Informatica - Area scientifica
Dipartimento di Scienze matematiche, informatiche e multimediali
Università di Udine

Progetto di Social Computing

Parata Loris (144338)
Arzon Francesco (142439)
Dal Fabbro Lorenzo (142300)
Galvan Matteo (142985)

Anno accademico 2020/2021

Indice

Capitolo 1

Sottografo di Twitter

1.1 Introduzione

Questo primo progetto di Social Computing consiste nello studio della rete sociale di 5 utenti di Twitter.
Lo studio è stato svolto mediante la costruzione di un sottografo di Twitter, costituto da cinque nodi che rappresentano gli utenti principali e della loro relativa rete di contatti costituita dai loro follower, following e da rispettivi sottoinsiemi campionati in maniera random. Nello specifico abbiamo analizzato la relazione diretta di **follows** tra tutti i nodi del grafo ed i cinque profili scelti.

1.2 Costruzione del grafo

1.2.1 Download dei nodi

Il primo passo consiste nello scaricare tutti i followers attraverso la **api.followers()** di Twitter dei cinque nodi principali:

- @Mizzaro
- @damiano10
- @Miccighel_
- @eglu81
- @KevinRoitero

```
followers_ids,following_ids={},{}

for utente in users_id:
    print(utente)
    followers_utente,following_utente=[],[]
    for item in tweepy.Cursor(
        api.followers,
        id=utente,
        skip_status=True,
        include_user_entities=False
    ).items():
        json_data=item._json
        user={"id":json_data["id"]}
        time.sleep(181)
        followers_utente.append(user)

    followers_ids[utente]=followers_utente
```

L'uso della funzione **Cursor()** ci permette di restituire i dati richiesti ad una certa API in pagine. I parametri di questa funzione sono:

-**api.followers**: l'api desiderata, seguita dai parametri necessari ad essa.
-**id**: utente, indica l'id dello user "target" a cui siamo interessati
-**skip_status= True**: in modo che gli stati non verranno inclusi negli oggetti utente restituiti.
-**include_user_entities= False**: in modo da non includere l'oggetto utente "entities"
.items() ci permette di indicare l'eventuale la grandezza desiderata del blocco di dati richiesto.
Lo stesso procedimento è stato effettuato per i rispettivi following di ogni nodo principale, utilizzando la **api.friends()**.

```
#Download dei followers dei 10 followers degli utenti random
followers_ids= []
followers_ids = read_json("data_ids/followers_5_utenti.json")
random_followers_utenti_ids, followers_of_followers_ids = {}, {}

for utente in followers_ids:
    random_followers_utenti_ids[utente] = random.sample(followers_ids[utente], 5)
try:
    for utente in random_followers_utenti_ids:
        for f in random_followers_utenti_ids[utente]:
            fof = []
            print("scarico per " + str(utente))
            for item in tweepy.Cursor(
                    api.followers,
                    id=f['id'],
                    skip_status=True,
                    include_user_entities=False
            ).items(10):
                    time.sleep(10)
                    json_data = item._json
                    user = {"id": json_data["id"]}
                    fof.append(user)
                    print("Downloaded: " + str(user))

            followers_of_followers_ids[f['id']] = fof
except tweepy.TweepError as error:
    print(error)
```

Successivamente abbiamo selezionato gli **id** di 5 followers e 5 following randomicamente per ognuno dei 5 account. Da ognuno di essi sono stati scelti e scaricati gli id di altri 10 account followers e 10 account following sempre in maniera casuale.

Infine, una volta ottenuti tutti gli account, abbiamo scaricato tutte le informazioni principali relative agli account mediante la **api.get user()**, che ha come parametro l'**id** di un account precedentemente individuato.

```
nodes = {}

for user_id in id_nodi_grafo:
    #richiamo API GET_USER
    utente=api.get_user(id=user_id['id'])._json
    node_infos= {}

    node_infos["name"]=utente["name"]
    node_infos["screen_name"]=utente["screen_name"]
    node_infos["location"]=utente["location"]
    node_infos["followers_count"]=utente["followers_count"]
    node_infos["friends_count"]=utente["friends_count"]
    node_infos["statuses_count"]=utente["statuses_count"]
    node_infos["created_at"]=utente["created_at"]

    nodes[user_id['id']] = node_infos
```

Per un totale di **3103 nodi**.

1.2.2 Creazione degli archi

Successivamente abbiamo controllato l'esistenza di una relazione tra tutti gli account scaricati ed i 5 nodi principali con la funzione **api.show friendship()**. Aggiungendo gli archi raffiguranti l'azione di follows un file json, indicando nodo sorgente, tipo di relazione e nodo target. Il try-catch è stato utilizzato per rilevare eventuali nodi problematici.

```
friendships=[]
for main_user in user_id:
    for node_id in id_nodi_grafo:
        print(contatore/len(id_nodi_grafo))
        time.sleep(1)
        try:
            relationship = api.show_friendship(source_id=node_id['id'],target_id=main_user)
            relation = relationship[0]
            #Controllo nell'oggetto relationship delle relazioni di following tra nodo e nodo principale
            if(relation.following == True): # node follows the user
                infos_of_relation={}
                infos_of_relation["source"] = node_id['id']
                infos_of_relation["type"] = "follows"
                infos_of_relation["target"] = main_user
                friendships.append(infos_of_relation)
        except:
            print(node_id['id'])
```

Questa operazione, essendo molto costosa temporalmente, è stata effettuata modificando il codice in maniera tale da poterlo eseguire, con chiavi di autenticazione developer diverse, per ogni singolo utente. In modo da parallelizzare l'operazione, che costava circa 5 ore di tempo per singolo utente, a causa dei limiti temporali di 180 richieste ogni 15 minuti di all'api Twitter. Tutti gli archi sono stati uniti in un unico file json. Abbiamo ottenuto un totale di **1922 archi**.

Ottimizzazione archi

E' possibile rilevare tutti i nodi direttamente connessi ai 5 account andando a visualizzare direttamente i rispettivi followers, con la **api.followers()**, riducendo significativamente i costi in termine di richieste all'API. Ma per attinenza alla traccia abbiamo fatto un controllo completo per ogni nodo scaricato precedentemente.

1.2.3 Creazione del grafo

La costruzione del grafo è stata effettuata mediante l'utilizzo delle funzioni messe a disposizione di networkx.
Abbiamo inserito i nodi al grafo mediante la funzione **add_node()**. Indicando come parametri:
-ids: identificatore del nodo
-id di visualizzazione
-title: il titolo visualizzato dal nodo
-colore: colore del nodo
-phisics: per abilitare o meno l'influenza del nodo sulla fisica del grafo
-i restanti attributi del nodo.

1.2.4 Visualizzazione del grafo

La visualizzazione interattiva del grafo costruito con le funzioni messe a disposizione di networkX avviene utilizzando la libreria apposita pyvis.

Ottimizzazione visualizzazione

E' possibile ridurre i costi per l'elaborazione grafica di costruzione del grafo impostando il parametro opzionale phisic = False. Questo parametro a discapito dell'interazione fisica nel trascinamento dei nodi che avrebbero una risposta fisica, permette di risparmiare l'80 percento del tempo di rendering.

1.3 Analisi del grafo completo

Applicando le relative funzioni messe a disposizione dalla libreria di networkX abbiamo potuto stabilire che il grafo è:
Il grafo da noi analizzato è risultato **non connesso**.
Questo sottolinea che è errato dar per scontato che tutti gli utenti che seguono un determinato account **UtenteTwitter** a loro volta sono seguiti da utenti che seguono anche loro l' **UtenteTwitter**.

Nel caso in cui tenessimo traccia delle relazioni interne tra i nodi di secondo livello e quelli di terzo livello, considerando i path indiretti, allora sarebbe risultato connesso.
Ma questo dipende dalla componente casuale che sceglie da quali nodi scaricare i relativi follower dei follower.
Ecco perchè abbiamo deciso di procedere alla rimozione dei nodi sconnessi del grafo principale e di confrontare le proprietà dei due grafi dove è possibile.

1.3.1 Rimozione dei nodi sconnessi

```
#Creo una copia del grafo
sub_twitter_graph =twitter_graph.copy()

nodes_to_delete=[]
#Controlla se un nodo ha out_degree = 0,
#perchè ci interessano solo i nodi che seguono un account principale
for node_id in twitter_graph.nodes():
    if(twitter_graph.out_degree[node_id] == 0):
        nodes_to_delete.append(node_id)
sub_twitter_graph.remove_nodes_from(nodes_to_delete)
```

Questo codice ci permette di rimuovere dal grafo tutti i nodi che hanno **out-degree = 0**, cioè i nodi che non hanno archi uscenti.E nel nostro caso di ricerca dei follower,questa condizione basta per escludere i nodi sconnessi.
Otteniamo un grafo con soli **1679 nodi**.

1.3.2 Proprietà dei grafi

Il grafo completo risulta **non connesso**, mentre il suo sottografo è **connesso**.
Ma entrambi risultano **non bipartiti**.
Centro, Diametro e Raggio sono calcolabili solamente per il sottografo perchè il grafo completo essendo sconnesso ha valore di diametro e raggio infinito per definizione.
Per il sottografo abbiamo :

- Centro: i nodi di **Kevin Roitero**, **Gianluca Demartino** e **Damiano Spina**,
  tre dei 5 nodi principali.
- Diametro: **4**
- Raggio: **2**

1.3.3 Misure di centralità

Per le misure di Betweenness, Closness, Degree e In-centrality, otteniamo valori massimi riferiti al nodo di damiano10, seguito da eglu81,Miccighel e dai restanti nodi principali.
Per quanto riguarda l'out-centrality abbiamo come valore massimo un nodo che segue tre profili principali, per poi andare a decrescere per chi ne segue due e così via.

- Betweenness: valore massimo: 0.000375
  sottografo 0.0012
- Closness: valore massimo 0.353
  sottografo: 0.653

- Degree: valore massimo 0.254
  sottografo 0.653
- In-centrality: 0.253
  sottografo:0.469
- Out-centrality: 0.00128 di Kevin Callegher
  sottografo: 0.00238

Riportiamo anche i valori relativi al sottografo connesso che hanno un andamento coerente con il grafo grafo principale, ma i valori sono più alti perché abbiamo un numero di nodi complessivo minore.

- PageRank: grafo completo :0,216
  sottografo: 0,243
  per entrambi i valori si presentano in ordine al numero di followers che ogni account possiede.
- Hits:
  - Hubs: l'hub principale è Luke Gallagher, con valore 0.0015, che segue 3 profili su 5.
  - Authorities: la maggiore autority è Damiano con valore di 0.624, seguito dai restanti nodi principali.

Dai risultati notiamo come l'algoritmo di **PageRank** sia influenzato dal numero di nodi complessivi rispetto a **HITS**. Inoltre,conferma come le autorità siano i nodi con più archi entranti, in questo caso i 5 nodi principali, mentre gli hubs siano i nodi che seguono, in questo caso, più nodi principali(Luke Gallagher segue 3 nodi su 5).

1.3.4 Albero di copertura minimo

Gli archi tra i nodi che compongo l'albero di copertura minimo è possibile visualizzarli all'interno file *SC_progetto.ipynb* .

1.4 Analisi del sottografo dell'utente KevinRoitero

Considerando il sottografo dell'account KevinRoitero:
Lo estraiamo utilizzando la funzione **nx.ego_graph(sub_twitter_graph.reverse(), "3036907250")** sul sottografo,ma considerando la sua versione reverse, essendo che siamo interessati ai nodi che seguono Kevin (id: 3036907250). Vista la complessità computazionale dell'operazione di maxClique, l'abbiamo applicata su uno dei sottografi più piccoli del grafo principale.

1.5 Smallworldness

Considerando il sottografo non direzionato il valore dei parametri:

- Omega: 0.00683
- Sigma: 0.98653

Il valore di omega vicino allo 0, indica che il grafo ha le caratteristiche di una rete piccolo mondo. I parametri sono stati calcolati con:
- niter = 10, Numero approssimativo di ridirezioni per arco, per calcolare il grafico casuale equivalente
- nrand = 2, Numero di grafici casuali generati per calcolare il coefficiente di clustering medio e la lunghezza del percorso medio più breve.
Abbiamo optato per valori più bassi rispetto a quelli di default (100,10), questo per ridurre la complessità computazionale del calcolo dei parametri.

1.6 Analisi delle correlazioni di Pearson e Kendall

0	btw_centrality	cls_centrality	(0.8733993223251802, 0.0)	(1.0, 0.0)
1	btw_centrality	dg_centrality	(0.9910425083190649, 0.0)	(0.07646960324418119, 1.2850857755873085e-05)
2	btw_centrality	in_centrality	(0.9916571758957484, 0.0)	(1.0, 0.0)
3	btw_centrality	out_centrality	(0.12736453492501887, 1.071886184584708e-12)	(0.07449925773728967, 2.137232727948775e-05)
4	cls_centrality	btw_centrality	(0.8733993223251802, 0.0)	(1.0, 0.0)
5	cls_centrality	dg_centrality	(0.9041405908224469, 0.0)	(0.07646960324418119, 1.2850857755873085e-05)
6	cls_centrality	in_centrality	(0.9033962532946789, 0.0)	(1.0, 0.0)
7	cls_centrality	out_centrality	(0.15199990321507398, 1.6921549195351964e-17)	(0.07449925773728967, 2.137232727948775e-05)
8	dg_centrality	btw_centrality	(0.9910425083190649, 0.0)	(0.07646960324418119, 1.2850857755873085e-05)
9	dg_centrality	cls_centrality	(0.9041405908224469, 0.0)	(0.07646960324418119, 1.2850857755873085e-05)
10	dg_centrality	in_centrality	(0.9993531356368165, 0.0)	(0.07646960324418119, 1.2850857755873085e-05)
11	dg_centrality	out_centrality	(0.16327635144167213, 5.53423015238216e-20)	(0.9998902123524452, 0.0)
12	in_centrality	btw_centrality	(0.9916571758957484, 0.0)	(1.0, 0.0)
13	in_centrality	cls_centrality	(0.9033962532946789, 0.0)	(1.0, 0.0)
14	in_centrality	dg_centrality	(0.9993531356368165, 0.0)	(0.07646960324418119, 1.2850857755873085e-05)
15	in_centrality	out_centrality	(0.12769071521714853, 9.37845594949372e-13)	(0.07449925773728967, 2.137232727948775e-05)
16	out_centrality	btw_centrality	(0.12736453492501887, 1.071886184584708e-12)	(0.07449925773728967, 2.137232727948775e-05)
17	out_centrality	cls_centrality	(0.15199990321507398, 1.6921549195351964e-17)	(0.07449925773728967, 2.137232727948775e-05)
18	out_centrality	dg_centrality	(0.16327635144167213, 5.53423015238216e-20)	(0.9998902123524452, 0.0)
19	out_centrality	in_centrality	(0.12769071521714853, 9.37845594949372e-13)	(0.07449925773728967, 2.137232727948775e-05)

Figura 1.1:

1.7 Conclusioni

Dopo quest'analisi possiamo affermare che il grafo sociale costruito evidenzia la centralità dei nodi principali nel grafo, ma possiamo anche notare la presenza di nodi completamente scollegati da esso. I risultati ottenuti dalle funzioni ci permette di affermare che il sottografo connesso si tratta di una rete **"Piccolo Mondo"**.