ANÁLISIS DE REDES SOCIALES CON *PYTHON*

PROYECTO

Javier Becerra Elcinto - javier@auva.es
José Antonio Fontaneda Pérez - contacto@webmocionarte.com

ThinkTIC - *Mayo 2020*

AUVA FORMACIÓN IA

Este proyecto consta de dos partes:

1. Captura de datos:
   - Conexión a la API de Twitter para obtención de datos.
2. Análisis de datos:
   - Generación de un grafo a partir de los datos capturados.
   - Identificación de los 'intermediadores' de la red.
   - Identificación de los 'influencers' de la red.
   - Identificación de comunidades.

Si no disponéis de una *app*/permisos en *Twitter* para capturar los datos podéis utilizar el fichero adjunto *trump_graph.csv* como punto de partida y pasar directamente a la parte 2.

El ficheros trump_graph.csv contiene un grafo generado a partir de la información de extraída de 108228 tweets obtenidos con la búsqueda 'Trump'.

1. CAPTURA DE DATOS

En este apartado tendremos que:

1. Conectarnos a la API de Twitter para recopilar datos.
2. Procesar los datos y generar un fichero en formato *edgelist* de relaciones entre usuarios.

1. EJERCICIO (OPCIONAL)

CONEXIÓN A LA *API* DE TWITTER

Nota: si no has podido registrar una app con Twitter puedes pasar directamente a la parte 2 y utilizar el fichero trump_graph.csv como entrada.

1. Si no la tienes aún, da de alta tu cuenta de desarrollador en *Twitter* y da de alta tu primera *app* para conseguir las credenciales de acceso.

EL GRAFO

Queremos conseguir un grafo para poder analizar mediante *networx* e identificar *intermediadores* (nodos que conectan comunidades) e *influencers* (nodos que ejercen de atractores).
Tenemos varias opciones:

- Podemos utilizar para generar nuestra red la lista de seguidores (*followers*) y seguidos (*friends*) de la red de seguidores de un usuario en concreto (como hicimos para el apartado de *Análisis de Grafos*).
- También podemos utilizar las relaciones tipo 'quién retuitea a quién'. Es lo que vamos a hacer en este caso.

En este caso vamos a seleccionar un *topic* que nos interese y hacer la captura de datos utilizando el *endpoint* de búsqueda (`search`) de la api de Twitter. Eso tiene dos implicaciones fundamentales:

- La API nos proporciona hasta 18000 tweets cada 15 minutos (100 tweets para cada una de las 180 peticiones máximo permitidas), lo que nos permitirá recopilar una gran cantidad de datos en poco tiempo **(siempre y cuando nuestro *topic* haya generado suficiente interés durante la última semana)**
- El grafo que obtengamos de esta forma va a ser mucho menos **denso** que el que se obtiene por la lista de seguidores de un usuario, que tienen un vínculo en común relativamente más fuerte que haber 'tuiteado' sobre un tema (i.e. necesitaremos muchos datos, aunque luego haya que descartar nodos 'poco interesantes')

2. EJERCICIO (OPCIONAL)

1. Conéctate a la *api* de *Twitter*, (si utilizamos el parámetro `wait_on_rate_limit=True` al construir el objeto `tweepy.API` tweepy esperará de forma automática cuando se supere la cuota de peticiones a *Twitter*, pero será difícil parar el proceso cuando esté *en espera*).
2. Utiliza la api de búsqueda (`api.search` y descarga mensajes relacionados con el tema elegido. Uno con el tendremos *tweets* de sobra es usar como query 'Trump'.
   En la página siguiente tienes un ejemplo que gestiona automáticamente el que los tweets se reciban de 100 en 100.

En este ejemplo guardamos los `tweets` al completo, en el formato `pickle` de *Python*- Si lo preferís podéis utilizar una base de datos o almacenar los lotes en formato 'json' (en ese caso tendréis que crear una lista con el campo `status._json` de cada tweet). Guardamos cada lote de 100 *tweets* en un fichero *pickle* (faltaría los `import`, inicialización de la api, etc.):

```
for page_id, page in enumerate(tweepy.Cursor(
        api.search, q='Trump', count=100,
        tweet_mode='extended',
        languages = ['en', 'es']).pages()):
    # process status here
    fn = f'page_{page_id:06d}.pkl'
    with open(fn, 'wb') as f:
        pickle.dump(page, f)
```

Si dejamos nuestro código un rato capturaremos un buen número de *tweets* (para este ejercicio una cantidad como 100000 estaría bien, lo que supone esperar aproximadamente una hora y media respetando los límites de la API de *Twitter*).

Aunque pueden parecer muchos *tweets*, luego veremos que vamos a eliminar la mayoría de usuarios de nuestra red:

- Los que tengan pocas conexiones con el resto de la red.
- Las componentes aisladas del grafo (con estos datos es normal ver la aparición de varias redes aisladas, mantendremos únicamente la red más grande de contactos).

Dependiendo de tu equipo, es posible que tengas algún problema al manipular tanta información. En ese caso puedes reducir el número de tweets hasta que tu equipo responda.

3. EJERCICIO (OPCIONAL)

Ahora tenemos que cargar los datos y generar el grafo.
Volvemos a utilizar `pickle` como ejemplo, si habéis utilizado otro formato tendréis que adaptar el código.

Para obtener todas las relaciones de quién menciona y quién retwittea a quién tenemos que analizar información en varios campos, en el fichero `utils.py` tenéis funciones para facilitar esa tarea (sin ser el código ni más bonito ni más eficiente que vais a ver, dicho sea de paso...).

AUVA FORMACIÓN IA

1. Carga los datos y almacena únicamente los campos que hacen referencia a id's de usuario (user_id del

(Sigue...)

2. Almacena la lista de aristas (ahora está en `df_graph`) como un fichero csv (`df_graph.to_csv`) para poder cargarlo con `networkx.read_edgelist`:

- Sin cabecera (parámetro `header=False`)
- Sin el número de muestra (parámetro `index=False`)
- Usando un espacio en blanco como separador (parámetro `sep=' '`)

PARTE 2: ANALIZANDO LA RED

Ya tenemos nuestro red definida. Ya solo nos quedan tres tareas:

1. Limpieza de los datos
   - Eliminar nodos con pocas conexiones con el resto.
   - Eliminar subgrafos que no estén conectados con el grafo más grande.
2. Análisis de la red.
3. Análisis de los nodos.
4. Detección de comunidades.

1. EJERCICIO

1. Carga el fichero que has generado (`trump_graph.csv`) y y visualiza el grafo de la red utilizando la biblioteca `networkx`. Carga el grafo como un grafo **no dirigido** (`nx.Graph`)

> *Si la visualización tarda demasiado, puedes pasar al siguiente punto y visualizar la red después de simplificarla*

2. EJERCICIO

Vamos a simplificar un poco la red para que el análisis sea más rápido y efectivo.

1. Calcula el grado de cada nodo (número de conexiones y salientes) (método `G.degree`).
2. Filtra los nodos que tengan un grado inferior a 15 y genera un nuevo grafo sin esos nodos:

```
...
nodes = [k[0] for k in deg if k[1]>=deg_min]
G_simple = G.subgraph(nodes)
```

1. Compara el grado medio (`nx.info`) y la densidad (`nx.density`) de ambas redes.

COMPROBACIÓN IMPORTANTE

A partir de ahora utilizaremos el grafo `G_simple`, con muchos menos nodos que el original, para que los tiempos de proceso sean razonables.

*En un equipo portátil con procesador Intel **i7**, las operaciones más costosas (estimación de posición de los nodos para dibujar el grafo mediante `spring_layout`, o el cálculo de la centralidad de Katz) llevan de **10 a 20 segundos**. En equipos con menos prestaciones puede que tarde algún minuto, pero si véis que el cálculo se alarga en exceso **comprobad que estáis usando el grafo simplificado en los cálculos.***

Si aún utilizando el grafo simplificado tenéis problemas, recortadlo con un valor de `deg_min` superior, hasta que tengáis un conjunto de datos manejable.

3. EJERCICIO

Con esta fuente de datos es probable que tengamos varias componentes inconexas. Vamos a quedarnos con la componente más grande:

1. Utiliza la función `nx.connected_components` para recuperar las distintas componentes conexas que hay en nuestro grafo.
2. Utiliza el método `subgraph` del grafo para crear un nuevo grafo únicamente con la mayor componente conexa detectada.

4. EJERCICIO

Ahora vamos a analizar la centralidad de la red:

1. Calcula la centralidad de intermediación de la red y visualiza los nodos con mayor valor.
2. Calcula la centralidad de Katz (pagerank o la centralidad de valor propio debeíran dar resultados similares) y visualiza la posición de los nodos de mayor valor.

5. EJERCICIO (OPCIONAL)

1. Vuelve a conectarte a la API de *Twitter* y recupera la información de los ids con mayor importancia en la red (puedes fijarte en los 10 o 20 primeros).

6. EJERCICIO

Analiza las comunidades que aparecen en el grafo:

1. Utiliza el módulo `python-louvain` para ver qué comunidades aparecen en la red (recuerda que se importa como `community` una vez instalado).
2. Visualízalas.

7 EJERCICIO (OPCIONAL)

1. Recupera la lista de los usuarios con más importancia en cada comunidad (primero necesitas la lista de usuarios de cada comunidad, y luego ordenarla por su importancia).
2. Conéctate a la API de *Twitter* e identifica los usuarios que aparecen. ¿Se pueden identificar las comunidades que se han generado?
3. ¿Qué criterio de centralidad utilizarás para identificar los usuarios más representativos? ¿Qué usuarios aparecen con los mayores valores de centralidad de intermediación?

Y si has llegado hasta aquí...

GRACIAS

Espero que hayas disfrutado del curso y te haya resultado provechoso.

Y sin duda espero que volvamos a vernos pronto (mejor en persona que en remoto, pero nos conformaremos con lo que haga falta...)

AUVA FORMACIÓN IA