Web and Social Information Extraction

Documentazione del progetto

Leonardo Mainardi *(Matr. 1310167)*

1 Task 1

1.1 Estrazione di utenti correlati

Il primo task richiesto dalle specifiche consiste nell'individuare un piccolo gruppo di utenti presumibilmente correlati. Ovviamente esistono svariati approcci per risolvere questo problema: tuttavia, molti di questi risultano essere eccessivamente onerosi dal punto di vista computazionale, mentre molti altri richiederebbero un numero elevato di *query* al database di Twitter, e questo rappresenterebbe un ostacolo non irrilevante, dal momento che la libreria Twitter4J utilizzata nel progetto permette solo un numero ristretto di interrogazioni in un preciso intervallo di tempo, limitando di fatto la quantità di informazioni disponibili in un tempo ragionevole. La scelta della miglior soluzione per questo primo task, quindi, ha dovuto tenere in considerazione innanzitutto questi due vincoli, cercando però, allo stesso tempo, di conciliarli con un'accuratezza dei risultati che fosse abbastanza soddisfacente.

L'approccio scelto, dunque, è basato sull'utilizzo dei soli dati offerti dal dataset, ossia dalla raccolta dei tweet fornita con il progetto. In particolare, l'idea si basa su una rappresentazione dei vari utenti come vettori, grazie alla quale è stato possibile clusterizzarli e individuare così quelli più "vicini". Si noti che in un approccio di questo tipo, un fattore determinante nella precisione finale dei risultati è la scelta delle coordinate da usare per rappresentare ogni utente, la quale deve essere coerente con il tipo di informazione che si vuole ottenere, ma anche con i dati contenuti nel dataset utilizzato: gli elementi in base ai quali confrontare i vari utenti, cioè, non dovrebbero essere aprioristici, bensì ricavati dal dataset stesso, in modo da adattarsi il più possibile ai dati da classificare.

Nella soluzione presentata, quindi, si è tenuto conto di tutto ciò che è stato detto fin qui. Più precisamente, essa si compone di tre punti fondamentali:

- Scelta delle coordinate e rappresentazione degli utenti come vettori
- Clustering
- Calcolo delle distanze e dei vettori più simili

Di seguito, viene spiegata in dettaglio ognuna di queste tre fasi.

Scelta delle coordinate e rappresentazione degli utenti come vettori Per rappresentare i vari utenti come vettori, si è scelto di utilizzare come coordinate il numero di occorrenze dei 10 utenti più menzionati: in altre parole, per ogni utente, si è proceduto raccogliendo tutti i tweet da lui pubblicati e contando, all'interno di essi, quante volte veniva menzionato ognuno dei 10 utenti scelti come coordinate, determinando così i valori da inserire all'interno del vettore. Ad esempio, poniamo che, in seguito all'analisi del dataset, abbiamo individuato che i 10 utenti più menzionati sono *user1*, *user2*, ... *user9*, *user10*. A questo punto, per rappresentare l'utente *Alice* come vettore, si procede individuando tutti i tweet da lei pubblicati, che potrebbero essere i seguenti:

1. *Penso che @user1 ha abbia sbagliato a ritirare la sua candidatura alle elezioni*
2. *@user3 sta per rilasciare un'intervista*
3. *@user8 è stato accusato di poca trasparenza da @user3*

Dall'analisi dei tweet, si ricava facilmente che *user1* e *user8* sono stati citati 1 volta, mentre *user3* è stato menzionato 2 volte: il vettore relativo ad Alice, quindi, sarà $(1, 0, 2, 0, 0, 0, 0, 1, 0, 0)$.

Il senso di tutto ciò è che, verosimilmente, utenti che citano le stesse persone sono correlati tra loro: in particolar modo questa evenienza dovrebbe verificarsi in un dataset riguardante la politica, specialmente se le citazioni analizzate si riferiscono a personalità rilevanti. Questo spiega il motivo per cui si è scelto di utilizzare come coordinate gli utenti più menzionati: questi, infatti, molto probabilmente rappresenteranno persone particolarmente conosciute e "di spicco", e ciò rispecchia proprio ciò che vorremmo. Effettivamente, i risultati empirici ottenuti da questa analisi coincidono con le nostre ipotesi: gli utenti più menzionati all'interno del dataset, infatti, risultano essere: *matteorenzi, matteosalvinimi, beppe_grillo, forza_italia, GiuliaDiVita, GiorgiaMeloni, angealfa, borghi_claudio, pdnetwork, ale_dibattista*. I vari utenti del dataset, quindi, sono stati rappresentati come vettori le cui coordinate sono proprio le occorrenze di questi nomi all'interno dei tweet di ogni utente.

Clustering Una volta "tradotti" tutti gli utenti in vettori, è possibile usare tecniche di *clustering* per raggrupparli in insiemi di vettori "simili": ciò equivale ad individuare gruppi di utenti accomunati dal fatto di citare (e anche di *non* citare) le stesse persone. Tuttavia, un ostacolo comune a tutti i problemi di *clustering*, è che difficilmente è possibile stabilire in anticipo il numero di insiemi in cui raggruppare le istanze: anche in questo caso, quindi, non conoscendo a priori questa informazione, si è dovuto procedere per tentativi, alla ricerca del valore che meglio suddivideva i dati. Con l'ausilio del software Weka, perciò, sono state eseguite diverse clusterizzazioni dei vettori, fino ad ottenere una sufficiente differenziazione tra i vari gruppi. Nella fig.1 è riportata la tabella corrispondente al miglior risultato di clustering ottenuto.

Calcolo delle distanze e dei vettori più simili A questo punto, si è potuto procedere scegliendo un gruppo tra quelli creati ed estraendone gli utenti più rappresentativi: più precisamente, dopo aver individuato il gruppo più "coeso", ne è stato calcolato il centroide, dopodiché sono stati estratti i 4 vettori più vicini a tale centroide. Si noti che la scelta della misura da utilizzare è ricaduta sulla *distanza euclidea* e non su tecniche quali la *cosine similarity*: questa scelta è dovuta al fatto che per stabilire se un'istanza è "simile" al centroide è necessario basarsi sulla distanza effettiva tra i 2 punti, e non sui rapporti tra le loro coordinate.

Un altro punto da chiarire, infine è il significato di gruppo "coeso": con questa espressione, si intende indicare un gruppo che rappresenti in modo abbastanza univoco un insieme di utenti. A titolo di esempio, si faccia riferimento ai cluster #6 e #7 in fig.1. Entrambi questi cluster hanno come coordinata "dominante" l'utente *matteosalvinimi*: tuttavia, sebbene abbiano centroidi su valori differenti di tale coordinata, avrebbe comunque poco senso distinguere gli utenti in base all'appartenenza all'uno o all'altro di questi 2 cluster[1]. Al contrario, un gruppo come quello rappresentato dal cluster #10 può dirsi "coeso", in quanto è l'unico raggruppamento di istanze posto sull'asse *GiorgiaMeloni*. Per la soluzione del progetto, quindi, è stato scelto proprio questo cluster, sia per il motivo appena spiegato, sia perché tale raggruppamento è supportato da un numero sufficiente di istanze. Calcolando le distanze dei vettori dal centroide di questo cluster, quindi, si è ottenuto che gli utenti più vicini sono *DPLuci, catefunel, _marymod, albertocardillo*, che effettivamente sono tutti militanti del partito rappresentato dal cluster scelto.

Cluster # / Attr.	0 *(2663)*	1 *(915)*	2 *(10465)*	3 *(47418)*	4 *(546)*	5 *(94)*	6 *(887)*	7 *(809)*	8 *(4017)*	9 *(78)*	10 *(294)*	11 *(64)*
matteorenzi	0.0214	0.1246	0.1069	1.1449	1.2821	1.0851	0	0	0.0889	0	0	0.0313
matteosalvinimi	0.0004	0.0011	0.0038	0.1109	0	0	11.3202	47.7379	0.0072	0	0	0
beppe_grillo	0.154	7.2721	1	0.0001	0.0037	0	0	0	2.5026	0	0	28.6875
forza_italia	0.0015	0.0022	0.0046	0.2158	0.0147	0	0	0	0.003	43.6795	0	0
GiuliaDiVita	0.0315	0.0022	0.0087	0.0531	0	0	0	0	0.0092	0	0	0
GiorgiaMeloni	0	0	0.0011	0.1223	0.0037	0	0.0011	0	0.0037	0	56.3605	0
angealfa	0.0008	0	0.0036	0.0943	0.0037	15.8723	0.0011	0	0.0047	0	0	0
borghi_claudio	0.0008	0	0	0.0777	0	0	0.0023	0	0	0	0	0
pdnetwork	0.0049	0.012	0.0069	0.1189	5.0183	0	0	0	0.0075	0	0	0
ale_dibattista	1.4078	0.0142	0	0	0	0	0	0	0	0	0	0.0156

Figura 1: Risultato del *clustering* dei vettori corrispondenti agli utenti del dataset. I valori tra parentesi nell'intestazione della tabella indicano il numero di istanze contenute nel *cluster* corrispondente, mentre i valori all'interno della tabella rappresentano le coordinate dei centroidi.

[1] A prima vista potrebbe sembrare che questo fenomeno sia causato da un valore sovradimensionato del numero di cluster da creare: tuttavia, i vari test hanno dimostrato che questa situazione continuava a verificarsi anche con un numero minore di cluster.

1.2 *Crawling*

Una volta ottenuti i 4 utenti correlati, è stato possibile procedere eseguendo il *crawling* della rete di Twitter. In particolare, questa operazione è stata effettuata utilizzando come *seeds* proprio i 4 utenti individuati nel punto precedente, e, a partire da questi, è stata costruita la rete attraverso le relazioni di *following*: per ogni utente, cioè, sono stati estratti tutti gli utenti da esso seguiti, creando così un grafo diretto i cui archi connettono ogni persona con tutti i suo amici. Tuttavia, dal momento che le liste di amici potrebbero contenere anche milioni di utenti, il *crawling* è stato effettuato attraverso una visita in profondità, in cui il livello massimo di profondità era limitato a 2 (a partire dai *seeds*) e il grado uscente massimo era 25; inoltre, si è imposto (come da specifiche) che il grafo risultante avesse al più 250 nodi.

Dal punto di vista implementativo, come struttura dati per rappresentare il grafo è stata usata la classe `InMemoryGrph` fornita dalla libreria *Grph*, come suggerito nelle specifiche. Dal momento, però, che questa classe permette di gestire solo grafi i cui nodi sono rappresentati da interi positivi consecutivi, le informazioni riguardanti ogni utente (in particolare l'*id* numerico) sono state memorizzate all'interno di ogni nodo usando il metodo *setVertexLabelProperty()*. Questa complicazione, unita al fatto che per ricavare i *followings* di ogni utente ci si doveva interfacciare con la libreria Twitter4J, e con le relative limitazioni temporali, ha portato alla decisione di creare una classe apposita per effettuare l'operazione di *crawling*, chiamata appunto `Crawler`. Questa classe, quindi, si occupa di:

- Prendere in input i *seeds* e ricavarne, ricorsivamente, gli amici
- Creare ogni singolo nodo del grafo risultante e gestirne le proprietà (mappare cioè all'*id* numerico del nodo l'*id* dell'utente Twitter corrispondente)
- Gestire opportunamente la frontiera, in modo da rispettare i vincoli imposti sulla struttura dati e contemporaneamente ottenere un grafo coerente con la reale rete di Twitter
- Gestire le limitazioni temporali per le interrogazioni al database di Twitter, fermando l'esecuzione del programma fino a quando non si otteneva una nuova finestra di tempo per le query ed evitando, così, il lancio di eccezioni da parte della libreria Twitter4J

Relativamente al terzo punto, è bene approfondire le modalità secondo cui vengono creati gli archi del grafo. La maggior parte di essi viene creata durante la visita in profondità, sfruttando le relazioni di *following*: tuttavia, i vincoli imposti dalle specifiche potrebbero causare la mancata creazione di alcune connessioni, generando quindi delle inconsistenze con la reale rete di Twitter. Una situazione di questo tipo potrebbe verificarsi nel caso in cui 2 utenti, u_1 e u_2, abbiano un amico v in comune, ma questo venga elencato solo nella visita di uno dei 2 suoi *follower*. Ad esempio, supponiamo che v venga elencato tra gli amici di u_1, mentre venga ignorato

durante la visita di u_2 in quanto non rientra tra i primi 25 amici: in questo modo, all'interno del grafo ottenuto mancherebbe l'arco (u_2, v), nonostante sia u_2 che v appartengano al grafo. La soluzione proposta per evitare questa evenienza consiste nell'aggiungere una fase finale all'algoritmo, durante la quale vengono aggiornati tutti gli archi entranti di ogni nodo, in particolare aggiungendo quelli che hanno entrambi gli estremi all'interno del grafo. In questo modo, la rete risultante risulta essere precisamente un sottografo della reale rete di Twitter, all'interno del quale non manca alcun arco: gli unici archi eliminati, infatti, corrispondono solo a quelli che connettono nodi del sottografo a nodi esterni. Questa operazione viene effettuata all'interno del metodo *stabilize()* invocato, appunto, nella fase finale dell'algoritmo di *crawling*, ossia non appena viene raggiunta quota 250 nodi. L'intera procedura di *crawling* è rappresentata in fig.2. Il grafo ottenuto, invece, è illustrato in fig.3a.

1.3 PageRank

Per calcolare il PageRank per ogni nodo del grafo, è stato semplicemente invocato il metodo *getPageRanking()* esposto dalla classe `Grph`. L'unica accortezza che si è rivelata necessaria è stata quella di utilizzare la mappa creata contestualmente alla creazione dei nodi del grafo nel punto precedente per riuscire ad associare ogni nodo (rappresentato da un intero) al corrispondente utente Twitter, in modo da poter rendere leggibili i risultati stampati.

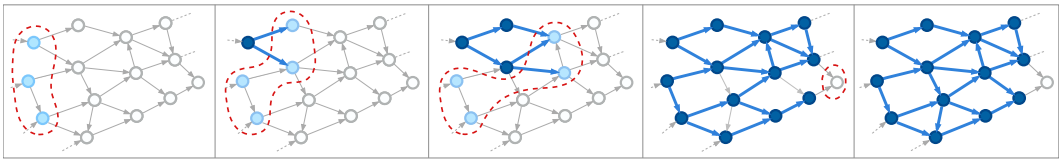

Figura 2: Fasi della procedura di *crawling*. In grigio è mostrata la rete reale di Twitter, mentre in blu sono evidenziati i nodi già visitati; i nodi celesti circondati dalla linea rossa tratteggiata rapresentano invece la frontiera. Per semplicità, i vincoli sul grafo sono ridotti a 2 vicini per ogni nodo e 13 vertici totali. (a) Fase iniziale, in cui nella frontiera sono presenti solo i *seeds*. (b) Primo passo del *crawling*, in cui vengono aggiunti i 2 nodi adiacenti al primo *seed* e questo viene segnato come visitato. (c) Grafo creato dopo 3 passi. Si noti che, a causa dei limiti imposti, non tutti i vicini dei nodi già visitati sono presenti nella frontiera. (d) Grafo creato al raggiungimento del limite di 13 vertici totali. Si noti come il nodo ignorato nel punto precedente sia ora presente tra i nodi visitati, grazie al fatto che esso è raggiungibile da un altro vertice: tuttavia, proprio per questo motivo, alcuni archi risultano assenti nel grafo generato. (e) Grafo ottenuto in seguito all'invocazione del metodo *stabilize()*: tutti gli archi della sottorete indotta dai nodi visitati sono ora presenti nel grafo generato.

2 Task 2

2.1 Influenza e supporto

Per calcolare i valori di influenza e supporto di ogni nodo è stato sufficiente i metodi *getOutEdgeDegree()* e *getInEdgeDegree()* che restituiscono, rispettivamente, il grado uscente ed entrante di un nodo. Come per il punto precedente, anche in questo caso è stata utilizzata la mappa tra i nodi e gli utenti per stampare i risultati.

2.2 Clustering del grafo

Per individuare due cluster all'interno del grafo, si è scelto di basarsi sulla misura di *centrality degree* degli archi, eliminando ad ogni iterazione l'arco con un valore più alto di centralità fino a disconnettere il grafo. Se dal punto di vista implementativo ciò non presenta particolari difficoltà, computazionalmente questo algoritmo risulta essere decisamente oneroso, tanto che eseguendolo sul grafo preso in considerazione, contenente 250 nodi, il tempo di esecuzione sarebbe nell'ordine di centinaia di anni [2]. Per risolvere questo problema, si è deciso di creare un grafo che approssimasse in maniera sufficientemente fedele il grafo originale e che permettesse, allo stesso tempo, un calcolo più efficiente della centralità degli archi.

2.3 Approssimazione del grafo ottenuto

Osservando il grafo ottenuto, riportato in fig.3a, si può notare facilmente che quasi tutti i nodi appartengono alla parte centrale e sono densamente connessi tra loro, mentre i pochi vertici rimanenti presentano un grado entrante o uscente compreso tra 1 e 2. L'idea, dunque, è stata quella di sostituire la componente centrale con un clique, diminuendone ovviamente il numero di nodi: il senso era quello di eliminare tutti i vertici non connessi con nodi "esterni", dal momento che questi non influivano in maniera sostanziale sulla topologia del grafo, ma ne aumentavano soltanto la dimensione. Si noti, tuttavia, che questa operazione deve essere eseguita rispettando alcuni vincoli, al fine di mantenere le proprietà del grafo originale: in particolare, è necessario che il clique generato contenga un numero strettamente maggiore del numero di nodi "esterni". Di seguito sono spiegati e dimostrati in maniera approfondita i dettagli dell'approssimazione eseguita. Il grafo finale è illustrato in fig.3b.

[2] Per convincersi che un tempo del genere è verosimile e coerente con il lavoro richiesto, si pensi che per ogni arco, ossia per ogni coppia di nodi, è necessario analizzare tutti i cammini minimi tra ogni ulteriore coppia di nodi del grafo, quindi si deve iterare su circa $n^2 \cdot n^2 = n^4$ nodi; dal momento che nel nostro caso $n = 250$, si ottiene che è necessario invocare la procedura di calcolo del percorso minimo (già di per sé piuttosto onerosa) per $250^4 \approx 4 \cdot 10^9$ volte.

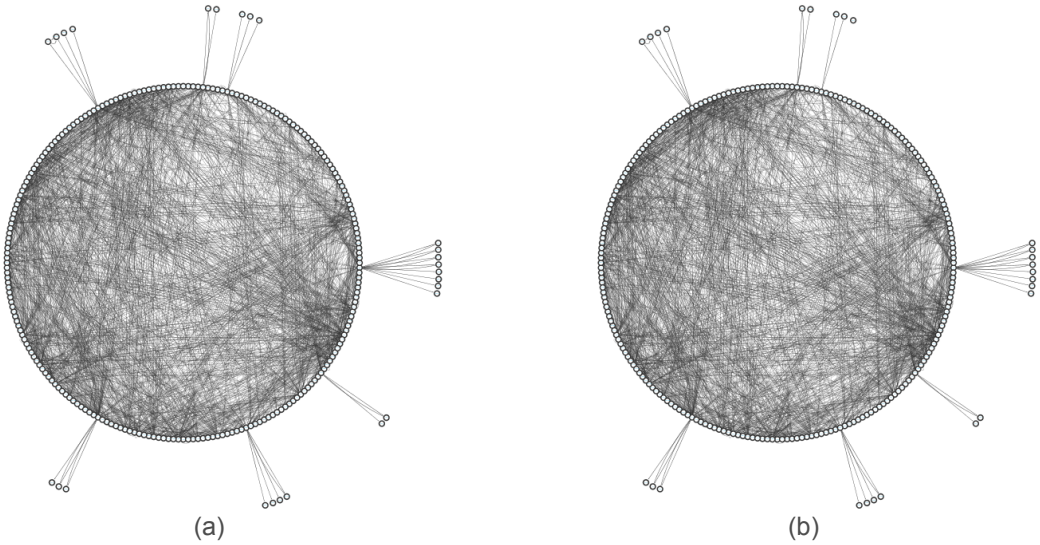

Figura 3: (a) Grafo ottenuto in seguito al *crawling* del dataset. (b) Risultato del procedimento di *clustering* sul grafo del dataset. I due cluster sono composti dall'unico nodo disconnesso (in alto a destra) e dal resto del grafo.

2.3.1 Dimostrazione di correttezza

Consideriamo dapprima un grafo indiretto di dimensione n, in cui $n-1$ nodi formano un clique e il vertice restante è connesso ad uno solo dei nodi del clique (fig.4).

Possiamo dimostrare che l'arco che connette il nodo esterno ad un nodo del clique ha una centralità maggiore degli archi del clique. Detti V e E rispettivamente gli insiemi dei nodi e degli archi, chiamiamo C l'insieme dei nodi del clique, E_C l'insieme degli archi del grafo indotto da C, o l'unico nodo che appartiene a $V - C$ e c l'unico nodo connesso ad o. Consideriamo dapprima solo l'insieme C. Dato che in un clique ogni nodo è connesso con tutti gli altri, la distanza massima tra ogni coppia di nodi $u, v \in C$ è pari ad 1, ed equivale alla lunghezza del cammino che li connette, rappresentato banalmente dall'arco (u, v). Inoltre, dal momento che non ammettiamo l'esistenza di multiarchi, tale cammino sarà unico. Possiamo quindi stabilire una biiezione tra l'insieme degli archi $(u, v) \in E_C$ e l'insieme dei cammini minimi tra ogni $u, v \in C$. In particolare, tale biiezione consisterà nell'associare ad ogni arco (u, v) l'unico cammino minimo che passa per tale arco, ossia proprio il cammino minimo che unisce u e v. Possiamo quindi affermare che ogni arco del clique giace su uno ed un solo cammino minimo, per cui la centralità di ogni arco del grafo indotto da C è pari a 1, ossia $centr((u, v)) = 1$ per ogni $(u, v) \in E_C$.

Estendiamo ora il ragionamento all'intero grafo, ossia al grafo indotto da $V = C \cup \{o\}$. Aggiungendo o all'insieme dei nodi (e dunque l'arco (o, c) all'insieme degli archi), si creeranno dei nuovi cammini minimi. Tuttavia, è facile dimostrare che gli

unici camini minimi che verranno ad aggiungersi ai precedenti sono rappresentati dalle nuove coppie di nodi (o, v) con $v \in E_C$: infatti, l'aggiunta del nuovo arco (o, c) non può creare nuovi cammini minimi tra nodi del clique, in quanto, come già spiegato, per ogni $u, v \in C$ può esistere uno ed un solo cammino minimo, ossia l'arco (u, v). Per aggiornare la centralità degli archi, quindi, sarà sufficiente tenere conto dei nuovi cammini tra ogni nodo e o: in particolare, tali cammini passeranno necessariamente per c, dal momento che questo è l'unico nodo connesso ad o, e l'unico percorso più breve che unisce ogni nodo $v \in C$ a c è rappresentato dall'arco (v, c). Dunque, per ogni nodo in $C - \{c\}$, l'unico cammino minimo che permette di arrivare ad o è rappresentato dal cammino v, c, o. Da ciò segue banalmente che la centralità di tutti gli archi (v, c) aumenterà di 1, mentre resterà invariata la centralità degli archi adiacenti a nodi diversi da c. Infine, per quanto riguarda l'arco (o, c), questo avrà ovviamente centralità pari al numero dei nodi rimanenti, in quanto tale arco giace su tutti e soli i cammini che connettono o con gli altri nodi. In definitiva, quindi:

$$centr((u,v)) = \begin{cases} |C| = |V| - 1 & \text{se } (u = o) \wedge (v = c) \\ 2 & \text{se } (u = c) \wedge (v \neq o) \\ 1 & \text{se } u, v \notin \{o, c\} \end{cases}$$

La situazione, tuttavia, diventa più complicata nel caso in cui i nodi esterni al clique siano più di uno. Prendendo in considerazione un grafo di questo tipo (fig.5), si può facilmente verificare che la centralità di ogni arco è la seguente:

$$centr((u,v)) = \begin{cases} |V| - 1 & \text{se } (u = c_i) \wedge (v \in V_{c_i}) \\ (n_{c_i} + 1) \cdot (n_{c_j} + 1) & \text{se } (u, v) = (c_i, c_j) \\ n_{c_i} + 1 & \text{se } (u = c_i) \wedge (v \in C - \{c_i\}) \\ 1 & \text{altrimenti} \end{cases}$$

Dal momento che per il grafo che si vuole approssimare vale $n \gg n_{c_i}$ per ogni i, vale anche che $(n - 1)$ è sempre maggiore di qualsiasi prodotto $n_{c_i} \cdot n_{c_j}$, e dunque ci aspettiamo che gli archi che connettono nodi esterni a nodi del clique abbiano centralità maggiore. È necessario, quindi, mantenere questi rapporti tra n e i vari n_{c_i} affinché l'approssimazione rispecchi il problema originale. Dunque, dal momento che nel grafo da approssimare il massimo prodotto tra due n_{c_i} è pari a 32, per rispettare la condizione $n - 1 > n_{c_i} \cdot n_{c_j}$ il clique che dovremo creare dovrà contenere almeno 34 nodi.

Quanto finora spiegato, però, vale per grafi indiretti: tuttavia, dal momento che un grafo indiretto può essere visto come un grafo diretto in cui per ogni arco (u, v) esiste un arco (v, u), la dimostrazione sopra riportata può essere facilmente adattata al nostro caso. In particolare, per quanto riguarda il clique all'interno del grafo di approssimazione, essa continua a valere senza bisogno di modifiche, proprio perché esso rispetta la proprietà appena illustrata. Al contrario, osservando il grafo, si nota

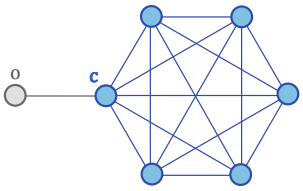

Figura 4

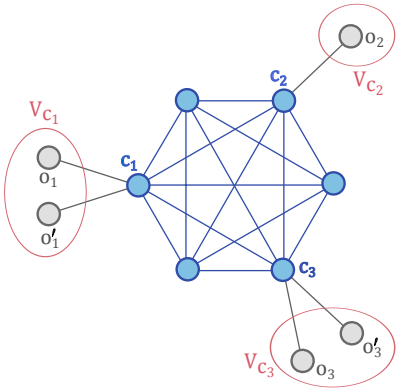

Figura 5

facilmente che gli archi che connettono il clique con i nodi esterni non rispettano tale proprietà: è evidente, inoltre, che la maggior parte di essi siano diretti dal clique verso l'esterno. Tale asimmetria fa sì che alcune coppie di nodi non siano connesse (in particolare tutte le coppie che hanno solo archi entranti). Questo si risolve in una maggior centralità degli archi diretti dall'esterno verso il clique: questi, infatti, saranno gli unici archi che non subiranno un abbassamento della loro centralità, visto che continueranno a giacere sullo stesso numero di cammini minimi: questo accade perché questi archi permettono di raggiungere il clique, e da qui è sempre possibile raggiungere tutti gli altri nodi. Al contrario, gli archi (u, v) dal clique verso l'esterno perderanno tutti i cammini minimi che connettevano un nodo che ora non ha più archi uscenti con v. È evidente, a questo punto, che eseguendo il clustering in base alla centralità degli archi, i primi ad essere eliminati saranno proprio gli archi (u, v) con $u \notin C$ e $v \in C$. Una volta eliminati tutti questi archi, i rapporti dei valori di centralità torneranno ad essere proprio quelli illustrati nella dimostrazione precedente, dal momento che gli archi restanti avranno perso lo stesso numero di cammini minimi giacenti su di essi. Dunque, per come abbiamo costruito il nostro grafo di approssimazione, ci aspetteremo che il prossimo arco ad essere eliminato sia un arco di tipo (u, v) con $u \in C$ e $v \notin C$. Effettivamente, eseguendo l'algoritmo, questo è proprio ciò che succede. In questo modo, quindi, siamo riusciti a suddividere il nostro grafo di approssimazione in 2 cluster, di cui il primo corrisponde ad un solo nodo esterno al clique, e il secondo coincide con il resto del grafo. Sapendo questo, per riportare il risultato ottenuto al grafo originale, basterà disconnettere il nodo del primo cluster dal resto del grafo, avendo così la certezza di aver disconnesso anche in questo caso il grafo e ottenendo, così, i 2 cluster richiesti.

2.3.2 Dettagli implementativi

È opportuno, infine, riportare alcune precisazioni riguardo all'implementazione dell'algoritmo. Innanzitutto, la prima osservazione che si potrebbe fare analizzando

la procedura descritta fin qui è che i cammini minimi non dipendono, ovviamente, dall'arco di cui si sta calcolando la centralità: sarebbe inutile, quindi, ripetere per ogni arco la ricerca dei percorsi più brevi tra ogni coppia di nodi, e proprio per questo motivo, nell'implementazione proposta, tutti gli *shortest path* vengono calcolati nella fase iniziale di ogni round e vengono salvati in una struttura dati. Il metodo che calcola la centralità di un arco, dunque, dovrà semplicemente verificare quanti dei cammini contenuti in tale struttura contengono l'arco preso in considerazione.

Un ulteriore accorgimento che è interessante riportare, infine, è di natura più teorica e prende spunto da alcune deduzioni ricavate durante la dimostrazione di correttezza appena discussa. In particolare, si è visto come la centralità di ogni arco interno al clique e con estremi non connessi con nodi esterni sia costante e pari a 1: sapendo ciò, è stato possibile evitare di calcolare la centralità di questi archi, riducendo in maniera drastica il tempo di esecuzione dell'algoritmo. Dal punto di vista implementativo, questo è stato fatto semplicemente creando una lista contenente tutti i nodi del clique che non avevano archi uscenti dal clique stesso e, per ogni arco preso in considerazione, controllando che i suoi estremi non appartenessero a tale lista: in caso contrario, il valore della centralità dell'arco veniva posto ad 1 senza bisogno di ulteriori calcoli.

3 Task 3

3.1 Individuazione degli utenti-chiave

Per individuare un insieme di utenti particolarmente rilevanti all'interno del dataset si potrebbero usare diverse strategie, che variano sia in base alla complessità che al risultato che si vuole ottenere. Un possibile approccio potrebbe essere quello utilizzato per risolvere il primo task, ossia estrarre gli utenti più citati all'interno dei tweet; un altro metodo, invece, potrebbe essere quello di scegliere gli utenti che vantano il maggior numero di retweet. Sebbene entrambe queste strategie (così come molte altre) potrebbero sembrare ugualmente valide, in questo caso esse si rivelano essere non ottimali se prendiamo in considerazione l'utilizzo che si vorrà fare della lista di utenti ottenuta: infatti, dal momento che i punti successivi richiederanno di indicizzare tutti i tweet pubblicati dagli utenti scelti, avrebbe poco senso effettuare questa operazione su utenti che hanno pubblicato un numero esiguo di tweet. Nella soluzione proposta, dunque, la lista di utenti di rilievo è stata ottenuta semplicemente estraendo gli utenti che hanno pubblicato il maggior numero di tweet: sebbene possa sembrare piuttosto naïve, in realtà questo approccio rappresenta un valido compromesso tra i due obiettivi che si vogliono raggiungere, ossia ottenere una lista di utenti che risultino essere "importanti" e che, allo stesso tempo, abbiano pubblicato abbastanza tweet da renderne sensata l'operazione di indicizzazione. Proprio questo aspetto rende questa strategia preferibile alle due accennate precedentemente: ad esempio, sebbene sia garantito che se un utente vanta un alto numero di retweet allora ha una

certa rilevanza, non è però assicurato che i suoi tweet siano abbastanza numerosi da giustificarne l'indicizzazione.

Utilizzando il metodo appena descritto, quindi, sono stati estratti i 100 utenti più importanti: per motivi di spazio, i nomi non sono qui riportati, ma è comunque possibile consultarli all'interno del file `mfuMap.txt` contenuto nella cartella `out`.

3.2 Indicizzazione dei tweet e creazione del diagramma temporale

Dopo aver classificato manualmente gli utenti individuati nel punto precedente, suddividendoli nelle categorie *politici*, *media* e *cittadini comuni*, si è proceduto indicizzandone i tweet con l'ausilio di Lucene. Poiché era richiesto di ricavare le 5 parole più frequenti in ognuno di questi gruppi, sono state create 3 directory di Lucene (una per ogni gruppo), all'interno delle quali venivano creati i vari documenti: in particolare, ogni documento rappresentava un tweet, e la scelta della directory in cui salvare tale documento dipendeva ovviamente dall'autore del tweet, o meglio dalla sua classe di appartenenza.

Una volta compiuta questa operazione, si è potuto procedere con l'estrazione delle 5 parole più frequenti in ogni gruppo: tuttavia, dal momento che alcuni termini frequenti erano comuni a diversi gruppi, la lista finale è risultata contenere solo 11 parole distinte. A partire da queste, infine, si è analizzato l'intero dataset, ed è stato creato il diagramma temporale riportato in fig.6: questo rappresenta il numero di citazioni di ognuna delle 11 parole in ogni intervallo di tempo di un giorno.

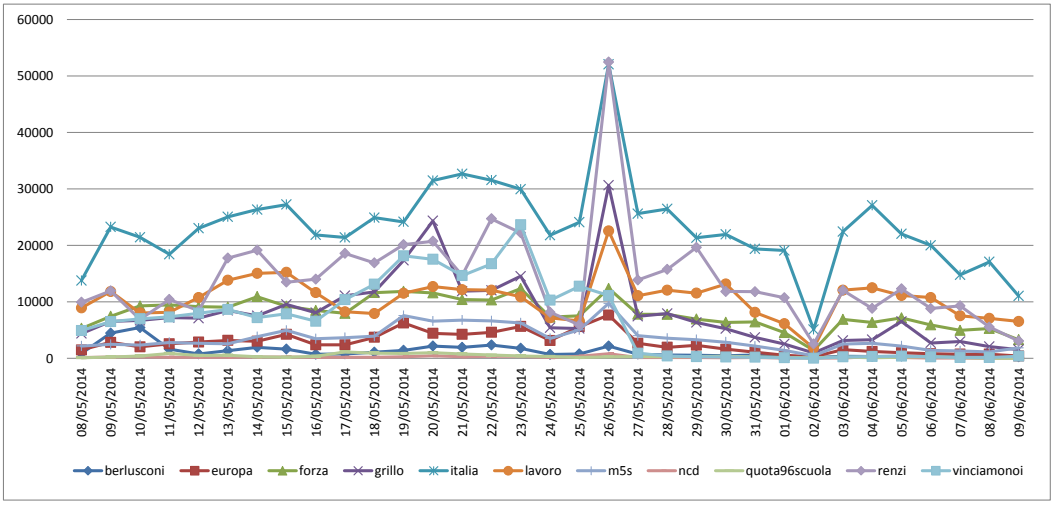

Figura 6: Diagramma temporale delle 11 parole più frequenti. È interessante notare come nei giorni delle elezioni si verifichi un picco nei termini corrispondenti ai nomi dei politici e dei partiti, mentre nei giorni successivi si verifichi un calo nelle espressioni usate come *slogan*. Si noti inoltre il calo in corrispondenza del 2 giugno, probabilmente dovuto alla maggiore frequenza di tweet riguardanti la festa della Repubblica.