UNIVERSIDAD ESTATAL A DISTANCIA
LABORATORIO DE INVESTIGACION E INNOVACIÓN TECNOLÓGICA
PROYECTO PRO RED
VICERRECTORÍA DE INVESTIGACIÓN

Proyecto:
FakeNews

Informe y Reporte de Avances
Manual de Text2graph

Estudiante Vinculado:
Brayan Rodríguez Delgado

Coordinadores

Adriana Céspedes
Andrés Segura Castillo

Noviembre 2020

Tabla de Contenidos

1. Introducción:.....3
2. Información General:.....3
3. Estructura del proyecto:.....3
4. Instalación:.....3
    4.1 Dependencias:.....4
    4.2 Diagrama básico del prototipo:.....5
5. Extractor CrowdTangle (ct_extractor):.....5
    5.1. Objeto principal: get_cd_data(token).....5
    5.2 Objetos ctpost, post y links:.....6
    5.3 Objeto posts:.....7
    5.4 Objeto lists:.....8
    5.5 Objeto list:.....9
    5.6 Objeto leaderboard:.....9
    5.7 Objeto search:.....10
6. Tokenizador de textos (tokenizer_text):.....10
    6.1 tokenize_text.....11
7. Generador de grafos (data_graph):.....11
    7.1 Objeto data_graph.....12
    7.2 Funciones internas, no muestran resultados para el usuario:.....16
8. Capturador de texto html (html_text):.....17
    8.1. get_html_text:.....17
    8.2. search:.....17
9.Mejoras a realizar:.....18
10.Historial de Versiones:.....18
    Versión 0.0.2:.....18

Text2graph

1. Introducción:

Text2graph es una librería dirigida al análisis y visualización de grafos de conceptos, obtenidos de un texto, el cual puede ser directamente introducido por el usuario, mediante un enlace, leyendo un documento, una base de datos en .csv o bien capturando el texto por medio de CrowdTangle.

2. Información General:

Nombre proyecto: Text2graph
Descripción: Libería para la visualización y análisisd de grafos generados mediante textos.
Versión actual: 0.0.2 (Prototipo)
Autor o autores: Brayan Rodríguez Delgado <bradrd2009jp@gmail.com>
Institución: Laboratorio de Investigación e Innovación Tecnológica LIIT-UNED.
Coordinadores: Adriana Céspedes y Andrés Segura.

3. Estructura del proyecto:

El proyecto cuenta con cuatro módulos independientes entre sí, aunque algunos módulos pueden alimentarse de la salida de los otros, estos módulos son: el generador de token (*tokenizer_text*), el generador de grafos (*data_graph*), un estractor de datos para CrowdTangle (*ct_extractor*) y una librería para capturar texto mediante html (*html_text*). La organización de la carpeta principal del proyecto presenta los siguientes archivos:

```
. /
| _ _ text2graph/
| _ _ _ _ _ ct_extractor.py
| _ _ _ _ _ data_graph.py
| _ _ _ _ _ html_text.py
| _ _ _ _ _ _init_.py
| _ _ _ _ _ tokenizer_text.py
| _ _ AUTHORS.md
| _ _ CrowdTangle_API.odt
| _ _ install_model.py
| _ _ LICENSE
| _ _ MANIFEST.in
| _ _ README.md
| _ _ setup.cfg
| _ _ setup.py
| _ _ Text2Graph.odt
```

4. Instalación:

Para instalar la librería y sus dependencias, se requiere descomprimir los archivos principales, ingresamos a la carpeta donde se encuentre *text2graph_master.zip*:

```
unzip text2graph_master.zip
```

o bien:

tar -xvzf text2graph_master.zip

Si se trabaja desde Windows, simplemente descomprimir el archivo proprorcionado. Se ingresa en la carpeta descomprimida y se digita el siguiente comando:

python3 setup.py install --user

Si utiliza Windows puede utilizar el siguiente comando:

python setup.py install --user

Esta libería requiere instalar el modelo correspondiente de spacy, el cual se instala mediante los siguientes comandos en la terminal:

python3 install_model.py #Para linux

o bien

python install_model.py #En Windows

Se recomienta tener instalado python 3.6 o superior, además esta librería no se ha probado todavía en MacOS, por lo que no se puede brindar por el momento soporte para este sistema operativo.

4.1 Dependencias:

Además de los archivos presentes en la carpeta principal del proyecto, también se instalarán las siguientes dependencias necesarias para el correcto funcionamiento de los módulos:

"networkx>=2.4",
"pandas>=0.24.1",
"pandas-datareader>=0.7.0",
"plotly>=3.4.1",
"scipy>=1.3.0",
"numpy>=1.16.4",
"pyreadr>=0.2.9",
"spacy>=2.3.2",
"tldextract>=2.2.3",
"nltk>=3.4.0",
"mpld3>=0.3.0",
"beautifulsoup4>=4.9.1",
"requests>=2.25.0",
"tldextract>=2.2.3",
"lxml>=4.3.1",
"google>=3.0.0"

También es posible utilizar los modulos en forma independiente dentro de otros proyectos copiando los archivos de la carpeta text_graph dentro del proyecto principal (requiere instalar las dependencias)

4.2 Diagrama básico del prototipo:

Text2graph cuenta con cuatro modulos principales o clases para realizar las funciones, separadas entre sí para permitir ser utilizadas como librería independiente cada uno de ellos, o en conjunto, o como parte de un proyecto mayor. Estas son:

- *ct_extractor:* un extractor de datos de CrowdTangle
- *html_text*: capturador de texto de una página web utilizando el enlace.
- *tokenizer_text:* crea la lista de tokens de un texto y genera las listas de source y target para el grafo.
- *data_graph:* que genera el grafo con la librería Networkx, y puede exportar a gephi.

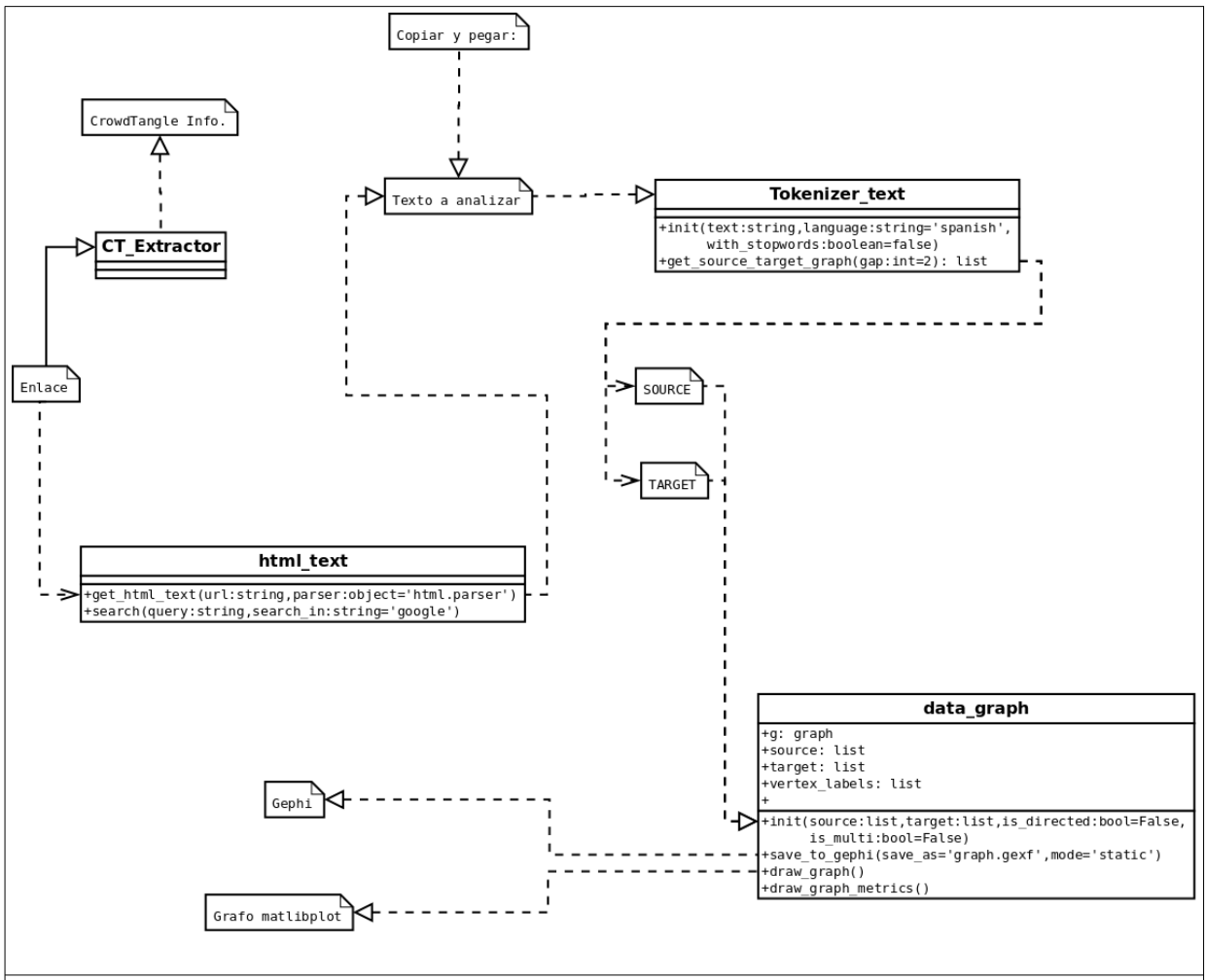

Figura 1. Diagrama parcial del analizador de texto: text2graph

5. Extractor CrowdTangle (ct_extractor):

Módulo que emula los comandos del API de CrowdTangle (Ver documento adjunto CrowdTangleAPI.odt), mediante el uso de clases y objetos que transforman las salidas *json* del API en datos de texto o csv según corresponda.

5.1. Objeto principal: get_cd_data(token)

get_cd_data(token)		
Parámetros:	token:	Crea un objeto de datos de CrowdTangle mediante el token dado

		por el API en el Dashboard principal.
Salida:	objeto:	Devuelve un objeto CrowdTangle, con el cual se pueden realizar diferentes consultas, devolviendo los objetos correspondientes: *ctpost*: por CrowdTangle post. *post*: por Facebook post (id_ de Facebook). *lists*: listas guardadas en el dashboard *list*: devuelve la información de una lista mediante su número de identificación. *links*: devuelve la información de CrowdTangle a través de un enlace o dirección, si existe una publicación asociada a Facebook a dicho enlace. *posts*: devuelve la información de los posts almacenados dentro del leaderboard. *leaderboard*: devuelve la información contendida en la tabla de clasificación principal del CrowdTangle. *search*: devuelve la búsqueda mediante parámetros de CrowdTangle, requiere permiso de Facebook para poder realizarse este tipo de consulta, por lo que no se ha programado por completo.

Ejemplo:

```
>>> import text2graph.ct_extractor as ce
>>> token = . . .
>>> ct_dt = ce.get_ct_data(token)
```

5.2 Objetos ctpost, post y links:

.ctpost(ctpost)		
Parámetros:	ctpost:	Número de identificación dado por CrowdTangle al post.
.post(fbpost, includeHistory=True)		
Parámetros:	fbpost	Número de identificación de facebook [usuario]_[post]
	includeHistory	Si desea incluir historial de interacciones (True o False)
.links(link, count=100, includeHistory=False, includeSummary=False, **kwargs)		
Parámetros:	link	Enlace externo o interno de la publicación
	count	Número máximo de enlaces relacionados a mostrar, si los encuentra.
	includeHistory	Si se desea que se incluya el historial (True o False)
	includeSummary	Si se desea que se incluya el resumen (True o False)
	kwargs	startdate, endDate, sortBy, (consultar CrowdTangleAPI.odt)
Funciones:	.raw_dict():	devuelve el diccionario del *json* con la data original.
	.status():	devuelve el estado de la búsqueda: 200 si ha encontrado

		todos los datos.
	.result():	devuelve el diccionario con los resultados.
	.platform_id():	devuelve la plataforma del post.
	.date():	devuelve la fecha de publicación.
	.notes():	devuelve ntoas si son incluidas en el post.
	.message():	devuelve el texto si es incluido en el post.
	.title():	devuelve el título si es incluido en el post.
	.ct_id():	devuelve el número de CrowdTangle.
	.link():	devuelve el link externo.
	.post_url():	devuelve el url dentro de Facebook.
	.domain():	devuelve el domino en facebook.
	.type():	devuelve el tipo de publicación (link, post)
	.media_type():	devuelve el tipo de media (video, foto, etc.)
	.media_url():	devuelve el url de media.
	.statisitcs():	devuelve el diccionario con las estadísticas de las interacciones.
	.statistics_df()	devuelve el dataframe con las estadísticas de las interacciones actuales y esperadas.
	.history():	devuelve el diccionario con el historial de las estadísticas.
	.history_df()	devuelve el dataframe con el historial de las estadísticas de las interacciones actuales y esperadas.
Solo en link():		
	.caption():	devuelve el subtítulo de la publicación.

Ejemplo:

```
>>> ctpost_numb = . . .
>>> ct_dt.ctpost(ctpost_numb).statistics_df()
. . .
>>> fbpost_numb = . . .
>>> ct_dt.post(fbpost_numb).history_df()
. . .
>>> link = ‘http://. . .’
>>> domain = ct_dt.links(link).domain()
>>> print(domain)
. . .
```

5.3 Objeto posts:

.posts(count=10, includeHistory=False, includeSummary=False, **kwargs)		
Parámetros:	count:	Número de post a mostrar.

	includeHistory	Si incluye historial, admite dos valores True o False.
	includeSummary	Incluye resumen de los datos, (True o False).
	kwargs:	startdate, endDate, language, sortBy, types (consultar CrowdTangleAPI.odt)
Funciones:	.raw_dict():	devuelve el diccionario del *json* con la data original.
	.status():	devuelve el estado de la búsqueda: 200 si ha encontrado todos los datos.
	.result():	devuelve el diccionario con los resultados.
	.get_df():	devuelve el dataframe con la lista de post resumiento los siguientes datos: *plataformId*: tipo de plataforma. *date*: fecha. *update*: fecha actualización. *type*: tipo de publicación, foto, video, etc. *title*: título de la publicación. *caption*: subtítulo. *decription*: decripción. *message*: mensaje o texto. *link*: enlace externo. *postUrl*: url de faceboo. *subscriberCount*: cantidad de subcriptores. *score*: puntaje en facebook.

Ejemplo:

```
>>> ctpost_numb = . . .
>>> ct_dt.posts().get_df()
. . .
```

5.4 Objeto lists:

.lists()		
Parámetros:	None:	
Funciones:	.raw_dict():	devuelve el diccionario del *json* con la data original.
	.status():	devuelve el estado de la búsqueda: 200 si fue exitosa.
	.result():	devuelve el raw_data de los resultados.
	.list_of_dict():	devuelve el diccionario con el resultado de búsqueda
	.list_df():	devuelve el dataframe con ID, Título y Tipo.
	.list_of_id():	devuelve la lista con los números de identificación de las listas.

Ejemplo:

```
>>> ct_dt.lists().list_df()
. . .
```

5.5 Objeto list:

.list(id_, count = 10, offset_options = 0)		
Parámetros:	id_:	Número de lista a mostrar.
	count	Número de renglones dentro de la lista a mostrar.
	offset_options	Opciones de impresión (Ver documento CrowdTangleAPI.odt).
Funciones:	.raw_dict():	devuelve el diccionario del *json* con la data original.
	.status():	devuelve el estado de la búsqueda.
	.result():	devuelve el diccionario de la búsqueda
	.accounts_df():	Devuelve el DataFrame con los siguiente atributos: *id*: identificación. *name*: nombre de la cuenta. *handle*: manipulación. *profileImage*: imagen de perfil. *suscriberCount*: número de subscriptores. *url*: url de la cuenta. *platformId*: plataforma. *accountType*: tipo de cuenta. *pageAdminTopCountry*: lugar de administración. *verified*: si ha sido verificado.

5.6 Objeto leaderboard:

.leaderboard(count=50, **kwargs)		
Parámetros:	count:	Número de post a mostrar.
	kwargs:	startdate, endDate, orderBy, sortBy, (consultar CrowdTangleAPI.odt)
Funciones:	.raw_dict():	devuelve el diccionario del *json* con la data original.
	.status():	devuelve el estado de la búsqueda: 200 si ha encontrado todos los datos.
	.result():	devuelve el diccionario con los resultados.
	.get_df():	devuelve el dataframe con la lista de post resumiento los siguientes datos: *account*: cuenta *summary*: resumen.

		subscriberData: datos del subcriptor. *id*: identificación. *name*: nombre de la cuenta. *handle*: manipulación. *suscriberCount*: número de subscriptores. *url*: url de la cuenta. *platformId*: plataforma. *pageAdminTopCountry*: lugar de administración. *verified*: si ha sido verificado. *likeCount*: total de likes *loveCount*: total de me encanta. *hahaCount*: total de me divierte. *wowCount*: total de me sorprende. *thankfulCount*: total de agradecido. *angryCount*: total de enojado. *sadCount*: total de me entristece. *shareCount*: total de compartido. *commentCount*: total de comentarios. *totalInteractionCount*: total de interacciones. *interactionRate*: tasa de interacciones.

Ejemplo:

```
>>> ct_dt.lists().leaderboard().get_df()
. . .
```

5.7 Objeto search:

.search(count=10, includeHistory=False, **kwargs)		
Parámetros:	count:	Número de post a mostrar.
	kwargs:	startdate, endDate, orderBy, sortBy, language, searchField, searchTerm (consultar CrowdTangleAPI.odt)
Funciones:	.raw_dict():	devuelve el diccionario del *json* con la data original.

No se ha programado más funciones debido a que se requiere permiso especial para utilizar la función search dentro del API de CrowdTangle.

6. Tokenizador de textos (tokenizer_text):

Es módulo toma un texto, que puede ser introducido mediante un link, consumido desde CrowdTangle o copiado de alguna fuente externa y realiza una normalización del texto, eliminando tildes, mayúsculas, además de lematizar todos los radicales para formar el concepto base, y con ella realizar la lista de aristas para alimentar el módulo de grafos.
En esta versión, se está trabajando con una clase que devuelve el objeto de texto, una lista tokenizada y normalizada, que mediante los parámetros proporcionados por el usuario genera una lista de vértices de un grafo tanto de salida (*sources*) como de llegada (*targets*) que son utilizados por el generador de grafos (*data_graph*).

6.1 tokenize_text

tokenize_text(text, language=’spanish’, with_stopwords=False)		
Parámetros:	text:	Texto a ser tokenizado, puede provenir de diferentes fuentes.
	language	Configura el lenguaje a utilizar, por default se utiliza el español (*spanish*), para alimentar el modelo.
	with_stopwords	Mantiene las palabras que no agregan significado semántico como artículos, pronombres personales, preposiciones, etc.
Funciones:	.get_source_target_graph(gap=2):	
	Decripción:	Devuelve la lista de source y target para alimentar el modelo de redes, dada el número de saltos correspondientes: gap = 0: p[0] → p[1], … , p[n-1] → p[n] gap = 1: p[0] → p[2], … , p[n-1-gap] → p[n]
	Parámetros:	Gap: o salto entre palabras
	Salida:	Source y target: listas con los nodos de salida y llegada respectivamente, cada renglón de las listas corresponde a una arista.

Ejemplo:

```
>>>import text2graph.tokenizer_text as tkt
>>>text = ”En un lugar de la Mancha, de cuyo nombre no quiero acordarme, no ha mucho tiempo que vivía un hidalgo de los de lanza en astillero, adarga antigua, rocín flaco y galgo corredor. Una olla de algo más vaca que carnero, salpicón las más noches, duelos y quebrantos los sábados, lantejas los viernes, algún palomino de añadidura los domingos, consumían las tres partes de su hacienda.”
>>>tokens = tkt.tokenize_text(text)
>>>gap = 2
>>>source, target = tokens.get_source_target_graph(gap=gap)
. . .
```

El resto de funciones del tokenizador de textos son utilizadas en los procesos internos y permiten limpiar el texto, filtrar *stopwords*, crear el radicalizador (*stemmer*), lematizar el conjunto de palabras.

7. Generador de grafos (data_graph):

Este módulo genera un objeto tipo grafo, cuando recibe por parte del usuario la listas de nodos de salida o *source* y la de llegada o *target*, aunque es un módulo independiente de los anteriores, pueden utilizarse en conjunto para generar redes de conceptos y palabras a partir de un texto de determinado. Se utiliza como base la librería networkx, y posee un módulo que permite exportar las redes en formato “.gexf” para ser leído por aplicaciones como Gephi.

7.1 Objeto data_graph

data_graph(source, target, is_directed=False, is_multi=False)		
Parámetros:	source	Lista con los nodos de salida.
	target	Lista con los nodos de llegada.
	is_directed	Si desea crear un grafo dirigido (True), o bien un grafo no es dirigido (False).
	is_multi	Si se desea crear un multigrafo (True), o bien un grafo simple (False).
Atributos:	.g	devuelve el grafo interno generado para ser trabajado por la librería networkx.
	.source	devuelve la lista de nodos de salida.
	.target	devuelve la lista de nodos de llegada.
	.vertex_labels	devuelve la lista con las etiquetas para los nodos.
Funciones:	.get_node_frequency_df(save=False, save_as='node_freq.csv', index=False)	
	Decripción:	Genera el dataframe con la frecuencia de los nodos tanto en source como en target. Se puede manipular mediante la librería pandas.
	Parámetros:	*save*: si desea que sea salvado a un archivo csv. *save_as*: Nombre para ser salvado por default se tiene "node_freq.csv" *index*: omite el índice en el dataframe guardado.
	Salida:	Source y target: listas con los nodos de salida y llegada respectivamente, cada renglón de las listas corresponde a una arista.
	.plot_node_frequency(tail_number=25, save=False, save_as='node_freq.png', html=False, ascending=True, html_name='node_freq.html', figw = 10, figh = 10, color='g')	
	Descripción:	Crea un gráfico de barras horizontales con la frecuencia de los nodos.
	Parámetros:	*tail_number*: número de elementos a mostrar. *save*: si desea que sea salvado (True), o no (False). *save_as*: nombre para guardar el archivo. *html*: si desea exportar a html (True), o no (False). *html_name*: nombre para guardar el html. *ascending*: de menor a mayor. *figw*: ancho de la figura. *figh*: altura de la figura. *color*: color de las barras.
	Salida:	imagen, html con el gráfico de barras horizontales de frecuencia.
	.self_loop_nodes_number()	
	Descripción:	Devuelve el número de nodos con conección a sí mismos.

	Parámetros:	
	Salida:	Entero.
	.adj_matrix_dataframe()	
	Descripción:	Devuelve el dataframe con la matriz de adyacencia, incluye las etiquetas de los nodos.
	Parámetros:	
	Salida:	Dataframe manipulable con librería pandas.
	.adj_matrix_csv(save_as='adjacency.csv', index=False)	
	Descripción:	Almacena en disco el dataframe de la matriz de adyacencia.
	Parámetros:	*save_as*: nombre para salvar la matriz de adyacencia. *index*: False si no quiere que aparezca el índice en el dataframe.
	Salida:	.csv
	.number_of_vertex()	
	Descripción:	Devuelve el número de vértices o nodos.
	Parámetros:	
	Salida:	Entero.
	.number_of_edge()	
	Descripción:	Devuelve el número de aristas.
	Parámetros:	
	Salida:	Entero.
	.shortest_distance_matrix()	
	Descripción:	Devuelve la matriz con la distancia más corta entre nodos.
	Parámetros:	
	Salida:	Array, manipulable con la libería numpy.
	.shortest_distance_dataframe()	
	Descripción:	Devuelve la matriz con la distancia en formato dataframe.
	Parámetros:	
	Salida:	Data frame, manipulable con la libería pandas.
	.metrics_df(metrics=’all’)	
	Descripción	Devuelve el dataframe con todas la métricas indicadas por el usuario.
	Parámetros	*metrics*: que puede ser ‘all’ y devuelve todas las métricas programadas, o bien una lista con los siguientes nombres: metrics = ['degree', 'eccentricity', 'pagerank', 'eigenvector', 'harmonic', 'closeness', 'betweenness']
	Salida:	Dataframe manipulable con librería pandas.

	.metrics_csv(save_as=’metrics.csv’, metrics=’all’, index=False)	
	Decripción	Devuelve el dataframe y lo guarda como .csv.
	Parámetros	*metrics*: que puede ser ‘all’ y devuelve todas las métricas programadas, o bien una lista con los siguientes nombres: metrics = ['degree', 'eccentricity', 'pagerank', 'eigenvector', 'harmonic', 'closeness', 'betweenness'] *save_as*: nombre para guardar el dataframe. *index*: ignora el índice del dataframe en el documento .csv.
	Salida	.csv
	.plot_node_metric(metric, tail_number=25, ascending=True, save=False, save_as='metric.png', html=False, html_name='metric.html', figw = 10, figh = 10, color='g')	
	Descripción	Muestra el gráfico de los nodos según la métrica elegida por el usuario.
	Parámetros	*metric:* métrica elegida por el usuario, entre 'degree', 'indegree', 'outdegree' 'eccentricity', 'pagerank', 'eigenvector', 'harmonic', 'closeness', 'betweenness' *tail_number*: número de elementos a mostrar. *save*: si desea que sea salvado (True), o no (False). *save_as*: nombre para guardar el archivo. *html*: si desea exportar a html (True), o no (False). *html_name*: nombre para guardar el html. *ascending*: de menor a mayor. *figw*: ancho de la figura. *figh*: altura de la figura. *color*: color de las barras.
	Salida	Figura en html o png, o mostrar en pantalla.
	.save_to_gephi(save_as='graph.gexf', description='', mode='static')	
	Descripción	Salva el grafo a formato .gexf para ser leído por gephi.
	Parámetros	*save_as*: nombre del archivo. *description*: descripción dada por el usuario. *mode*: ‘static’ o ‘dynamic’
	Salida	Archivo .gexf
	.set_nx_layout(layout='random', **kwargs)	
	Descripción	Configura el layout para mostrar el grafo.
	Parámetros	*layout*: que puede ser shell, spring, circular, kamada_kawai, bipartite, spectral, spiral, o planar. *kwargs*: argumentos específicos a cada layout, consultar documetación de Networkx.
	Salida	pos
	.draw_graph(g = None, html=False, save=False, html_name = 'graph1.html', save_as='graph1.png', **kwargs)	
	Descripción:	dibuja el grafo con la librería matlibplot, admite

		diferentes parámetros.
	Parámetros	*g*: grafo *html*: si desea salvar a html (True) o no (False) *save*: si desea salvar a png (True) o no (False) *html_name*: nombre para guardar html. *save_as*: nombre para guardar a png. ***kwargs*: otros parámetros
	Salida	grafo
	.draw_graph_metrics(html=False, save=False, html_name ='graph1.html', save_as='graph1.png', metrics='pagerank', size_factor = 10000, with_labels = False, with_node_label = True, with_values = False, first_n_values = 5, normal_node_size = True, cmap = 'YlOrRd', **kwargs):	
	Descripción:	Dibuja y colorea el grafo según una métrica específica
	Parámetros	*html*: si se salva a html, True, si no False. *save*: si se salva a png, True, si no False. *html_name*: nombre con que se guarda el html. *save_as*: nombre con que se guarda el png. *metrics*: tipo de métrica, 'pagerank', etc. *size_factor:* factor de multiplicación por la métrica *with_labels*: etiquetas en el grafo *with_node_label*: nodos con etiqueta *with_values*: nodos con valores de la métrica *first_n_values*: etiqueta para los primeros n valores. *normal_node_size*: tamaño normal del nodo (True) *cmap*: tipo de mapeo de color, consultar documentación de matlibplot.
	Salida	grafo
	.remove_node_by_label(label)	
	Descripción:	Remueve un nodo conociendo su etiqueta.
	Parámetros	*label*: la etiqueta del nodo a ser eliminado.
	Salida	
	.reset()	
	Descripción:	devuelve el grafo a su condición inicial.
	Parámetros	
	Salida	

	.add_label_attrib()	
	Descripción:	agrega el atributo de etiquetas a los nodos, cuando se trabaja con el exportador .gexf de networkx, las etiquetas deben pasar mediante atributos de nodos.
	Parámetros	
	Salida	
	set_node_attributes(attr_dict, name)	
	Descripción:	agrega un atributo a un nodo definido por el usuario, útil cuando se exporta el .gexf, ya interno o bien el de networkx.
	Parámetros	*attr_dict*: diccionario basado en la identificación de nodos dada por .graph().nodes o .g.nodes y el atributo correspondiente. *name*: nombre que desea que tenga el atributo.
	Salida	
	.write_gexf(save_as='g_graph.gexf')	
	Descripción	llamada al módulo de Networkx para exportar a .gexf, requiere que los nodos tengan propiedades para exportarlas, en especial las etiquetas.
	Parámetros	save_as: nombre con el que el archivo aparecerá.
	Salida	

Ejemplo:

```
>>> import text2graph.data_graph as tg
>>> text_graph = tg(source, target)
>>> text_graph.plot_node_frequency()
>>> text_graph.save_to_gephi()
>>> text_graph.set_nx_layout(layout='spring')
>>> text_graph.draw_graph_metrics(metrics='betwenness')
. . .
```

7.2 Funciones internas, no muestran resultados para el usuario:

.create_graph()	la función interna .create_graph() genera el grafo con el source y target brindados por el usuario, se almacena en la variable g.
.graph()	devuelve el grafo almacenado en g.
.labels()	devuelve el diccionario con las etiquetas de los diferentes nodos.
.get_adj_matrix()	devuelve la matriz de adjacencia de los nodos del grafo, no incluye etiquetas.
.empty_dic()	crea un diccionario vacío.
.indegree()	devuelve el diccionario con el *indegree* de los nodos, solo se calcula en grafos dirigidos.
.outdegree()	devuelve el diccionario con el *outdegree* de los nodos, solo se calcula en grafos dirigidos.

.eccentricity()	devuelve el diccionario con la *eccentricity* de los nodos, solo se calcula en grafos dirigidos, solo es posible calcularlo en grafos conectados.
.harmonic_centrality()	devuelve el diccionario con la centralidad armónica, si no es posible calcularla devuelve un diccionario vacío.
.harmonic_closeness()	devuelve el diccionario con la centralidad de cercanía, si no es posible calcularla devuelve un diccionario vacío.
.eigenvector_centrality()	devuelve el diccionario con la centralidad por eigenvector, con alpha 0.85 y un épsilon de 1e-3.
.betweenness()	devuelve el diccionario con la centralidad por intermediación normalizada.
.return_extension()	devuelve la extensión de un archivo
.default_layout()	configura el layout del grafo.
.graph_pos()	devuelve la configuración del layout.
.remove_node(node)	remueve el nodo según el número de nodo.
.list_nodes_attributes()	lista con los nombres de los atributos definidos por el usuario.
.get_dict_node_attrib()	diccionario con clave nombre de los atributos y el atributo correspondiente.

8. Capturador de texto html (html_text):

Este módulo, permite mediante un link o enlace capturar todo el texto del html para poder alimentar el tokenizador en forma automatizada.

8.1. get_html_text:

get_html_text(url, parser='html.parser')		
Parámetros:	url:	dirección de html del cual se va a extraer el texto de la noticia.
	parser:	configura el extractor de etiquetas html de *BeautifulSoup*, puede ser *html.parse*r, *lxml*, *xml* o *html5lib*, para mayor detalle consultar documentación de BeautifulSoup en: (https://www.crummy.com/software/BeautifulSoup/bs4/doc/)
Salida:	output:	Texto del html contenido entre los tags

Ejemplo:

```
>>> import text2graph.html_text as html
>>> text = html.get_html_text('https://en.wikipedia.org/wiki/Lorem_ipsum')
>>> print(text)
…
```

8.2. search:

search(query, search_in='google', timeout=None, **kwargs)		
Parámetros:	query:	Palabras claves de búsqueda.
	search_in:	Selecciona la máquina de búsqueda 'google' o 'yahoo'
	timeout:	Indica el tiempo de espera de la búsqueda.
Salida:	df:	Devuelve el DataFrame con los datos de la búsqueda, con título del html y el url de la búsqueda.

Ejemplo:

```
>>> import text2graph.html_text as html
>>> df = html.search('covid', search_in='google')
>>> print(df)
>>> text = html.get_html_text(df['link'][24])
…
```

9.Mejoras a realizar:

- En el futuro se puede programar el módulo de search del ct_extractor, para tener las funcionalidades completas en el módulo.
- El lematizador requiere de mejorar en algunos aspectos, posiblemente integrarse o bien utilizar los modelos de los proyectos paralelos realizados por el equipo de #FakeNews.
- El módulo de métricas de *data_graph*, puede recibir parámetros específicos para cada una de las métricas.
- El código del tokenizador de texto puede mejorarse, en especial en lo que se refiere al uso del español como lenguaje predeterminado. Es necesario encontrar una manera óptima para el manejo de las excepciones del idioma español. Existen problemas con la lematización correcta de los verbos irregulares y la clasifiación de ciertos sustantivos, la radicación no es suficiente.
- Falta por decidir se es mejor trabajar los grafos con radicales o seguir trabajando con lematización.
- Actualmente el exportador de .gexf lo hace a la version 1.2, para asegurar el máximo de compatibilidad con gephi, pero se espera que en versiones posteriores se exporte al actual estándar 1.3.
- En el capturador de texto de html, falta mejorar el filtrado del texto, puede capturar algunos textos como etiquetas de comentarios o similares que puede no estar en el cuerpo principal.

10.Historial de Versiones:

Cambios principales entre versiones:

Versión 0.0.2:

- Se independiza el generador de grafos (*data_graph*) del tokenizador (*tokenizer_text*).
- Se agrega extractor de texto de html (*html_text*).
- Se agrega un buscador de términos para las máquinas de búsqueda Google y Yahoo, devuelve los *url* y títulos de las búsquedas.

Documentación consultada:

Gephi.org. (2007). **Gexf file Format**. Documentación y formato de archivos gexf. Consultado en:

https://gephi.org/gexf/format/

Richarson, L. (2020) **BS4: BeautifulSoup**. Documentación. Consultado en: https://www.crummy.com/software/BeautifulSoup/bs4/doc/