
Pygments		2.6.1
pyparsing		2.4.7
pyrsistent		0.16.0
PySocks		1.7.1
python-dateutil		2.8.1
pytz		2020.1
pywin32		227
pywinpty		0.5.7
pyzmq		19.0.2
regex		2020.7.14
requests		2.24.0
requests-oauthlib		1.3.0
s3transfer		0.3.3
schedule		0.6.0
scikit-learn		0.23.2
scipy		1.5.2
seaborn		0.10.1
Send2Trash		1.5.0
six		1.15.0
sklearn		0.0
smart-open		2.1.0
snowballstemmer		2.0.0
soupsieve		2.0.1
spacy		2.3.2
Sphinx		3.2.1
sphinx-rtd-theme		0.5.0
sphinxcontrib-applehelp		1.0.2
sphinxcontrib-devhelp		1.0.2
sphinxcontrib-htmlhelp		1.0.3
sphinxcontrib-jsmath		1.0.1
sphinxcontrib-qthelp		1.0.3
sphinxcontrib-serializinghtml		1.1.4
srsly		1.0.2
terminado		0.8.3
testfixtures		6.14.1
testpath		0.4.4
thinc		7.4.1
threadpoolctl		2.1.0
tornado		6.0.4
tqdm		4.48.2
traitlets		4.3.3
tweepy		3.9.0
twint		2.1.20
typing		3.7.4.3
typing-extensions		3.7.4.3
urllib3		1.25.10
wasabi		0.7.1
wcwidth		0.2.5
webencodings		0.5.1

continues on next page

Table 1 – continued from previous page

Module		Version
websocket-client		0.57.0
widgetsnbextension		3.5.1
wikipedia		1.4.0
wincertstore		0.2
wordcloud		1.8.0
yarl		1.5.1
zipp		3.1.0

CHAPTER

SEVEN

INDICES AND TABLES

- genindex
- modindex
- search

PYTHON MODULE INDEX

S

INDEX

J

L

M

O

P

Q

R

S

T

V

W