PRIMO PROGETTO PER IL CORSO DI SOCIAL COMPUTING A.A. 2020/21

Introduzione

Per iniziare a realizzare questo progetto abbiamo deciso, prima di tutto, di distribuire il carico di lavoro per ciascun membro, in modo tale che i tempi per scaricare tutti i dati da Twitter si accorciassero e il lavoro progredisse in maniera lineare.

Abbiamo iniziato a scrivere il codice importando i vari pacchetti iniziali (come os, json, tweepy e ppprint), per poi successivamente importarne altri ogni qualvolta servisse.

Ciascun membro ha inserito i propri codici identificativi e segreti per accedere alle API di Twitter, per poi definire le funzioni **serialize_json()** e **read_json()** viste precedentemente a lezione. La prima funzione, inserendo come input una folder, un nome di un file e una serie di dati, serve a serializzare quest'ultima salvandola in un file dentro folder, e trasformare i dati in formato json. La seconda invece agisce in modo inverso, quindi dato come input un file in formato json, è possibile estrapolare e leggere i dati da quest'ultimo, per poi elaborarli.

1 – Scaricare utenti follower e following

Siamo passati a realizzare il primo punto del progetto, ovvero scaricare attraverso le API tutti gli utenti follower e following dei cinque utenti forniti dal testo del progetto. Abbiamo deciso di assegnare un utente ad ogni membro del gruppo (un membro ha lavorato su due utenti) e di creare due parti di codice, una per i followers e una per i followings. Dato il numero considerevole di utenti da scaricare abbiamo usato un cursore dove ad ogni iterazione di esso, attraverso le funzioni **api.followers** (per i followers) e **api.friends** (per i followings), ci scaricava gli utenti in base a ciascun screen_name dato come input. Per ognuno di questi utenti abbiamo voluto inserire in un array l'Id, il name, screen_name, e la location, dandogli sempre un controllo che non permetteva l'inserimento di un utente se già presente. Finito questo ciclo del cursore, abbiamo salvato il tutto in un nuovo array, per poi serializzarlo attraverso la funzione **serialize_json()**. Una volta scaricato tutti i follower e i following di ogni utente ci siamo ritrovati nel nostro folder diversi file json, ciascuno relativo ai follower e ai following dei nostri cinque utenti iniziali.

Successivamente, per avere meno file json da gestire, tramite la funzione **.extend()** abbiamo unito i vari file json in modo tale da avere solo due file, uno per tutti i follower degli utenti e uno per tutti i following(*all_prof_followers.json*, *all_prof_followings.json*).

2 – Scegliere 5 utenti followers a caso tra quelli di ciascuno dei cinque account e scaricate ulteriori 10 utenti followers

In questa fase, abbiamo importato la libreria random e tramite un ciclo for sono stati estratti cinque utenti casuali e salvati in un file chiamato *random_name.json* nella cartella Follower (data/Follower). Grazie a questo file abbiamo applicato la procedura **api.followers** utilizzando come input il nostro *random_name.json*.

Questa procedura varia leggermente da quella precedente perché all'interno del cursore è stata inserita una quantità limite di dieci items per utente random.

Anche qui il lavoro di acquisizione dei dati è stato suddiviso tra i vari membri del gruppo.

3 – Scegliete 5 utenti following a caso tra quelli di ciascuno dei cinque account e scaricate ulteriori 10 utenti following (following dei following)

A questo punto è stato facile procedere anche con i following in quanto è bastato sostituire la funzione **.followers()** con la funzione **.friends()**, cambiare la nostra cartella di destinazione (data/Following) e salvare un nuovo file *random_name.json* scaricando i relativi dieci following.

4 – Scaricare i dettagli del profilo di tutti gli utenti recuperati

Questa fase ha richiesto molto tempo in quanto la mole di dati era consistente e le limitazioni di Twitter non favorivano la velocità del lavoro.

Quindi abbiamo preso in considerazione tutti gli utenti scaricati fino a questo punto, comprensivi dei cinque utenti iniziali, tramite la funzione **api.get_user()**, a cui come input abbiamo assegnato lo screen_name degli utenti in quanto utilizzando l'id riscontravamo dei problemi nell' identificazioni di quest'ultimi, sono stati creati altri file json contenenti id, name, screen_name, location, description, followers_count.

Facendo tutto questo siamo riusciti ad ottenere i dettagli dei nostri utenti che saranno necessari per i punti successivi del nostro progetto.

5 – Costruzione della rete sociale

Importando le librerie Networkx, pandas e numpy siamo passati alla creazione del grafo con la funzione **nx.Graph()**. Per ogni file json che racchiude i dettagli degli utenti trovati nel punto precedente, attraverso la funzione **.add_node()** abbiamo inserito ciascuno di questi utenti nel nostro grafo utilizzando come identificatore del nodo il proprio ID, trovando oltre tremila nodi. All'interno di questa funzione abbiamo aggiunto inoltre degli attributi che specificavano il name, lo screen_name, la location, la description e il followers_count, inserendo anche come attributi del grafo i nomi e cognomi dei membri del gruppo.

Per la creazione degli archi avevamo bisogno di un file json che comprendesse i cinque utenti iniziali, i loro followers e followings e tutti i followers e followings degli utenti random, cosi abbiamo unificato questi dati in un unico file *all_user.json*.

Grazie a questo passaggio siamo stati in grado di verificare la relazione, attraverso la funzione **.show_friendship**, tra i nostri cinque utenti iniziali e tutti gli utenti del file *all_user.json,* creando i relativi archi quando presente una relazione di follows.

"Un metodo alternativo, non utilizzato durante questo progetto, per la creazione degli archi consisteva nel prendere in considerazione ogni singolo utente iniziale e il proprio relativo file contenete follower e following creando manualmente gli archi."

Durante la fase iniziale di questo passaggio abbiamo riscontrato problemi, in quando certi utenti non venivano identificati (probabilmente a causa di variazioni sugli account Twitter di alcuni utenti). Per risolvere questo errore abbiamo implementato il nostro ciclo for con un try e except, che ogni qual volta trovato un errore passava all'utente successivo senza bloccare l'esecuzione del codice.

In fine ogni membro del gruppo ha salvato il grafo relativo ad ogni utente a lui assegnato in un file .pkl tramite la funzione **nx.write_gpickle()**.

In questa fase è stato deciso dal gruppo di verificare anche la relazione tra i nostri utenti random e tutti gli utenti follower e following scaricati da quest'ultimi e attraverso **add_edges_from** aggiungere gli archi di tutti i file pkl nel nostro grafo.

6 – Visualizzazione interattiva del grafo

Al punto sei del progetto viene importata la libreria di pyvis. Prima della visualizzazione interattiva abbiamo deciso di controllare la correttezza del nostro grafo attraverso un'immagine statica, riscontrando però dei nodi isolati probabilmente dovuti al try except utilizzato durante la creazione degli archi. Per non avere problemi nel completamento del progetto, utilizzando la funzione **.remove_nodes_from(list(nx.isolates()))**, abbiamo eliminato tutti i nodi isolati.

Quindi fatto questo è stato possibile creare senza errori la visualizzazione interattiva richiesta. Dopo aver impostato i vari parametri della pagina html (come dimensione, colore di sfondo e font) nella funzione Network e preparato il layout con la funzione **nx.barnes_hut(),** abbiamo inserito il nostro grafo nella pagina html.

7 – Verifiche sul grafo (connesso, bipartito)

Tramite le funzioni **nx.is_connented** e **nx.is_bipartite** siamo arrivati alla conclusione che il nostro grafo è connesso ma non bipartito.

8 – Distanze sul grafo

Successivamente siamo passati a calcolare le varie distanze sul grafo, ovvero il centro(**nx.center()**), il diametro(**nx.diameter()**) e il raggio(**nx.radius()**) trovando rispettivamente i seguenti risultati:

[132646210, 19659370, 3036907250]	che corrispondono a Damiano10, eglu81, KevinRoitero
6	diametro del nostro grafo
3	raggio del nostro grafo

9 – Centralità del grafo

A questo punto del progetto ci veniva richiesto di calcolare le varie centralità del grafo:

1. Betweenness centrality
2. Closeness centrality
3. Degree centrality
4. In-degree centrality
5. Out-degree centrality
6. Page Rank
7. HITS

Tutte queste centralità sono state eseguite con la relativa funzione, per alcune di queste, che richiedono un grafo diretto (4, 5, 6, 7), abbiamo utilizzato la funzione **.to_directed()** che trasforma un grafo indiretto in uno diretto. Inoltre per verifica, dopo ogni funzione, abbiamo visualizzato anche i grafici delle centralità dove possibile, attraverso la funzione **draw()**.

10 – Sottografo del grafo

Successivamente siamo passati alla creazione del sottografo Kevin_graph tramite la funzione **nx.ego_graph()** che prende in input il nostro grafo, l'ID di KevinRoitero e il radius. Per una visualizzazione più chiara abbiamo creato un array color_map che, con un ciclo for, da un colore per il nodo di Kevin e uno per il resto dei nodi e impostato una trasparenza su quest'ultimi. Inoltre abbiamo impostato la dimensione di ogni nodo in base al grado di quest' ultimo. Il nostro sottografo contiene 310 nodi e 735 archi.

Creato il sottografo siamo passati alla cricca massima, ovvero il sottoinsieme più grande di nodi in cui ogni coppia di nodi è adiacente. Per crearla abbiamo usato la funzione **.max_clique()** e con la funzione **.large_clique_size()** ci ha restituito la sua dimensione che nel nostro caso equivaleva a 7.

11 – Copertura minima degli archi

Al punto undici del progetto, abbiamo calcolato attraverso la funzione **.min_edge_cover()** la copertura minima degli archi del nostro grafo, che ritorna il più piccolo set di archi tale che ogni nodo del grafo è incidente ad almeno un arco del set.

12 – Small-world-ness

Successivamente con le funzioni **nx.omega() e nx.sigma()** abbiamo calcolato i rispettivi coefficienti omega e sigma, applicando al loro interno due parametri per ridurre il tempo di esecuzione, ovvero niter = 10 e nrand = 3.

Alla fine dell'esecuzione le due funzioni ci hanno restituito due valori:

omega	0.32439197506528494
sigma	0.6784089661255454

13 – Correlazione di Pearson Rho e di Kendall Tau

All'ultimo punto bisognava calcolare la correlazione di Pearson Rho e di Kendall Tau prendendo in considerazione le centralità calcolate.

Prima di tutto abbiamo importato la libreria *from scipy.stats import pearsonr, kendalltau* poi utilizzando due cicli for annidati abbiamo creato tutte le coppie possibili (senza ripetizioni e inversioni) tra le centralità. Attraverso le funzioni *scipy.stats.pearsonr* e *scipy.stats.kendalltau*, dandogli come argomenti le coppie create in precedenza, siamo riusciti a calcolare i coefficienti necessari.

Conclusioni

Grazie a questo progetto siamo riusciti ad applicare la teoria fornita durante le lezioni di Social Computing e imparato a risolvere i vari problemi riscontrati durante lo svolgimento dello stesso.

In conclusione, dal lavoro svolto siamo in grado di osservare il nostro grafo con all'interno tutti i dati scaricati e una serie di funzioni che gli elabora al fine di rispondere a domande del tipo: "Quali solo gli utenti più centrali?", "Da quanti nodi e archi è formato il nostro grafo?" oppure "Come sono distribuite i nodi in base alle loro relazioni?".

Membri del gruppo:

Marco Tateo	145194
Leonardo Scandino	142700
Leo Baric	137424
Matteo Palomba	140962

Sommario