RELAZIONE DEL 1° PROGETTO PER IL CORSO DI SOCIAL COMPUTING A.A 2020/21

Andrea Antonutti	**Michele Marchi**	**Marco Venir**	**Francesco Zigotti**
Mat. 142792	Mat. 128526	Mat. 144762	Mat. 125068

Terminologia usata:

- Utenti principali: *@mizzaro, @damiano10, @Miccighel_, @eglu81 e @KevinRoitero*
- Utenti correlati: utenti con i quali è presente una relazione di follower o following con qualsiasi altro utente.
- Grafo parziale: grafo che presenta relazioni solamente tra gli utenti principali e i nodi della rete.
- Grafo totale: grafo che presenta relazioni tra tutti gli utenti della rete.

Scaricamento dei dati

Lo scaricamento e la generazione dei dati sono gestiti attraverso due classi: **`TwitterUser`** e **`TwitterUserNetwork.`**

`TwitterUser` salva ed elabora i dati di ogni singolo utente. La generazione dell'istanza avviene passando come parametro lo **`screen_name`** dell'utente desiderato. Al momento dell'inizializzazione, mediante l'utilizzo di **`api.get_user`** di Tweepy, tutti i dati dell'utente vengono scaricati dal database di Twitter e vengono inseriti nell'attributo **`data`** dell'istanza. Per semplificare l'operazione di recupero dei follower e following di ogni utente, sono presenti rispettivamente i metodi **`TwitterUser.get_followers()`** e **`TwitterUser.get_following()`**, che, tramite l'oggetto Cursor messo a disposizione dalla libreria Tweepy, recuperano i dati di un numero arbitrario di utenti. Per ognuno degli utenti trovati viene creata un'istanza **`TwitterUser`** la quale viene poi inserita in un'apposita lista (**`followers_list`** / **`following_list`**), attributo della superclasse. Recuperando gli utenti delle suddette liste è possibile iterare per scaricare i dati dei followers dei followers ed dei following dei following.

Dopo aver recuperato i dati degli utenti principali e scaricato tutti i loro followers e following con i metodi sopra citati, abbiamo proceduto selezionato casualmente da ciascuno degli utenti principali 5 followers e 5 following e per ognuno di essi abbiamo scaricato i dati di ulteriori 10 followers e 10 following casuali. In totale abbiamo quindi recuperato i dati di 3084 utenti differenti.

Successivamente abbiamo serializzato tutti i dati scaricati in un file JSON.

Generazione grafo

La classe **`TwitterUserNetwork`** ha il compito di recuperare le relazioni tra utenti correlati e generare il grafo della rete sociale.

Ogni utente può essere inserito in una **`TwitterUserNetwork`** tramite il metodo **`TwitterUserNetwork.addUser()`** passando come parametro l'istanza di un oggetto **`TwitterUser`**.

La funzione **`TwitterUserNetwork.generate_partial_social_graph()`** provvede ad istanziare un nuovo grafo, a generare un nodo univoco per ciascuno utente ed a verificare, tramite **`api.show_relationship()`**, se esiste una relazione di follows tra i nodi degli utenti principali e il resto dei nodi della rete creando il relativo arco.

Nel nostro caso abbiamo inserito le istanze di ognuno degli utenti principali in un oggetto **`TwitterUserNetwork`** e salvato su disco il grafo originato tramite la funzione **`Networkx.save_gpickle()`**.

Del grafo abbiamo poi creato una versione interattiva tramite le funzioni fornite dalla libreria Pyvis che genera un file HTML che permette la visualizzazione e l'esplorazione della rete sociale via browser.

Ottimizzazioni

Facendo qualche calcolo, abbiamo stabilito che seguendo le istruzioni per lo scaricamento dei dati descritte nella traccia, ricordando che il recupero delle relazioni è limitato a 180 richieste ogni 15 minuti (720 ogni ora), si nota che, usando una singola macchina, il tempo impiegato è:

Scaricamento dei dati:	3084 utenti / 300 richieste/h ≈ 10 ore
Recupero delle relazioni:	3084 utenti * 5 utenti principali = 15420 richieste
	15420 / 720 ≈ 21 ore
Totale:	10 + 21 = 31 ore

Per ridurre i tempi abbiamo sviluppato una versione ottimizzata di **`TwitterUserNetwork`** e **`TwitterUser`** che utilizzano delle funzioni alternative di Tweepy.

Abbiamo sostituito il **`Cursor`** presente nelle funzioni **`TwitterUser.get_followers()`** e **`TwitterUser.get_following()`**, estremamente limitato in termini di richieste effettuabili, con la funzione **`api.lookup_users()`**. Quest'ultima richiede un array di **`user_id`** o **`screen_name`** come argomento ed è in grado di restituire i dati degli utenti cercati, in gruppi di 100 a richiesta, con un limite di 900 richieste ogni 15 min.

Gli **`user_id`** degli utenti correlati sono stati recuperati mediante le funzioni **`api.followers_id()`** e **`api.following_id()`** e vengono di volta in volta salvati come attributi dell'istanza **`TwitterUser`**.

Queste ultime hanno la limitazione di 15 richieste ogni 15 minuti ciascuna ma vengono richiamate solo una volta per ogni utente di cui si vuole recuperare i relativi utenti correlati. Nel nostro caso abbiamo 5 utenti principali da cui recuperare i followers e following, 25 utenti correlati dai quali recuperare solamente followers e 25 utenti dai quali recuperare solamente following. Un ammontare di 30 richieste a **`api.followers_id()`** e 30 richieste a **`api.following_id()`**, svolte in parellelo, per un totale di circa 15 minuti di recupero dei dati.

Il metodo **`TwitterUserNetwork.generate_partial_social_graph()`**, che svolge la funzione di generare il grafo parziale, è stato ottimizzato in modo da sfruttare gli array **`followers_id`** e **`following_id`** appena introdotti per verificare se per ogni utente principale è presente ciascuno degli altri nodi all'interno di essi. Nei casi affermativi viene generato il relativo arco "follows".

Se gli utenti principali hanno già salvato gli id dei loro affiliati (questo avviene sempre nei casi in cui siano stati recuperati dei followers o following da questi ultimi) tutto ciò viene svolto unicamente in tempo di elaborazione della CPU che, anche nei grafi di modesta grandezza, su un discreto hardware, risulta irrisorio.

Se invece si desidera generare il grafo completo, il metodo **`TwitterUserNetwork.generate_complete_social_graph()`** recupera automaticamente tutti i **`followers_id`** e **`following_id`** di tutti gli utenti della rete e per ognuno di essi verificherà la presenza o meno degli id di ciascuno degli altri utenti, generando gli archi necessari.

Apportando queste modifiche le 3084 * 3083 = 9.507.972 richieste a **`api.show_relationships()`**, effettuate dalla classe non ottimizzata in un anno e mezzo, vengono convertite in altrettante computazioni della CPU che, con l'hardware a nostra disposizione, sono state svolte tranquillamente in pochi minuti.

In sintesi, con le ottimizzazioni apportate siamo riusciti a ridurre drasticamente i tempi di costruzione del grafo parziale (da 31 ore a circa 15 minuti, più del 99% di efficienza) e generare un grafo completo con le relazioni tra tutti i nodi della rete in un tempo di circa 51 ore di scaricamento dati e pochi minuti di computazione.

Misure del grafo e risultati

Le seguenti misure sono state rilevate sul grafo parziale.

Come prima cosa si è cercato di verificare se il grafo fosse connesso e bipartito ottenendo due risultati negativi. Inoltre, si sono calcolate le distanze del grafo tramite le opportune funzioni di NetworkX. Essendo poi necessario un grafo connesso per calcolare diverse misure abbiamo usato la funzione **`NetworkX.isolates`** per individuare i nodi non connessi e rimuoverli.

In particolare abbiamo calcolato:

Centro:	"KevinRoitero", "eglu81", "damiano10"
Diametro:	4
Raggio:	2

Come si poteva prevedere, i nodi rilevati tramite la funzione **`NetworkX.center`** sono tra gli utenti principali.

Il diametro ci suggerisce che siamo in un contesto "small world", rientrando quindi nella euristica dei 6 gradi di separazione della connessione della rete.

Per verificare ulteriormente la questione, dal calcolo dei coefficienti di "small-world-ness" omega e sigma, abbiamo ottenuto i seguenti risultati:

Omega:	0.0020
Sigma:	0.9832

Questi risultati sono un'ulteriore conferma che il nostro grafo è aderente al modello "small world".

Per ottenere il coefficiente omega in tempi ragionevoli abbiamo diminuito il parametro **`niter`** della funzione **`smallworld.omega`** da 100 a 1 iterazione.

Successivamente ricaviamo le misure di centralità del grafo delle quali riportiamo in seguito per ognuna le tre più significative:

Betweenness: “damiano10” : 0.2035, “eglu81”: 0.1437, “Miccighel_” : 0.0883

Closeness: “damiano10” : 0.3554, “eglu81”: 0.3237, “cikm2020” : 0.2667

Degree: “damiano10” : 0.5284, “eglu81”: 0.3763, “Miccighel_” : 0.1586

In-Degree: “damiano10” : 0.2559, “eglu81”: 0.1745, “mizzaro” : 0.1077

Out-Degree: “damiano10” : 0.2724, “eglu81”: 0.2017, “mizzaro” : 0.1074

PageRank: “damiano10” : 0.1275, “eglu81”: 0.0846, “Micchighel_” : 0.0568

HUB: “damiano10” : 0.0725, “eglu81”: 0.0392, “mizzaro” : 0.0270

Authority: “damiano10” : 0.0067, “eglu81”: 0.0043, “mizzaro” : 0.0024

Rileviamo che “damiano10” è il nodo più centrale del grafo, seguito dai restanti utenti principali.

In seguito abbiamo generato il sottografo indotto dal nodo “damiano10” per calcolare la cricca massima e la sua dimensione:

Cricca massima: {“eglu81”, “essir2017”, “mizzaro”}

Dimensione: 3

È interessante osservare come i due nodi “cikm2020” e “essir2017”, relativi ad eventi sulla ricerca e gestione di informazioni, siano usciti come primi risultati rispettivamente di closeness e di cricca massima. Dopo aver effettuato il loro sottografo abbiamo notato che quasi tutti gli utenti principali hanno relazioni con questi due nodi, possiamo quindi dedurre che essi hanno questo interesse in comune.

Infine abbiamo calcolato le misure di correlazione di Pearson e Kendall e tramite OpenPyXL abbiamo creato direttamente dal notebook una tabella Excel con i seguenti risultati:

Pearson								
	BETWENNESS CENTRALITY	CLOSENESS CENTRALITY	DEGREE	IN DEGREE	OUT DEGREE	PAGERANK	HITS HUB	HITS AUT
BETWENNESS CENTRALITY	1,00	0,07	0,99	1,00	0,98	1,00	0,91	0,34
CLOSENESS CENTRALITY	0,07	1,00	0,08	0,10	0,06	0,09	0,05	0,77
DEGREE	0,99	0,08	1,00	0,99	0,99	0,99	0,95	0,36
IN DEGREE	1,00	0,10	0,99	1,00	0,97	1,00	0,91	0,37
OUT DEGREE	0,98	0,06	0,99	0,97	1,00	0,96	0,96	0,35
PAGERANK	1,00	0,09	0,99	1,00	0,96	1,00	0,90	0,36
HITS HUB	0,91	0,05	0,95	0,91	0,96	0,90	1,00	0,37
HITS AUT	0,34	0,77	0,36	0,37	0,35	0,36	0,37	1,00

Kendall								
	BETWENNESS CENTRALITY	CLOSENESS CENTRALITY	DEGREE	IN DEGREE	OUT DEGREE	PAGERANK	HITS HUB	HITS AUT
BETWENNESS CENTRALITY	1,00	0,13	0,17	0,17	0,16	0,14	0,12	0,13
CLOSENESS CENTRALITY	0,13	1,00	0,53	0,84	-0,17	0,83	-0,11	0,96
DEGREE	0,17	0,53	1,00	0,66	0,52	0,58	0,47	0,59
IN DEGREE	0,17	0,84	0,66	1,00	-0,16	0,86	-0,13	0,86
OUT DEGREE	0,16	-0,17	0,52	-0,16	1,00	-0,13	0,87	-0,13
PAGERANK	0,14	0,83	0,58	0,86	-0,13	1,00	-0,10	0,82
HITS HUB	0,12	-0,11	0,47	-0,13	0,87	-0,10	1,00	-0,07
HITS AUT	0,13	0,96	0,59	0,86	-0,13	0,82	-0,07	1,00

I risultati che riteniamo significativi dalle correlazioni dalle due tabelle sono:

- *PageRank* e *in-degree*: poiché il PageRank indica la qualità di un nodo in base alle pagine dai cui è riferito e l'in-degree è il puro conteggio degli archi in entrata di un nodo, riteniamo che questa correlazione suggerisce che la qualità della pagina è legata dal solo numero degli archi entranti.
- *HITS Hub* e *out-degree*: Usando una discussione simile a quella precedente, possiamo dire che l'hub non sia distinguibile dall'out-degree. Si nota che nel nostro grafico i nodi con più utenti seguiti, ovvero i 5 utenti principali, siano anche gli hub migliori in quanto tendono a seguire le migliori authority, coincidenti sempre con gli utenti principali. In generale questo conferma che questi ultimi sono tra loro molto legati e molto influenti nella rete.
- *Closeness* e *HITS authority*: sempre data la centralità degli utenti principali questa correlazione conferma che essi sono le migliori authority e raggiungono più facilmente gli altri nodi. Riprendendo l'euristica discussa a lezione "il mio amico ha più amici di me".

Come ultima misura, abbiamo voluto verificare la reciprocità delle relazioni dei nodi attraverso il comando **`reciprocity`** di NetworkX ottenendo un valore del 40% per il grafo parziale e del 34% sul grafo totale, da cui deduciamo che, nonostante Twitter non abbia relazioni adirezionali di base come sarebbero gli amici su Facebook, una buona fetta degli utenti tende a ricambiare il follow.