Relazione primo progetto di Social Computing

Andrea Gritti (152206) Anna Sacchet (153005) Giovanni Palma (152336)
Paola Sagliocca (151861)

Introduzione

In questa prima esperienza di laboratorio, il nostro gruppo ha avuto l'occasione di prendere dimestichezza con il linguaggio di programmazione Python e di accedere alle API di Twitter, oltre che di effettuare analisi su grafi e reti grazie alle librerie: Tweepy, utilizzata per i primi 3 punti del progetto, e NetworkX, per i rimanenti 7 punti.
Al centro della nostra analisi vi è il noto social network Twitter: un social finalizzato alla comunicazione e alla divulgazione di informazioni tramite lo scambio di messaggi di testo di al massimo 280 caratteri che prendono il nome di "tweet", e per cui si contano oltre 500 milioni di iscritti e 330 milioni di utenti attivi in tutto il mondo, motivo per cui gli oltre 5780 tweet al secondo lo rendono uno dei più grandi database al mondo da cui poter estrapolare qualsiasi tipo di informazione.

Tweepy

Tweepy è una libreria Python per l'accesso alle API di Twitter, consente agli sviluppatori di integrare facilmente le funzionalità di Twitter nelle proprie applicazioni Python. Con Tweepy è possibile eseguire operazioni quali la lettura e la scrittura di tweets, la ricerca di tweet, utenti e relativi followers. Per utilizzare la libreria Tweepy è, dunque, necessario aver accesso agli strumenti da sviluppatore Twitter indispensabili per estrarre i dati pubblicati sulla piattaforma. Ogni componente del gruppo ha dovuto creare dapprima un account Twitter e in seguito un account Twitter Developers che fornisce all'utente le credenziali d'accesso alle API, tra cui il bearer token. Installata la libreria ed importata all'interno del progetto, è possibile quindi utilizzare la funzione `tweepy.Client()` per autenticarsi tramite `BEARER_TOKEN` permettendo di creare un `Client` che fungerà da interfaccia per le API di Twitter.

NetworkX

NetworkX, invece, è una libreria Python che fornisce strumenti per la creazione e la manipolazione di reti complesse. Può essere utilizzata per creare e analizzare grafi, ovvero rappresentazioni matematiche di reti formate da un insieme di punti (vertici o nodi) e un insieme di linee (archi) che uniscono coppie di nodi. NetworkX può essere utilizzato per modellare e studiare un'ampia gamma di sistemi complessi, tra cui reti sociali, biologiche e tecnologiche. È uno strumento potente per comprendere la struttura e la dinamica dei sistemi complessi e può aiutare i ricercatori e gli analisti a comprendere meglio le relazioni tra i vari elementi di una rete. Nel nostro caso utilizzeremo NetworkX per analizzare i followers di un utente, @KevinRoitero, analizzeremo le relazioni di following tra essi e infine andremo a svolgere varie misurazioni sul grafo risultate.

Svolgimento

Raccolta dati

Punto 1

Prima di iniziare abbiamo definito due funzioni: una per salvare dati JSON in locale e l'altra, invece, per leggere file JSON da locale.
A questo punto abbiamo potuto finalmente iniziare il nostro progetto! Per il primo punto ci è stato richiesto di utilizzare la libreria Tweepy e scaricare tutti i follower del profilo Twitter con username @KevinRoitero, scaricando per ciascuno di essi: attributi di default, descrizione del profilo, metriche pubbliche dell'account (followers_count, following_count, tweet_count, listed_count) e lo stato di protezione dell'account, infine salvare il risultato in

una unica serializzazione JSON. Per fare ciò abbiamo creato una funzione `get_parsed_users_followers(id)` che sfrutteremo anche nei punti successivi e che, specificato l'id dell'utente di interesse, ci restituisce, grazie alla funzione `client.get_users_followers` di Tweepy, un dizionario di tutti i suoi followers con le relative informazioni richieste nella consegna del punto 1. In questo passaggio, inoltre, per aggirare il limite imposto da Twitter di 100 risultati abbiamo usato il `tweepy.Paginator` di Tweepy che ci ha permesso di paginare i risultati del `client`. Abbiamo quindi fornito come parametro l'id dell'utente @KevinRoitero e salvato tutte le informazioni in un file JSON utilizzando la funzione da noi definita nei primi passi.

Punto 2

Andando avanti con il progetto, per ciascuno dei follower di @KevinRoitero siamo riusciti a risalire al numero di tweet pubblicati da ogni profilo durante l'ultima settimana utilizzando la funzione `client.get_recent_tweets_count` di Tweepy, la quale, però, ci forniva per ogni utente solamente il conteggio rispettivo al singolo giorno e quindi, per ogni utente, abbiamo dovuto sommare i diversi risultati giornalieri per ricavare il numero di tweet totale degli ultimi sette giorni. Dopo, per ogni follower, abbiamo aggiunto una variabile denominata `last_week_tweets_count` contenente il risultato precedentemente trovato, infine abbiamo serializzato il tutto nel file JSON creato in precedenza.

Punto 3

Seguendo il punto 3, abbiamo preso in considerazione all'interno della lista di follower di @KevinRoitero solamente gli utenti con un profilo non protetto e che avessero almeno un follower. Abbiamo scaricato per ognuno di essi al massimo 1000 follower utilizzando ancora la funzione da noi in precedenza definita `get_parsed_users_follower(id)`, salvando quindi per ciascuno di essi le stesse informazioni del punto 1 e aggiungendo poi il risultato sotto la dicitura `followers` nel JSON fatto in precedenza.
Per scaricare tutte le informazioni ci sono volute più di due ore, di conseguenza per facilitare l'analisi, abbiamo creato una copia denominata `followers_completed.json` che, tramite la funzione `read_json`, potrà essere utilizzata per i punti successivi.

Creazione dei grafi

Punto 4

Arrivati al punto 4 abbiamo messo da parte la libreria Tweepy e iniziato a lavorare con NetworkX, ovvero una libreria open source che si occupa della creazione e manipolazione di grafi, e che noi utilizzeremo per analizzare i dati appena ottenuti ed estrapolati da Twitter, effettueremo delle misurazioni al fine di fare una analisi delle cerchie sociali del profilo di Kevin Roitero. Una volta scaricata e importata la libreria abbiamo costruito un grafo che avesse come nodi gli id di Kevin Roitero e dei suoi follower, e come attributi (username, descrizione, numero di follower) gli stessi dei profili twitter presi in considerazione, tutto ciò utilizzando la funzione `Graph.add_node`. Abbiamo quindi effettuato un check basandoci sulle informazioni di cui disponiamo dal file JSON ottenuto al punto 3 per verificare le relazioni di following tra i profili e, qualora due profili fossero coinvolti in tale relazione, sono stati connessi da degli archi grazie alla funzione `Graph.add_edge`. In questo modo si è andato a creare un grafo diretto, ovvero un grafo in cui gli archi sono caratterizzati da una direzione e, di conseguenza, in cui il grado dei nodi si misura sia in archi entranti (In-Degree) che in archi uscenti (Out-Degree).

Punto 5

Al punto seguente, tuttavia, ci è stato richiesto di trasformare il grafo da diretto a indiretto; pertanto, abbiamo utilizzato la apposita funzione `Graph.to_undirected()` andando, in questo modo, a creare un grafo non orientato in cui non vi è più la distinzione tra archi entranti ed archi uscenti, e tale per cui vi siano 2 rappresentazioni equivalenti per lo stesso arco, essendo, esso, per l'appunto, non direzionato. Dopodiché, dato che il nostro obiettivo era quello di ampliare il grafo già ottenuto, abbiamo aggiunto nodi e archi fino ad ottenere il doppio dei nodi rispetto al grafo di partenza, seguendo il metodo del "Preferential Attachment", abbiamo utilizzato la funzione di NetworkX chiamata `barabasi_albert_graph` per far sì che i nuovi nodi si colleghino tramite archi agli altri nuovi nodi proporzionalmente al grado di questi ultimi. Essenza dell'algoritmo "Preferential Attachment", è, infatti, proprio questa tendenza dei nodi a collegarsi ad altri nodi a loro volta già molto collegati e connessi e che quindi caratterizzati da un alto grado. Meccanismo, questo, studiato dall'omonimo ricercatore Albert Laszlo Barabasi, il quale scorse proprio in questo algoritmo l'origine del fenomeno, molto frequente nel mondo reale, nelle reti "scale-free", ovvero quelle reti in cui, tracciando i gradi dei nodi sull'asse x e la probabilità di avere quei gradi nella rete sull'asse y, troviamo che la

distribuzione dei gradi risultante su una scala logaritmica sarà una linea retta (distribuzione "Power Law"), e in cui possiamo notare una maggiore presenza di nodi con basso grado e, invece, una ristretta cerchia di nodi con un grado di collegamenti che supera la media (Hub).

Visualizzazione dei grafi

Punto 6

Una volta riusciti a creare i due grafi richiesti, uno diretto e l'altro indiretto, andremo a visualizzarli graficamente testando sia le funzionalità della libreria PyVis, per ottenere una visualizzazione interattiva, che quelle della libreria NetworkX, per la versione statica.
Per quanto riguarda la prima modalità, è bastato usare per entrambi i grafi la funzione `pyvis.network.Network`, mentre per la visualizzazione senza animazioni abbiamo usato `nx.draw_networkx` e impostato opportunamente i parametri di visualizzazione applicando il layout di Fruchterman Reingold, il quale visualizza i nodi del grafo come se fossero legati tra loro da una forza.

Misurazioni

Punto 7

Una volta raccolti tutti i dati necessari è arrivato il momento di analizzarli.
Come prima misurazione abbiamo identificato in entrambi i grafi la componente connessa più grande (SCC), ovvero un sottografo in cui ogni coppia di nodi è connessa tra loro da almeno un cammino e la cui dimensione dipende dal parametro p (probabilità che presi due nodi si trovi un arco fra loro). Con $p = 0$, il grafo è disconnesso e la componente connessa è 1, mentre se $p = 1$ allora avremo un grafo completo e la componente connessa sarà n, ovvero tutto il grafo. Tale fenomeno è stato riscontrato anche nella nostra analisi. Come ben si può vedere dai due grafici, mentre i nodi coinvolti nella SCC del grafo indiretto (*graph2*), essendo esso strettamente connesso, coinvolgono tutti i nodi, nel caso del grado diretto (*graph1*), essendo invece debolmente connesso, ne coinvolgono solo una parte.

Punto 8

Andando avanti con il punto 8 del progetto ci è stato richiesto di misurare raggio, centro, distanza massima e distanza media dei due grafi.

Prima di riportare tutte le misurazioni ottenute vorremmo chiarire le definizioni dei vari termini:
Il *raggio* di un grafo è il valore che indica l'eccentricità minima dei suoi vertici, ovvero la distanza minima alla quale un nodo può essere trovato da tutti gli altri.
Seguendo questa definizione, il *centro* non è altro che l'insieme di tutti i nodi del grafo di minima eccentricità.
La *distanza massima*, anche chiamata "diametro", è l'eccentricità massima ovvero la più grande distanza possibile che può esistere tra due nodi, mentre per *distanza media* s'intende la lunghezza media del percorso più breve.

Tali valutazioni le abbiamo calcolate usando, rispettivamente, la funzione `nx.radius(graph2)` per il raggio, `nx.center(graph)` per il centro, `nx.diameter(graph)` per la distanza massima e, infine `nx.average_shortest_path_length(graph)` per la distanza media.

Le misurazioni ottenute dal grafo indiretto (graph2) sono le seguenti:

Parametri	Valori
Centro	[3036907250, 2582519183, 998681467696885761, 553507111, 1275818050848206848, 28607182, 1546349170315120640, 1520733236678299650, 2989739661, 1511721268558630916, 248721193, 1708743114, 1269179580843200512, 465693357, 75525450, 1222505104080809985, 833597084167122945, 40576466, 1387788296215015428, 29866761, 1190461842922995712, 22597445, 1330708819761004545, 554687023, 338989477, 1187585868493537280, 39697728, 1265795638551212032, 184125091, 839139788846411777, 1111315973125169152, 391867472, 514667085, 18874262, 393675585, 249661913, 2382305514, 130671142, 16868154, 59217922, 934851466082357249, 94732055, 947311503216066560, 458399248, 78969930, 4053818032, 406303880, 132646210, 14451127, 3102522680, 3033312071, 2309104237, 15455450, 125483940, 436609034, 308456550, 810893744593584128, 18932422, 341000847, 2190533245, 2601589496, 15750573, 150]
Raggio	3
Distanza media	2.5298786964056124
Distanza massima	5

Per quanto riguarda, invece, il grafo diretto (graph1), esso risulta essere debolmente connesso, essendo che nel grafo abbiamo considerato solo i followers dell'utente Kevin Roitero e non anche i suoi seguiti, motivo per cui il nodo di Kevin Roitero non sarà connesso ad alcun altro nodo, di conseguenza non sarà possibile effettuarvi alcuna misurazione. Per ovviare a questo inconveniente una soluzione potrebbe essere quella di convertire il grafo da diretto a indiretto, ed infatti, utilizzando la funzione `graph1.to_undirected()` siamo riusciti ad ottenere i seguenti risultati:

Parametri	Valori
Centro	[3036907250]
Raggio	1
Distanza media	1.8938390753001908
Distanza massima	2

Per ognuno dei due grafi, inoltre, abbiamo voluto visualizzare visivamente i nodi che costituivano il centro del grafo, e lo abbiamo fatto andando a raffigurare il grafo con la libreria NetworkX evidenziando in rosso i punti coinvolti.

Punto 9

Proseguendo la nostra analisi ci siamo soffermati sul concetto di "*centralità*", ovvero quanto un nodo è centrale/importante nel grafo, sulla base di diversi aspetti.
Anzitutto abbiamo calcolato la "Betweennes Centrality" ("Centralità di mezzo") che indica "quanto è rilevante un nodo nella comunicazione tra diverse parti della rete", o più semplicemente "quante volte bisogna passare per quel nodo per raggiungerne un altro", la funzione apposita è `nx.betweenness_centrality(graph)`.
Dopodiché abbiamo misurato, la "Closeness Centrality" ("Centralità di prossimità"), indicatore di quanto un nodo è "vicino" agli altri nodi della rete, in termini di cammino più breve, e la cui funzione associata è `nx.closeness_centrality(graph)`.
Per la "Degree Centrality" ("Centralità di grado"), che misura l'abilità di un nodo di comunicare direttamente con altri nodi e tale per cui più è alto il grado di un nodo e più quel nodo è centrale, abbiamo utilizzato per il grafo indiretto la funzione `nx.degree_centrality(graph)`, mentre per il grafo diretto le due funzioni `nx.in_degree_centrality(graph)` e `nx.out_degree_centrality(graph)` che calcolano rispettivamente il "grado di archi entranti", quindi il numero di archi che il nodo riceve, e "il grado di archi uscenti", ovvero il numero di collegamenti orientati verso i nodi adiacenti.
Se fino ad ora, però, abbiamo classificato i nodi solamente per la loro rilevanza, per la prossimità, per il grado, ora andremo a utilizzare metodi di ranking che andranno a classificare i nodi anche per la qualità dei loro collegamenti.
Il primo algoritmo di ranking che abbiamo preso in considerazione è "Page Rank", un algoritmo che ha lo scopo di assegnare un punteggio da 0 a 10 di importanza ad ogni nodo in base ai link entranti e al loro precedente punteggio di Page Rank, tutto questo con la funzione `nx.pagerank(graph)`.
Il secondo metodo che abbiamo utilizzato è, invece, l'algoritmo di Kleinberg "Hits", il quale classifica i nodi in 2 tipologie: Hub (nodi con tanti link uscenti verso altri nodi) e Authority (nodi con tanti link entranti), con l'appunto che, un nodo, per essere un buon Hub, deve linkare tante buone Authority, e , per essere una buona Authority, deve essere linkata da molti buoni Hub.
La funzione utilizzata per calcolare questo ultimo algoritmo si chiama `nx.hits(graph)`.
Infine sono stati tracciate tutte queste misurazioni in opportuni grafici.

Punto 10

Per concludere il nostro progetto abbiamo calcolato la "Small Worldness" del nostro grafo indiretto, escludendo quello diretto in quanto la funzione che andremo a utilizzare non sarà definita per i grafi orientati. Per Small Worldness usualmente s'intende quel fenomeno molto comune nelle reti reali che osserva se la distanza tra due nodi qualsiasi della rete è relativamente piccola rispetto alle dimensioni della rete, in altre parole: la maggior parte dei nodi del grafo può essere raggiunta da ogni altro nodo attraverso un piccolo numero di salti/passi.
Per analizzare questo aspetto ci siamo serviti di due parametri: Omega e Sigma.
Il primo misura quanto il grafo assomigli più a un reticolo, per valori negativi, o a un grafo casuale, per valori positivi, e sulla base di questo, per valori vicini allo 0 si può stabilire che il grafo abbia caratteristiche di piccolo mondo (Small World). Il secondo parametro, invece, stabilisce che il grafo può essere comunemente classificato come mondo piccolo, solamente per valori di sigma maggiori di 1. I risultati ottenuti sono stati quindi riportati nella seguente tabella:

Parametri	Valori
Omega	0.1973801941208444
Sigma	1.3457438647725815

Sulla base dei valori ricavati, possiamo concludere, che, essendo Omega un valore prossimo allo 0 e Sigma maggiore di 1, il nostro grafo indiretto si può sicuramente definire una rete "piccolo mondo" in cui la distanza tra i nodi è, quindi, minima.