Alma Mater Studiorum · Università di Bologna

facoltà di scienze matematiche, fisiche e naturali
Corso di Laurea Triennale in Informatica

Conversation Retrieval su Twitter

Tesi di Laurea in Basi di Dati e Sistemi Informativi

Relatore:
Chiar.mo Prof.
Danilo Montesi

Presentata da:
Gabriele Nunziante

II Sessione
Anno Accademico 2009/2010

Indice

Introduzione

Con l'avvento del Web 2.0 i contenuti generati dagli utenti hanno visto un aumento considerevole, in particolare quelli realizzati mediante l'uso di *Social Network* (SN). Tramite i servizi da essi forniti, le persone hanno trovato un modo per rimanere costantemente in contatto, sfruttando le peculiarità di ogni sistema per condividere, discutere e informare gli altri utenti di ciò che sta accadendo intorno a loro. L'aumento di queste informazioni ha reso difficile il recupero di contenuti presentati non più come semplici pagine web ma dati con una precisa struttura. Come è possibile recuperare informazioni prodotte dai vari *social network*, rendendo la ricerca simile a quanto siamo oggi abituati con i motori di ricerca sul web, ma senza perdere le strutture proprie di ogni SN?

Scopo di questa tesi è presentare una possibile risoluzione del problema in un preciso contesto, ovvero quello di uno dei *social network* di *microblogging* fra i più frequentati del momento: Twitter. Attivo da quasi 5 anni e in continua ascesa, Twitter si presenta come un SN dall'aspetto semplice e con un solo compito: condividere brevi messaggi di testo. Tuttavia la possibilità per gli utenti di interagire fra loro, rispondendo a messaggi altrui e instaurando conversazioni, va a formare una realtà tanto complessa quanto interessante. L'utilizzo del SN per diramare notizie anche di importanza mondiale è un ulteriore incentivo a trovare un meccanismo di recupero delle informazioni che non si limiti alla pura ricerca sul testo dei messaggi. La tesi in oggetto si occuperà quindi di analizzare Twitter, trovare un meccanismo per recuperare le conversazioni e infine realizzare un prototipo funzionante di ricerca su di esse.

Nel primo capitolo verrà presentato il SN in questione, le API messe a disposizione per gli sviluppatori e cosa viene inteso per conversazione all'interno di Twitter. Nel secondo si affronterà uno studio preliminare su come realizzare una base di dati che ci permetta un test su un campione significativo delle informazioni presenti nel SN. Nel terzo capitolo verrà affrontato il tema del ranking delle conversazioni, studiando ogni particolare indice per determinarne l'utilità nel riordino dei risultati di una ricerca. Nel quarto capitolo verrà presentata nella sua interezza una demo basata sui dati

campione, mentre nel quinto saranno presentati i risultati finali e possibili indicazioni per ulteriori sviluppi.

Capitolo 1

Caso di studio: Twitter

Viene introdotto in questo capitolo il *social network* oggetto dei nostri studi, ovvero Twitter. Nel primo paragrafo verrà presentata un'analisi delle sue funzionalità e degli scopi del sistema, nel secondo una panoramica sugli strumenti a disposizione degli sviluppatori e infine il modello di conversazione, sviluppato su Twitter, alla base della presente ricerca.

1.1 Introduzione a Twitter

Twitter è un *social network* gratuito che si occupa di *microblogging*, ideato dallo statunitense Jack Dorsey e sviluppato dalla Obvious Corporation di San Francisco. Il servizio offerto agli iscritti è l'inserimento di messaggi, chiamati in gergo *tweet*, composti da un massimo di 140 caratteri.

Fin dalla sua creazione e messa in rete nel marzo 2006, Twitter ha assunto un ruolo di rilievo all'interno dell'insieme dei *social network*, raggiungendo in quattro anni oltre 100 milioni di utenti iscritti. Giornalmente vengono inviati in media 55 milioni di messaggi [1], con punte che sfiorano i 3 mila tweet al secondo in casi eccezionali, solitamente legati a eventi di cronaca di carattere internazionale (disastri ambientali, attentati, eventi e ricorrenze religiose o civili), ma anche eventi sportivi e politici. Twitter ha assunto in questo caso anche un ruolo di rilievo per la diffusione di notizie: ne sono esempi la cosiddetta *Iran's Twitter revolution*, avvenuta durante le elezioni politiche in Iran nel giugno 2009, l'attacco terroristico a Mumbai nel 2008, il terremoto ad Haiti nel gennaio 2010 o, rimanendo sul suolo italiano, il terremoto in Abruzzo dell'aprile 2009, durante il quale gli utenti di Twitter segnalarono prima dei media tradizionali quanto stava accadendo.

Di notevole importanza su Twitter è anche l'iscrizione al servizio da parte di numerosi personaggi celebri, fra cui è possibile trovare, oltre a innumerevoli personalità dello spettacolo e del mondo della musica, politici, ma anche testate giornalistiche, aziende di importanza mondiale, associazioni e molti che hanno trovato nel servizio offerto da questo *social network* un metodo

veloce per interagire con il resto del mondo.

Oltre che semplice testo, all'interno dei tweet è possibile inserire delle parole chiave, dette *hashtag* e precedute dal carattere #, e link ad altri siti, solitamente abbreviati tramite servizi di *URL shorting*. I messaggi inseriti dagli iscritti vengono di default resi visibili a chiunque, sia esso iscritto o meno al servizio, mentre è possibile rendere i propri *tweet* privati per fare in modo che possano essere letti solamente da persone autorizzate. L'inserimento dei messaggi è reso possibile non solo tramite il sito del *social network*, ma anche da una serie di applicazioni esterne e, limitatamente ad alcuni paesi, tramite SMS. La possibilità di inviare messaggi tramite diversi dispositivi e applicazioni è uno dei punti di forza del *social network*: il 60% dei messaggi vengono infatti inseriti su Twitter tramite applicazioni realizzate da terze parti, mentre il 37% degli utenti iscritti inserisce i propri tweet tramite cellulare [1], [2].

Gli iscritti al servizio hanno la possibilità di seguire altri utenti registrati: essi assumono in questo caso il nome di *follower* e hanno la possibilità di visualizzare nella propria *home page* i messaggi inseriti da tali utenti. È inoltre possibile seguire liste di utenti, ovvero liste create da altri iscritti in cui è incluso un numero variabile di utenti. Altra *feature* importante per il servizio, di cui si tratterà in seguito, è la possibilità di rispondere ai messaggi di qualsiasi altro iscritto, realizzando così conversazioni *online*. Un iscritto può inoltre ripostare un messaggio di un altro utente, in modo che esso sia visibile anche a tutti i propri *follower*. Questa tecnica è definita *retweet* e viene segnalata nei messaggi anteponendo i caratteri RT al testo originale.

Come descritto da Ryan Kelly [3], i contenuti inseriti dagli utenti su Twitter sono riconducibili principalmente a sei categorie, qui di seguito ordinate in modo decrescente per frequenza: status personali, conversazioni, retweet, self-promotion, spam e news. È bene notare che le conversazioni e gli status personali, calcolati su un campione di tweet raccolti dal SN, raggiungono sommati quasi il 90% totale dei messaggi, mentre il 37.55% del totale è composto da messaggi in risposta ad altri tweet. Per quanto riguarda lo spam e i messaggi di self-promotion (ovvero tweet a solo scopo pubblicitario inseriti da aziende) essi sono limitati al 9.6%. Da queste statistiche è possibile dedurre come Twitter sia divenuto uno dei mezzi più efficaci per condividere le proprie esperienze e di come gli utenti lo utilizzino anche per comunicare fra di loro, riconducendosi all'idea originale di Jack Dorsey di realizzare un servizio simile agli SMS, ma applicato a gruppi di persone e disponibile sul web.

Per questi motivi lo studio di Twitter si è rivelato fin dal principio di enorme interesse, essendo esso un *social network* molto frequentato, dinamico e in cui gli utenti, tramite i loro tweet, contribuiscono a tenere informato

il pubblico di quanto stia accadendo attorno ad essi. Nei 140 caratteri di un messaggio vengono raccontate esperienze di vita tanto personali quanto relative al mondo che circonda gli utilizzatori, mentre la possibilità di includere link ad altri siti, nonchè immagini e video, è un'ulteriore metodo per poter divulgare quanto vi è di più importante nel *web*.

1.2 API per sviluppatori

Twitter fornisce per sviluppatori privati o facenti parte di organizzazioni la possibilità di realizzare applicazioni che interagiscano con il *social network* stesso. È possibile suddividere le applicazioni realizzabili in due grandi gruppi: quelle che permettono l'inserimento di informazioni e quelle che ne permettono il recupero. Nel primo tipo rientrano tutte quelle applicazioni, *web-based* o meno, che danno la possibilità a un iscritto di poter inserire un nuovo messaggio, nonché, in modo generale, di interagire in qualsiasi modo con gli altri utenti. Nel secondo tipo rientrano invece le applicazioni che permettono l'estrazione dei contenuti già presenti su Twitter: è possibile, ad esempio, richiedere informazioni relative a utenti o tweet singoli, la lista degli ultimi messaggi di un utente (ovvero la sua *timeline*) e molto altro. È possibile naturalmente scrivere applicazioni che rientrino in entrambe le categorie, arrivando a fornire quindi un servizio praticamente identico allo stesso Twitter. Vi è inoltre la possibilità di realizzare applicazioni di ricerca sui contenuti inseriti all'interno del *social network*, sempre utilizzando le API messe a disposizione.

Per lo svolgimento di questa tesi si è posta particolare attenzione sulla modalità di estrazione delle informazioni già presenti su Twitter ed è quindi in questo campo che si procederà nello studio. La possibilità di recuperare dati dal *social network* è resa possibile tramite una serie di API messe liberamente a disposizione degli sviluppatori che permettono di fatto l'estrazione di tutti i dati relativi a utenti e messaggi presenti su Twitter. Per l'utilizzo di esse non è richiesta la registrazione al sito, sebbene, come verrà spiegato in seguito, quest'ultima sia necessaria per superare alcune limitazioni. Le API disponibili, la cui documentazione è visionabile in [4], permettono tramite richieste HTTP l'estrazione delle informazioni in diversi formati, principalmente xml, json, rss e atom. In particolare la nostra attenzione è stata posta sui metodi che ci avrebbero permesso di recuperare informazioni sugli utenti e sulle loro *timeline*, nonché le connessioni fra i tweet postati (ad esempio le risposte ad un determinato messaggio).

Come precedentemente accennato, l'utilizzo di queste API non è soggetto a registrazione da parte degli sviluppatori. Tuttavia la possibilità di effettuare richieste è sottoposta ad alcune limitazioni di carattere quantitativo:

in particolare, per utenti "anonimi", è possibile compiere 150 richieste per ora per indirizzo IP. Registrando la propria applicazione ed effettuando le richieste con un metodo denominato OAuth, allo sviluppatore viene data la possibilità di effettuare fino a 350 richieste per ora. Vi è inoltre la possibilità di fare domanda per entrare in una *white list*, ricevendo così l'opportunità di effettuare fino a 20 mila richieste per ora. Questa possibilità non viene tipicamente concessa ai ricercatori ma solamente a sviluppatori di applicazioni con necessità particolari: non vi è stato quindi modo di poterne usufruire per lo sviluppo dell'applicazione in oggetto.

La costruzione di una richiesta utilizzando le API è di facile e immediata comprensione. Si prenda come esempio la richiesta per l'estrazione delle informazioni di un utente. Come descritto in [4], essa deve rispettare il seguente standard:

```
http://api.twitter.com/[vers]/users/show.[format]?user_id=[ID]
```

Modificando opportunamente i parametri in essa presenti (*vers*, *format* e *ID*) è possibile effettuare una richiesta delle informazioni, ad esempio, dell'utente contrassegnato con user_id 12 e nel formato xml. Il parametro *vers* è al momento limitato al valore 1 e presente per versioni future delle API in oggetto. La richiesta diventa dunque:

```
http://api.twitter.com/1/users/show.xml?user_id=12
```

La risposta restituita, in questo esempio nel formato xml, sarà simile alla seguente:

```
<user>
   <id>12</id>
   <name>Jack Dorsey</name>
   <screen_name>jack</screen_name>
   <location>NYC & San Francisco</location>
   <description>
    Creator, Co-founder and Chairman of Twitter; CEO of Square.
   </description>
   <profile_image_url>
    http://a3.twimg.com/profile_images/113072/image_normal.jpg
   </profile_image_url>
   <url/>
   <protected>false</protected>
   <followers_count>1593743</followers_count>
   ...
</user>
```

All'interno dell'xml risultante è dunque possibile recuperare tutte le informazioni necessarie dell'utente, quali nome, posizione geografica, una breve

descrizione, l'url della sua immagine del profilo e così via. Similmente avverrà per le altre richieste, quali l'estrazione di informazioni riguardanti un singolo messaggio o la *timeline* di un utente.

Come conclusione di questo paragrafo è bene segnalare l'esistenza di librerie per diversi linguaggi, quali ad esempio C, C++, Java, PHP, Ruby e Flash, che permettono allo sviluppatore di concentrarsi sugli scopi della sua applicazione piuttosto che sulla scrittura di codice relativo all'utilizzo delle API. Nello sviluppo della presente tesi è stata in particolare utilizzata una libreria non ufficiale per Java denominata Twitter4J [5], la quale mette a disposizione del programmatore tutti i metodi necessari per effettuare le richieste tramite le API di Twitter.

1.3 Modello di conversazione su Twitter

Dopo questa panoramica sulle possibilità di estrazione delle informazioni relative a utenti e tweet, è bene ora concentrarsi sull'argomento affrontato nella presente tesi. Introduciamo quindi il modello di conversazione in uso su Twitter.

Possiamo definire generalmente una conversazione come l'interagire in modo orale di due o più persone su argomenti comuni o correlati fra loro. Nella vita reale una conversazione vede un inizio nel momento in cui una delle persone partecipanti dà origine alla discussione stessa. Con l'avanzare del tempo altre persone prenderanno a loro volta la parola, aggiungendo alla discussione nuovi argomenti. Solitamente una conversazione, indipendentemente dal numero di partecipanti, vede ognuno di essi parlare non in modo simultaneo, ma alternato.

Occorre considerare inoltre l'inserimento di un nuovo partecipante nel momento in cui la conversazione ha già avuto inizio. In tal caso egli non avrà modo di sapere cosa è stato detto prima del suo arrivo, ma solamente ciò che viene aggiunto dal momento in cui ha preso parte alla conversazione. Oltre ai partecipanti possono essere inoltre presenti anche dei semplici ascoltatori che, con l'avanzare del tempo, possono a loro volta interagire attivamente o rimanere per tutta la durata semplici presenze.

Una conversazione tramite *social network* conserva solamente alcuni di questi aspetti. Come nelle conversazioni reali, la discussione è iniziata da una certa persona in un determinato istante. A questa altre persone possono rispondere, iniziando così una vera e propria conversazione. Le differenze possono però già nascere nel momento in cui essa è iniziata: ad esempio, più di una persona può rispondere allo stesso messaggio, che esso sia l'iniziale o uno qualsiasi all'interno della discussione, portando la stessa a suddividersi

in più parti. Queste conversazioni possono anche non condividere più, con il passare del tempo, i partecipanti, gli ascoltatori e perfino gli argomenti trattati. Una conversazione *on-line* ha un'altra sostanziale differenza, ovvero la possibilità per un nuovo partecipante di diventare parte attiva della discussione anche dopo l'inizio della stessa, ma conoscendo comunque i messaggi precedentemente inseriti. In questo caso si dice che lo *scope* del nuovo partecipante comprende l'intera conversazione.

Definite le differenze fra una discussione reale e una *online*, spostiamo ora l'attenzione sulle conversazioni ottenibili su Twitter. Come precedentemente descritto per i *social network* in generale, anche in questo caso il modello presenta caratteristiche differenti dalle conversazioni reali. In Figura 1.1 è possibile vedere il modello di una discussione ideale su Twitter. Il quadrato rappresenta il messaggio iniziale che dà origine alla discussione, mentre i tondi identificano le risposte ai messaggi a cui sono connessi tramite una freccia. Come è possibile vedere, le discussioni realizzabili su Twitter assumono un aspetto ramificato. Questo aspetto non è comune all'interno degli attuali *social network*: esempi celebri come Facebook, Friendfeed, Flickr o MySpace utilizzano tutti sistemi di conversazione non nidificati ma piatti, assimilabili a commenti in risposta a un messaggio principale. Nel caso di Twitter occorre dunque studiare un modello più complesso.

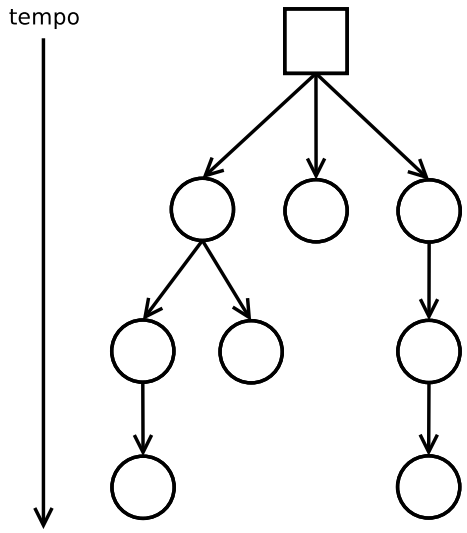

Fig. 1.1: Modello di conversazione di Twitter.

Per lo studio di questo modello introduciamo ora una serie di definizioni che verranno successivamente riprese nel caso reale.

Definizione 1.1 (Messaggio)**.** Definiamo come messaggio m una serie di caratteri inseriti da un determinato utente u in un preciso istante di tempo t.

Definizione 1.2 (Conversazione)**.** Definiamo come conversazione c un grafo orientato aciclico i cui nodi sono i messaggi $m_1, ..., m_n$ facenti parte di c. Le connessioni fra $m_1, ..., m_n$ indicano il rapporto di risposta fra questi messaggi. Una conversazione è caratterizzata da un istante di inizio *t_start* e uno di fine *t_end*, equivalenti rispettivamente al tempo t_1 del messaggio m_1 iniziale della conversazione e al tempo t_n del messaggio m_n ultima risposta alla conversazione.

Nel caso di Twitter, i messaggi sono connessi fra loro con una relazione rivolta dal figlio verso il padre. In questo modo è possibile dunque realizzare uno schema ramificato come presentato nella Figura 1.1.

Date le precedenti definizioni 1.1 e 1.2, possiamo introdurne una relativa agli utenti:

Definizione 1.3 (Partecipanti e lettori)**.** Viene definito partecipante alla conversazione c un utente u che inserisce almeno un messaggio m all'interno di c. Viene definito lettore di c un utente l che legge almeno un messaggio m appartenente a c. Un partecipante a c, fino all'istante in cui non inserisce un messaggio m nella conversazione, viene definito lettore della stessa.

Nel caso di Twitter, possiamo considerare come lettori di una conversazione tutti i *follower* di un partecipante alla discussione stessa. Per giustificare ciò, consideriamo che nel momento in cui un utente partecipa a una conversazione il suo messaggio m apparirà nella sua *timeline*. Tale *timeline* è direttamente leggibile dai *follower* del partecipante, il che farà di essi dei lettori della discussione.

Oltre a ciò è bene considerare che, nel caso la *timeline* di un utente sia pubblica, essa potrà essere letta non solo dai suoi *follower*, ma anche da tutti coloro che avranno modo di visualizzare la sua pagina. Tuttavia, non essendo possibile stimare il numero di visite di un certo messaggio, tali lettori non verranno considerati per i fini della nostra ricerca. Il numero di *follower* per un utente può essere comunque considerato un buon indice del numero di visualizzazioni dei suoi messaggi.

Nella definizione di messaggio e conversazione viene introdotto il fattore tempo. Questa caratteristica, propria sia delle conversazioni reali sia di quelle presenti su *social network*, assume nel caso di quest'ultima realtà un aspetto importante di cui si terrà conto successivamente. Per il momento è bene notare che, a differenza di una conversazione reale il cui svolgersi è solitamente limitato nel tempo, nel caso dei *social network*, e quindi di Twitter,

lo svolgimento di una discussione pu protrarsi nel tempo anche per lunghi periodi. È inoltre possibile che ad una conversazione che consideriamo già terminata sia inserito in risposta un nuovo messaggio, che potrà eventualmente far ripartire la conversazione. In tale caso è lecito domandarsi quanto sia importante una risposta inserita dopo un lungo periodo, relativa magari a una conversazione che ha visto diminuire l'interesse nel suo argomento. Analizzando i dati estratti per la realizzazione del sistema è possibile trovare tweet in risposta anche dopo mesi dall'inizio della conversazione e, caso limite, anche dopo un anno e mezzo, sebbene meno dell'1% delle conversazioni si protrae per più di una settimana dal suo inizio. Questo fattore verrà trattato successivamente in modo più approfondito.

Capitolo 2

Analisi del sistema

2.1 Una panoramica sul sistema

Introdotta la realtà di interesse, veniamo ora alla trattazione del sistema nello specifico. Ciò che si vuole realizzare è un sistema capace di recuperare le conversazioni su Twitter e renderle disponibili per ricerche su testo, con la possibilità di organizzare i risultati in base a determinate caratteristiche. Si pone ora l'attenzione sulla prima parte del sistema, ovvero quella dedicata al recupero, la memorizzazione e l'indicizzazione delle conversazioni, mentre si rimanda al capitolo successivo per la descrizione del ranking dei risultati.

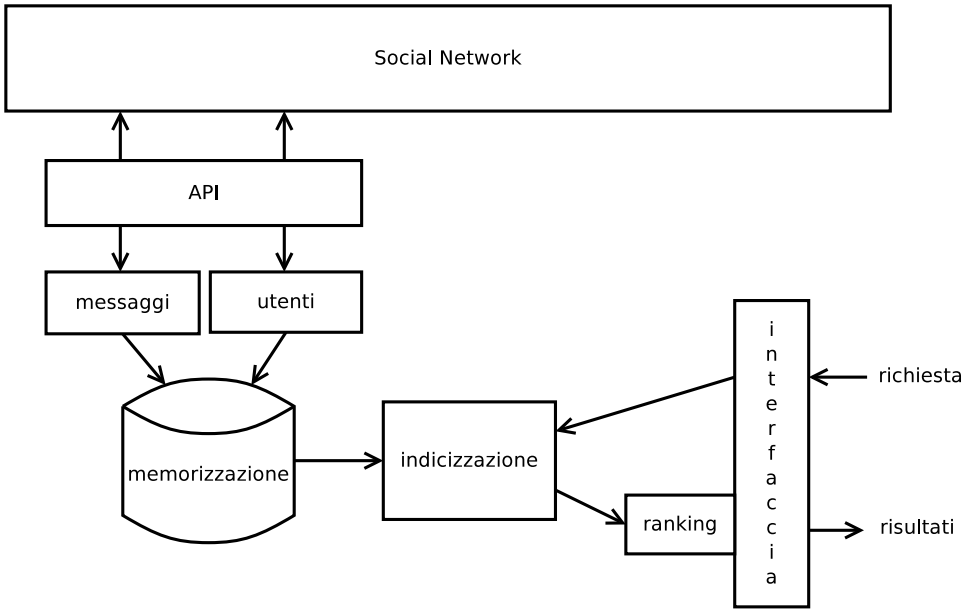

Fig. 2.1: Modello del sistema.

In Figura 2.1 viene indicato il modello del sistema che si andrà a realizzare. Partendo dall'alto, troviamo la rete rappresentante il *social network* (SN) in questione. Tramite le API messe a disposizione per gli sviluppatori è possibile accedere alle informazioni presenti nel SN; in particolare utilizzeremo queste per poter ricavare i dati relativi a messaggi e utenti. Estratte queste informazioni, è necessario pensare a dove memorizzarle: possibili soluzioni sono un *database* o un *file system*. Il passo successivo sarà l'indicizzazione di questi dati, necessaria per avere un veloce accesso alle informazioni. Viene infine fornita una interfaccia che permetta ad un utilizzatore di effettuare l'interrogazione, tramite l'inserimento di una o più parole di ricerca, e la visualizzazione dei risultati ordinati.

Nei seguenti paragrafi verrà trattata in maniera più approfondita ciascuna delle parti in questione.

2.2 Estrazione delle informazioni

Il primo passo necessario al recupero delle conversazioni è lo studio delle API messe a disposizione da Twitter, in maniera tale da poter ricavare un buon metodo per estrarre tutti i dati di cui abbiamo bisogno. In questo paragrafo viene presentata una possibile soluzione e un modello per la ricostruzione di una conversazione, mentre il sistema effettivamente realizzato verrà descritto nel capitolo 4.

Una conversazione su Twitter, come descritto precedentemente, è formata da messaggi postati da più utenti. Identifichiamo quindi le API necessarie al recupero delle informazioni su queste due entità. Come indicato in [4], esse sono:

- statuses/show/:id - restituisce le informazioni su un singolo status (ovvero un tweet) specificato dal parametro ID;
- users/show - restituisce le informazioni di un utente specificato tramite i parametri ID o Screen Name.

Tramite queste due semplici API avremmo già a disposizione tutti gli strumenti per recuperare informazioni su tweet e utenti. Rimane però il problema del come realizzare, su una rete tanto vasta come quella di Twitter, un meccanismo per l'estrazione automatica dei messaggi.

Studiando il servizio è possibile notare come gli identificativi dei tweet siano numeri interi assegnati in modo sequenziale. Un semplice script per l'estrazione dei messaggi potrebbe dunque tener conto di ciò ed estrarre in modo ordinato tutti i tweet disponibili, partendo da un numero e incrementandolo a ogni richiesta. Con questa soluzione risolveremmo i problemi relativi alla ricostruzione delle conversazioni: avremmo infatti sicuramente

la certezza che, trovato un tweet in risposta a un altro, quest'ultimo sia già stato estratto, poichè con id sicuramente inferiore.

Questa soluzione, utilizzata in una fase iniziale, si era rivelata effettivamente un metodo molto semplice e funzionale. Tuttavia tale procedimento ha presentato problemi causati da due caratteristiche: la prima riguardante il numero di tweet estratti, la seconda al cambiamento, avvenuto proprio nell'ultimo mese, del generatore di status id di Twitter.

Come descritto nel capitolo precedente, le interrogazioni tramite API sono soggette ad alcune limitazioni quantitative, pari a 350 ad ora per gli sviluppatori registrati. Per questo motivo, essendo il numero di tweet estratti con questo meccanismo pari a un tweet per ogni singola richiesta, riusciremmo ad estrarre non più di 350 tweet in un'ora. Considerando che il totale dei tweet inseriti su Twitter in questo stesso periodo è pari a 2.3 milioni, è facile intuire come con tale procedimento non riusciremmo mai a raggiungere un numero abbastanza significativo di messaggi.

In seconda analisi, il cambiamento effettuato su Twitter in questo ultimissimo periodo ha portato alla ridefinizione dell'id del tweet. A partire da novembre 2010 esso non è infatti più un numero sequenziale ma un intero basato sull'orario di inserimento del messaggio [6].

Principalmente per questa ultima caratteristica, l'estrazione di dati utilizzando ID creati "artificialmente" non è più ritenuto possibile.

Per risolvere entrambi i problemi, possiamo orientarci su metodi che permettano di estrarre più informazioni possibili con poche richieste e che non siano soggetti al dover "creare" un id. Ci vengono in aiuto per far ciò le API che permettono di estrarre la *timeline* pubblica e la *timeline* di un utente specifico:

- statuses/public_timeline - fornisce gli ultimi 20 status pubblici inseriti su Twitter;

- statuses/user_timeline - fornisce gli ultimi 200 status dell'utente specificato dal parametro ID.

Tramite *public_timeline* è possibile ricavare, oltre ai 20 tweet più recenti, i relativi 20 utenti che li hanno inseriti. Fra queste informazioni è direttamente disponibile l'id utente: questo ci sarà utile per recuperare, tramite *user_timeline*, i suoi ultimi 200 status. Con queste due sole richieste abbiamo dunque ricavato 200 tweet: a differenza del precedente sistema con cui ricavavamo un tweet per ogni singola richiesta, questo metodo risulta dunque ben più performante.

Sorge però il problema della ricostruzione delle conversazioni: trovato infatti in una *user_timeline* un tweet in risposta a un altro, non è detto che quest'ultimo sia già stato estratto. Per risolvere questo problema, prendiamo

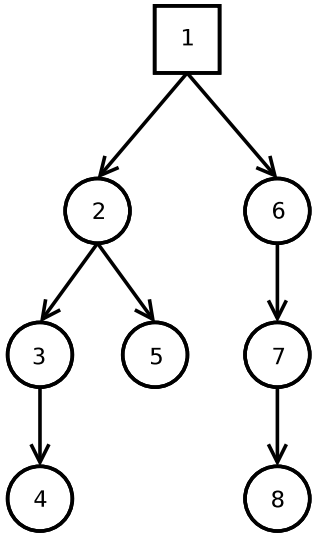

Fig. 2.2: Esempio di conversazione con tweet contrassegnati da relativo ID.

ad esempio la conversazione in Figura 2.2 e poniamo di avere le informazioni relative al tweet contrassegnato da id 4:

```
<status>
   ...
   <id>4</id>
   <text>Example of tweet in reply to another</text>
   ...
   <in_reply_to_status_id>3</in_reply_to_status_id>
   ...
</status>
```

Grazie all'attributo "in_reply_to_status_id" è possibile conoscere l'id del padre del presente tweet. Per recuperarlo utilizzeremo quindi questo id e il metodo *statuses/show* come già descritto precedentemente. Effettuando ricorsivamente questa ricerca, passeremo dal tweet 4 al 3, dal 3 al 2, dal 2 all'1. Quest'ultimo tweet sarà contrassegnato da un valore *in_reply* pari a *-1*, valore di default per messaggi non in risposta: in questo modo sapremo di aver raggiunto il limite superiore e di aver ricostruito l'intero ramo della conversazione.

Vista la loro natura ramificata, non è invece possibile costruire l'albero di una conversazione in modo inverso, ovvero partendo da un nodo e scendendo verso le foglie. Per questo motivo, recuperato un tweet, non possiamo sapere se esso è l'ultimo o meno di una conversazione. È quindi anche possibile che si arrivi a ricostruire solamente un ramo della discussione o solamente una parte di essa. Si pensi ad esempio di avere ottenuto il tweet 7: da esso potremo solamente ricostruire la conversazione in altezza, ricavando

dunque i tweet 6 e 1, ma non l'8. Allo stesso modo, avendo il tweet 4, non riusciremo a ricavare né il ramo composto dal 5 né quello composto da 6-7-8.

Questi limiti sono dovuti principalmente alla struttura delle conversazioni su Twitter e, non esistendo API che ci permettano di ricavare i messaggi in risposta a un determinato tweet, non risulta possibile realizzare un sistema migliore. Tuttavia il metodo presentato, con alcune modifiche per aumentarne l'efficienza, è risultato essere quello con un miglior rapporto tweet estratti/richieste e quindi utilizzabile nella fase pratica.

2.3 Memorizzazione dei dati

Affrontata la parte relativa all'estrazione delle conversazioni, è opportuno ora considerare il loro salvataggio in strutture dati che ci permettano un facile accesso in fase di ricerca.

2.3.1 Specifiche della realtà di interesse

La realtà studiata comprende due entità principali: gli utenti e i messaggi. Le informazioni forniteci dalle API di Twitter sono in numero eccessivo rispetto ai nostri scopi: elenchiamo dunque solamente le informazioni necessarie per la realizzazione del sistema.

Gli utenti sono caratterizzati da un id univoco, un nome utente privo di caratteri speciali, un nome, una immagine e un numero di *follower*. I messaggi vengono inseriti da un utente e sono caratterizzati da un id univoco, un testo di massimo 140 caratteri, un orario, un eventuale id di un altro messaggio a cui il presente è in risposta e il numero dei suoi retweet.

Si vuole inoltre tenere traccia delle conversazioni formate dai messaggi. Le conversazioni sono caratterizzate da un id di partenza (uguale all'id del messaggio da cui è iniziata la discussione), il numero di risposte, un orario di inizio, uno di fine e la densità della conversazione.

2.3.2 Schema relazionale

Introdotta la realtà di interesse, possiamo rappresentarla con il seguente schema relazionale:

tweet (<u>id</u>, id_user, text, time, reply_to, origin, n_retweet)
user (<u>id</u>, name, screen_name, image, followers, n_story, n_reply, last_update)
story (<u>id_start</u>, reply, text, time_first_post, time_last_post, n_retweet, time_density)

Sono stati introdotti alcuni nuovi campi per rappresentare le connessioni fra le entità e/o aggiungere informazioni ritenute necessarie in fase applicativa. In particolare:

- il campo *origin* presente nell'entità tweet identifica il tweet che ha dato inizio all'intera conversazione. Esso sarà uguale all'id del tweet se quest'ultimo non fa parte di una conversazione o è egli stesso l'inizio della conversazione; sarà uguale a reply_to se il padre del tweet è anche il padre della conversazione; sarà diverso da entrambi se il tweet ha profondità maggiore di 1 nell'albero della conversazione;

- i campi *n_story* e *n_reply* presenti nell'entità user sono stati introdotti per tenere traccia rispettivamente del numero di conversazioni iniziate dall'utente e del numero complessivo delle risposte alle sue conversazioni. Questi parametri verranno utilizzati per motivi di ranking, come descritto nel capitolo successivo;

- il campo *last_update* presente nell'entità user viene utilizzato dall'applicazione per segnalare quando è stato realizzato l'ultimo aggiornamento delle informazioni dell'utente. In questa maniera, se la richiesta è molto recente, può essere evitata una ulteriore estrazione delle informazioni;

- il campo *text* presente nell'entità story comprende il testo di tutti i tweet facenti parte della conversazione. Questo campo è introdotto per motivi di indicizzazione;

- il campo *time_density* presente nell'entità story indica la densità della conversazione. Questo parametro verrà utilizzato per motivi di ranking, come descritto nel capitolo successivo.

2.3.3 Memorizzazione fisica

Introdotto lo schema relazionale, è bene ora scegliere dove e come memorizzare fisicamente queste strutture. Le possibilità iniziali sono essenzialmente due: affidarsi alla memorizzazione su semplici file o l'utilizzo di un database. Per comprendere quale dei due faccia al caso nostro, si pone l'attenzione su due fattori principali: il carico dati atteso e il numero di interrogazioni necessarie durante lo sviluppo dell'applicazione.

Per realizzare un sistema di ricerca su Twitter, sapendo in partenza di non poter simulare l'intera rete del SN viste le sue dimensioni e gli strumenti a disposizione, è necessario raccogliere un quantitativo almeno indicativo della realtà di interesse. Possiamo stimare questo quantitativo come composto da almeno una decina di milioni di tweet e altrettante conversazioni, per un relativo carico di utenti pari ad alcune migliaia.

Per quanto riguarda gli accessi ai dati, essi sono effettuati principalmente per il recupero delle informazioni sulle conversazioni, dei tweet che le compongono e sugli utenti che vi hanno partecipato. Per ogni singola conversazione sono dunque necessari un minimo di tre accessi, a cui si aggiungono due accessi per ogni altra eventuale risposta inserita. Non è inoltre ritenuto necessario effettuare relazioni fra le varie tabelle.

Nonostante quest'ultimo punto, si è deciso di orientare la propria scelta verso un database, essenzialmente perchè l'inserimento e il recupero dei tweet sarebbe risultato complesso senza una indicizzazione o un ordine sugli indici. Difatti, essendo i tweet inseriti non in modo sequenziale ma in maniera random, non sarebbe stato possibile organizzarli in modo ordinato all'interno dei file, se non ordinando ad ogni inserimento la lista e garantendo così tempi di ricerca accettabili. Inserendo invece le informazioni all'interno di un database, i tempi di ricerca di un determinato messaggio tramite il suo valore identificativo saranno sicuramente inferiori.

2.4 Indicizzazione dei dati

Come espresso nel paragrafo precedente, memorizzare i dati all'interno di un database risulta la scelta migliore per potervi accedere velocemente. Tuttavia la ricerca all'interno dei contenuti di testo memorizzati in una base di dati risulta un possibile collo di bottiglia dell'applicazione, soprattutto pensando al carico dati atteso. Per questo motivo risulta indispensabile la scelta di indicizzare il contenuto delle conversazioni per poter garantire un accesso rapido in fase di ricerca.

Per poter effettuare questa indicizzazione ci si è rivolti verso una delle possibili soluzioni *open source* di IR engine. Un IR engine (acronimo di *Information Retrieval Engine*) è composto essenzialmente da due componenti: l'*indexing* e il *query processing*.

Il primo permette di realizzare, partendo da una serie di documenti (dove un documento è un insieme di campi di testo), una indicizzazione degli stessi, solitamente tramite *inverted index*. Questa tecnica consiste nel realizzare un vocabolario in cui sono elencate tutte le parole presenti nei documenti e una lista, per ognuna di queste parole, che indica dove esse sono presenti. Gli IR engine adottano inoltre alcune tecniche per ottimizzare questa fase, in particolare per ricavare un indice meno pesante in termini di spazio fisico occupato e una velocità di interrogazione superiore [7]. Fra queste tecniche le più frequenti sono l'eliminazione delle *stopword* e l'uso dello *stemming*. Per *stopword* si intendono tutte quelle parole che appaiono molto di frequente nei testi ma non sono rilevanti in una ricerca: ad esempio articoli, preposizioni, verbi ausiliari e così via. Le *stopword* variano quindi in base alla lingua in cui il testo viene presentato: in un *social network* come

Twitter, i cui utenti provengono da tutto il mondo e quindi scrivono in tutte (o quasi) le lingue esistenti, l'eliminazione delle *stopword* si rivela una priorità essenziale nella costruzione di un indice performante. Eliminando infatti tutte le *stopword* delle principali lingue utilizzate, si potrà ottenere una sensibile riduzione delle dimensioni dell'indice. Per quanto riguarda lo *stemming*, esso viene applicato per ricondurre le parole alla loro radice. Si pensi ad esempio alla forma singolare e plurale di un nome comune: esso spesso si differenzia, in italiano, unicamente per l'ultima lettera. Risulta quindi dispendioso memorizzare due volte la stessa parola, con il medesimo significato, all'interno del nostro indice. Oltre ai nomi, questa tecnica è ben applicabile ai verbi. Tramite lo *stemming* viene dunque minimizzato il numero di parole presenti nell'indice utilizzando la loro sola radice.

Indicizzato il contenuto dei documenti, è possibile interrogare l'indice tramite il *query processing*. Tali interrogazioni vengono realizzate con semplici query che permettono la ricerca di corrispondenze fra la parola chiave e le parole contenute nel dizionario creato. I risultati vengono forniti inoltre in un ordine già stabilito in base alla rilevanza dei contenuti rispetto alla chiave di ricerca inserita. Per un ranking più efficace occorre però associare a questo valore una serie di aspetti propri del *social network*: tale aspetto verrà ampiamente trattato nel capitolo relativo.

Nella nostra applicazione l'indicizzazione sarà realizzata sul campo *text* della tabella conversazioni, campo che include tutti i tweet facenti parte della discussione stessa. Per poter recuperare, in fase di interrogazione, un identificativo della conversazione, oltre al campo text sarà inserito in ogni documento l'*id* della conversazione (ovvero l'id del tweet iniziale). Si introduce inoltre un ulteriore campo riportante il numero totale di risposte alla conversazione: questo ci permetterà in fase di ricerca di potere a priori escludere o meno le conversazioni formate da tweet singoli. Le ricerche saranno quindi effettuate sul campo text indicizzato e i risultati ci restituiranno l'id della conversazione e un ranking, espresso dall'IR engine secondo le proprie caratteristiche, che ci fornirà un indice di come i risultati vadano ordinati.

Capitolo 3

Ranking delle conversazioni

Dopo aver descritto il sistema per il recupero delle conversazioni su Twitter, spostiamo ora la nostra attenzione sulle tecniche di *ranking* per i risultati di una ricerca effettuata sui dati estratti.

L'importanza del *ranking* si rivela nel momento in cui i risultati di una ricerca sono in numero alto. In questo caso risulta impossibile, o comunque dispendioso in termini di tempo, controllare ciascun risultato al fine di trovare la conversazione più adatta a rispondere alla nostra ricerca. Oltre all'identificazione di corrispondenze fra una parola e il testo di una conversazione, occorre dunque aggiungere dei metodi che portino in cima alla lista dei risultati le discussioni ritenute più interessanti.

All'interno dei paragrafi seguenti verranno mostrate una serie di indicazioni utili per il calcolo del *ranking* riguardanti il testo dei messaggi, gli utenti partecipanti e la durata delle conversazioni. Si vedrà infine come è possibile effettuare l'aggregazione di questi risultati per ottenere un *ranking* globale.

Sebbene tale studio sia incentrato principalmente sulla realtà di nostro interesse, ovvero Twitter, verranno presentati tratti comuni che potranno rivelarsi utili per lo studio di *social network* di differente tipologia.

3.1 Rilevanza del testo

Dalla ricerca di una parola è lecito aspettarsi come risultato quelle conversazioni che includono corrispondenze di tale parola. Il *ranking* in base alle occorrenze sul testo è dunque il primo, e probabilmente più importante, passo per poter ordinare i risultati di una interrogazione. Tuttavia, come verrà dimostrato in seguito, questo non può essere considerato come l'unico metodo per poter organizzare i risultati. Si procede dunque con una pano-

ramica sulla rilevanza del testo all'interno di una conversazione e di come esso dovrà essere trattato.

Dato il testo di una conversazione, ricavato dal contenuto di tutti i messaggi che la compongono, la ricerca di una parola chiave viene effettuata trovando eventuali corrispondenze fra essa e le parole contenute all'interno del testo. La presenza di una o più corrispondenze può dare alla conversazione in esame un *ranking* più elevato rispetto ad una conversazione in cui le corrispondenze sono nulle o prossime allo zero. Potremmo dunque definire la rilevanza del testo come segue:

Definizione 3.1 (Rilevanza del testo)**.** Sia c una conversazione composta da m_1,...,m_n messaggi e *exp* l'espressione che si vuole ricercare all'interno di c. Definiamo la rilevanza del testo il rapporto fra il numero di corrispondenze ad *exp* nel testo e il numero complessivo delle parole presenti in c.

L'espressione *exp* può essere una semplice parola oppure una espressione più complessa, ad esempio la ricerca di due parole in modo mutuamente esclusivo. Calcolando la rilevanza del testo di una conversazione è quindi possibile ricavare un primo ordinamento dei risultati di una ricerca. Eppure, come accennato all'inizio del paragrafo, è bene notare come questo indice non sia bene applicabile nel caso di Twitter e in generale nei *social network*.

Ciò è dovuto essenzialmente alla natura del servizio offerto, ovvero la possibilità di inserire messaggi con un numero limitatissimo di caratteri. Questo porterà al sottointendere l'argomento della conversazione man mano che essa prenderà corpo, limitando la presenza dell'espressione ricercata solamente ai primi messaggi della conversazione. Si avrà quindi, all'aumentare delle risposte, un valore di rilevanza della conversazione sempre più basso, nonostante l'argomento iniziale sia invece oggetto di una lunga e magari rilevante discussione.

Per meglio comprendere questo aspetto, si pensi alle due conversazioni d'esempio qui di seguito riportate:

Conversazione 1:
u_1: Inizia oggi la ricerca su Twitter.

Conversazione 2:
u_1: La ricerca su Twitter sta dando i suoi frutti.
u_2: Quali sono i risultati?
u_1: Siamo riusciti a ricavare un metodo per il ranking.

Effettuando la ricerca della parola "Twitter", entrambe le conversazioni appariranno fra i risultati, ma la prima con valore di rilevanza pari a $\frac{1}{6}$ e la seconda $\frac{1}{22}$, sebbene quest'ultima abbia una rilevanza ben più importante poichè sviluppa effettivamente la discussione.

Pur essendo questo un argomento di notevole importanza, non è stato approfondito nella realizzazione di questa tesi, poichè il compito di effettuare il ranking sul testo sarà demandato a strumenti di ricerca più efficaci di un semplice rapporto occorrenze-parole. Utilizzando infatti un IR engine, di cui già si è parlato nel precedente capitolo, è possibile ricavare automaticamente un valore di ranking calcolato in base alla parola cercata e al testo di una conversazione.

Ritornando alla definizione 3.1, è bene chiarire il significato di corrispondenza fra una espressione e il testo della conversazione. Occorre notare per prima cosa che il testo di un messaggio è inserito da utenti di qualsiasi estrazione ed è quindi possibile che vengano introdotti, più o meno volutamente, differenti stili di scrittura. Ne sono esempi gli errori di battitura, ma anche forme in *slang* o abbreviazioni. Inoltre, nel momento in cui una persona sta effettuando la ricerca di una parola, è possibile che non sia interessata solamente ad essa, ma anche a parole molto simili e che condividono, ad esempio, la stessa radice. Si pensi ai verbi e alle loro varie coniugazioni, ma anche nuovamente a errori di battitura o forme alternative di scrittura della medesima parola.

Principalmente per i motivi sopra esposti è bene demandare il compito del ranking sul testo a sistemi già collaudati e più efficienti, come possono essere dei motori di ricerca appositamente realizzati per lavorare su testi. Come è stato trattato nel precedente capitolo e come verrà mostrato successivamente nella pratica, l'utilizzo di IR engine riesce a tener conto di buona parte delle problematiche sollevate in questo paragrafo, in particolare quelle riguardanti i differenti stili di scrittura. Purtroppo, dai test effettuati con uno di questi motori di ricerca, non si è invece trovata soluzione al problema riguardante la presenza sempre più rada della parola di ricerca man mano che la conversazione cresce.

È per questo motivo principalmente che il *ranking* effettuato unicamente sul testo non potrà essere l'unico per poter ordinare correttamente dei risultati di ricerca. Affiancato a questo sistema, che potremmo definire un primo filtro per recuperare le conversazioni più interessanti, ne devono essere associati altri che permettano una valutazione più efficiente e corretta delle conversazioni.

Un primo passo verso questa soluzione potrebbe essere l'inserimento di una relazione fra il valore del testo e il numero di risposte alla conversazione. Riprendendo l'esempio precedente, potremmo ad esempio moltiplicare il risultato dello *score* sul testo con il numero di risposte:

Conversazione 1: $\frac{1}{6}$ x $1 = \frac{1}{6} \simeq 0.166$

Conversazione 2: $\frac{1}{22}$ x 3 = $\frac{3}{22} \simeq 0.136$

Sebbene la seconda conversazione non riesca a raggiungere con sola questa relazione un valore maggiore della precedente, la differenza fra le due è sicuramente inferiore. Associando dunque a questo parametro altri fattori di ranking, potremmo darle ulteriore importanza, facendo aumentare la sua posizione nella lista dei risultati.

3.2 Popolarità degli utenti in base ai follower

Affrontata la rilevanza del testo delle conversazioni, si pone ora il problema di realizzare un *ranking* sugli utenti per determinare la rilevanza dei loro messaggi. Per comprendere il significato di questo parametro, si pensi ad un discorso realizzato da due entità diverse: se proviene da una persona considerata importante, come potrebbe esserlo il presidente di uno stato, esso avrà sicuramente più rivelanza dello stesso identico discorso proferito da una persona qualunque. Allo stesso modo, nei *social network*, è possibile che un messaggio all'interno di una conversazione assuma più o meno rilevanza in base alla persona che è stata ad inserirlo. Vi è inoltre da tener conto del contesto della conversazione: ad esempio un messaggio di argomento musicale e uno di carattere politico avranno probabilmente pesi diversi se a proferirli è stata la stessa persona.

All'interno di un *social network* il metodo ritenuto più efficace per poter valutare la rilevanza di un messaggio, escludendo momentaneamente il contesto, è quello di considerare il numero di *follower* dell'utente. Tramite questo indice, riscontrabile praticamente in tutti i *social network*, è possibile infatti ordinare gli utenti in base alla loro importanza sociale. Effettivamente, scorrendo la lista degli utenti più seguiti su Twitter, è possibile trovare quasi esclusivamente personaggi famosi. D'altra parte, un utente con un altissimo numero di follower può essere socialmente rilevante nonostante non sia effettivamente famoso: in questo caso viene ritenuto importante poichè i messaggi da lui postati hanno comunque un alto indice di letture e quindi risultano interessanti a un pubblico vasto.

È inoltre da tener conto che il numero effettivo di lettori non è sempre correttamente stimato in base ai *follower*. Infatti, come indicato in [8], un utente con un basso numero di contatti ma che spesso partecipa a conversazioni avrà un numero di lettori ben più alto dei suoi *follower*. Per comprenderne il motivo, si pensi che il numero di utenti iscritti ad un *social network* è un numero finito: prendendo due utenti con moltissimi *follower*, sicuramente alcuni di essi saranno in comune. Se quindi questi due utenti intrattengono una conversazione, il numero effettivo dei lettori non sarà la somma dei *follower* di entrambi, ma un numero ben inferiore. Due utenti

con pochi *follower*, e quindi idealmente facenti parte di due reti sociali molto diverse fra loro, propagheranno invece la loro conversazione ad un numero più alto di utenti, poichè aventi contatti differenti.

Per quanto riguarda il contesto della conversazione di cui si accennava in apertura, risulta difficile poter effettivamente calcolare la rivelanza di un utente e del suo messaggio in una determinata conversazione, a meno di non valutare singolarmente ciascun messaggio e il suo contesto. Vista la natura di questa ricerca, indirizzata a ottenere un'applicazione automatica che possa lavorare su una rete di messaggi molto estesa, è stato deciso di tralasciare il contesto delle conversazioni e di considerare unicamente la rilevanza degli utenti.

Procediamo quindi con una definizione formale relativa alla popolarità di un utente:

Definizione 3.2 (Popolarità di un utente - 1)**.** Si definisce popolarità di un utente il numero complessivo degli utenti che lo seguono, altrimenti detti *follower*.

Data questa definizione, possiamo vedere come la popolarità degli utenti possa essere un indice delle conversazioni a cui essi prendono parte. Continuando dal medesimo esempio posto in apertura del paragrafo, è facile intuire come un discorso realizzato fra persone importanti abbia più rilevanza rispetto allo stesso discorso affrontato da persone qualsiasi. La rilevanza dei partecipanti alla conversazione segna quindi, tramite la loro popolarità, l'intera popolarità della conversazione. Possiamo ora definire in modo formale ciò:

Definizione 3.3 (Popolarità di una conversazione - 1)**.** Si definisce popolarità di una conversazione c la somma delle popolarità dei singoli partecipanti $u_1,...,u_n$ divisa per il numero totale dei messaggi $m_1,...,m_k$: $\frac{\sum_{i=1}^{n} pop(user(m_i))}{\#m}$

3.3 Popolarità degli utenti in base alle conversazioni

Nel precedente paragrafo è stata definita la popolarità di un utente come il suo numero di *follower*. Da un'analisi più accurata di Twitter è bene notare come tale definizione non restituisca sempre un risultato attendibile. Si prenda ad esempio il numero di *follower* dell'account di un giornale come il Times, stimato oltre i 2 milioni: pur avendo un numero così alto di *follower* ed essendo effettivamente un account con una certa rilevanza, è anche vero che difficilmente darà vita a conversazioni. Questo è dovuto essenzialmente al modo in cui gli utenti percepiscono tale account, ovvero come un bot automatico piuttosto che un utente reale: per questo motivo difficilmente gli

risponderanno sperando in una vera conversazione.

Affiancata alla popolarità di un utente è bene dunque porre un indice sul suo effettivo coinvolgimento, calcolato in base al numero di conversazioni da lui iniziate e il numero di risposte ricevute. Possiamo definire in modo formale ciò:

Definizione 3.4 (Popolarità di un utente - 2)**.** Si definisce popolarità di un utente u il rapporto fra il numero di risposte totali ricevute e il numero di conversazioni da lui iniziate: $\frac{\sum_{i=1}^{N} \#reply(c_i)}{\#conv(u)}$

In questa maniera è possibile che un utente con meno *follower* del Times abbia più importanza, poichè le conversazioni da lui iniziate hanno un numero di risposte maggiori. Ciò è indice che l'utente in questione e le sue conversazioni sono ritenuti interessanti da più persone.

Questo nuovo indice, più che sostituirsi al precedente, è un buon mezzo per ottenere risultati ancor più rilevanti. Associando infatti a questo parametro, come verrà descritto successivamente, un peso più o meno alto rispetto a quello riguardante i *follower*, potremo dare risalto agli utenti effettivamente attivi rispetto ai bot.

3.4 Ranking su parametri di tempo

Come accennato nel capitolo riguardante il modello di conversazione su Twitter, il fattore tempo all'interno delle conversazioni è un aspetto molto importante. Per effettuare un ranking sulle conversazioni basato su questo fattore, dobbiamo occuparci di due aspetti distinti: il primo riguardante il periodo in cui la conversazione si è sviluppata, il secondo i tempi di risposta alla stessa. Nei seguenti paragrafi verranno trattati questi due aspetti.

3.4.1 Attualità della conversazione

Ogni conversazione è caratterizzata da un istante di inizio e uno di fine, parametri che ci permettono di porre la conversazione in un determinato istante del passato. Questa particolarità può rivelarsi importante nel momento in cui un utente è alla ricerca di una conversazione avvenuta in un certo periodo. Un chiaro esempio è la ricerca riguardante un avvenimento appena accaduto: in tal caso si preferirà avere risultati molto recenti, piuttosto che riguardanti avvenimenti accaduti magari diversi mesi prima.

È necessario dunque introdurre una definizione per la rilevanza di una conversazione in base al periodo in cui essa si è svolta:

Definizione 3.5 (Attualità di una conversazione)**.** Si definisce attualità di una conversazione la differenza fra il tempo di inizio della conversazione *t_start* e un istante prefissato t inferiore: $t\_start - t$, con $t \leq t\_start$

Nel calcolo di questo parametro può essere considerato come istante prefissato il tempo di inizio più basso fra le conversazioni risultanti oppure un istante prefissato precedente a tutti i tempi di inizio delle stesse. In questa maniera è possibile ottenere un indice più alto per conversazioni recenti e uno più basso per quelle accadute da diverso tempo.

Questo parametro può essere o meno preso in considerazione durante una ricerca, in base alla scelta se ricavare discussioni recenti oppure discussioni più rilevanti senza tener conto di quanto esse siano datate. Questa scelta sarà possibile assegnando un determinato peso al parametro di attualità, come verrà mostrato nell'ultimo paragrafo.

3.4.2 Densità della conversazione

Poniamo ora l'attenzione sui tempi di sviluppo di una conversazione. Introduciamo la seguente definizione:

Definizione 3.6 (Durata di una conversazione)**.** Si definisce durata di una conversazione c composta dai messaggi m_1,...,m_n la differenza fra l'istante di fine e l'istante di inizio della conversazione stessa: $t_n - t_1$

Preso il modello di conversazione di Twitter, occorre evidenziare come l'istante di fine, nel caso la conversazione abbia più rami, sia identificato dall'istante di tempo maggiore, ovvero dall'ultimo messaggio inserito in uno qualsiasi dei rami.

Introdotta la 3.6, possiamo fornire una definizione di densità:

Definizione 3.7 (Densità di una conversazione - 1)**.** Si definisce densità di una conversazione c il rapporto fra la durata della conversazione e il numero dei messaggi che la compongono: $\frac{t_n - t_1}{\#m}$

Data la definizione 3.7, si noterà che l'indice ricavato non fornisce tutte le informazioni attese. Si prenda ad esempio la situazione in Figura 3.1, in cui sono indicate due conversazioni iniziate e finite nello stesso istante e con ugual numero di messaggi, ma con differenti intervalli fra essi. Il calcolo della densità come definito in 3.7 non terrà conto della differenza, in termini di durata degli intervalli, fra le due.

Per poter fornire un indice che tenga conto di queste possibili differenze, ridefiniamo il concetto di densità:

Definizione 3.8 (Densità di una conversazione - 2)**.** Si definisce densità di una conversazione c la sommatoria dell'inverso delle differenze fra gli istanti di tempo $t_1, ..., t_n$ dei messaggi $m_1, ..., m_n$: $\sum_{i,j \in c} \frac{1}{t_j - t_i}$

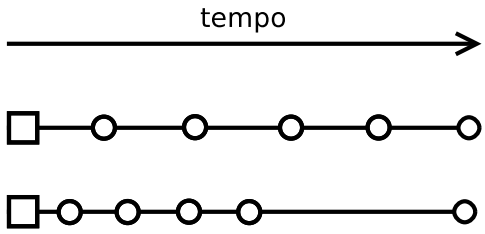

Fig. 3.1: Due conversazioni con ugual numero di messaggi ma differente densità.

L'indice di densità, come definito in 3.8, ci fornisce un indicatore di come una conversazione si è evoluta nel tempo. Un valore di densità alto è indice di una conversazione avvenuta in tempi brevi, mentre un valore basso è ricavato da una conversazione con risposte molto diluite nel tempo. Una conversazione composta da un unico messaggio avrà valore pari a 0.

È bene considerare che, all'aumentare del numero di messaggi all'interno di una conversazione, vi sarà un relativo crescere del valore della densità. Per questo motivo, per avere un indice più rigoroso, è bene una volta calcolata la densità dividere questo valore per il numero totale del messaggi della conversazione. Ridefiniamo dunque il concetto di densità:

Definizione 3.9 (Densità di una conversazione - 3)**.** Si definisce densità di una conversazione c il rapporto fra la sommatoria dell'inverso della differenza fra gli istanti di tempo $t_1, ..., t_n$ dei messaggi $m_1, ..., m_n$ e il numero totale dei messaggi: $\frac{\sum_{i,j \in c} \frac{1}{t_j - t_i}}{\#m}$

Discusse le modalità per il calcolo della densità, è necessario ora soffermarsi sull'effettivo utilizzo di tale indice. La densità di una conversazione, come definita in 3.9, ci fornisce un valore numerico per comprendere quanto questa abbia attirato l'attenzione dei suoi partecipanti. Si può pensare che una conversazione con valore di densità alto, e quindi con risposte molto ravvicinate, sia una discussione che ha destato molto interesse. D'altra parte, una conversazione con valore basso può indicarci una discussione che si è protratta per lunghi periodi ed è quindi idealmente meno importante. Si pensi all'analogia, nel mondo reale tanto quanto in quello *on-line*, con un dibattito su importanti temi politici e uno su temi personali: nel primo caso i tempi di risposta saranno quasi sicuramente più rapidi, soprattutto all'aumentare dei partecipanti, mentre nel secondo, vista la natura della conversazione, è difficile che vi sia un botta e risposta sempre più rapido.

Analizzando tuttavia la natura del servizio di Twitter e in generale delle discussioni *online*, è facile intuire che non sempre i tempi di risposta siano indice della sua importanza. Si pensi al caso in cui una conversazione inizi in un determinato istante t dall'utente u_1 e fino all'istante s, con $s \gg t$, il

diretto interessato u_2 non sia *online* o non possa rispondere. In tale caso la densità aumenterà, non tenendo conto che l'utente u_2 non è a conoscenza della conversazione o sia impossibilitato a parteciparvi. Questa particolarità rientra nel modello di discussione descritto nel primo capitolo, in cui si è fatto notare che i tempi di una conversazione *online* sono differenti rispetto a una reale.

Pur tenendo conto di questo particolare, possiamo ragionevolmente pensare che, all'aumentare del numero di partecipanti a una discussione, l'indice di densità assuma un valore effettivamente indicativo. Come verrà trattato nell'ultimo paragrafo del presente capitolo, occorrerà assegnare un peso ben ponderato a questo tipo di *ranking*, per evitare che sia un fattore troppo determinante nell'ordinamento dei risultati.

3.5 Popolarità dei messaggi

In ultima analisi per il calcolo del *ranking* delle conversazioni è bene trattare un argomento non sempre applicabile ai *social network*, ma che in Twitter assume un ruolo importante. Si tratta dei *retweet*, ovvero la possibilità di ripostare il messaggio di un altro utente affinchè tutti i propri *follower* possano leggerlo. È naturale comprendere come con questa tecnica il messaggio di una conversazione possa raggiungere ancor più visibilità, passando dalla *timeline* di un utente a quella di un altro (non necessariamente *follower* del primo). I motivi per cui un messaggio viene *retweetato* possono essere svariati, ma la base di questo sistema è che il tweet in questione viene considerato importante e quindi degno di essere letto da più persone, assumendo quindi un valore più alto dei messaggi normali.

È dunque possibile utilizzare questo indicatore come un nuovo parametro per il calcolo del *ranking* di una conversazione, e in particolare dei messaggi che la compongono. Definiamo la popolarità di un messaggio nella seguente maniera:

Definizione 3.10 (Popolarità di un messaggio)**.** Si definisce popolarità di un messaggio il numero complessivo dei suoi *retweet*.

Come è lecito aspettarsi in una rete estesa come quella di Twitter, il numero di messaggi ripostati è abbastanza basso: in [3] esso viene stimato sul 8.70% dei messaggi totali, valore confermato dal campione da noi raccolto, comprensivo di una quantità ben superiore di messaggi, in cui il numero di *retweet* raggiunge il 7.93%. È stato dunque deciso di mantenere questo indice, proprio per assegnare alle poche conversazioni con *retweet* un peso maggiore nei risultati.

Dalla definizione 3.10 ricaviamo dunque un nuovo metodo per il calcolo della popolarità di una conversazione:

Definizione 3.11 (Popolarità di una conversazione - 2)**.** Si definisce popolarità di una conversazione c la somma delle popolarità dei singoli messaggi m_1,...,m_n divisa per il numero totale dei messaggi m_1,...,m_n: $\frac{\sum_{i=1}^{N} pop(m_i)}{\#m}$

Questo fattore, come accennato in apertura, non è sempre presente nei *social network*, sebbene vi possano essere degli indicatori simili. Nel caso di Facebook, ad esempio, la possibilità di segnalare con il celebre "mi piace" un qualsiasi messaggio può divenire indice della popolarità dello stesso.

È bene notare inoltre che all'interno di Twitter è presente un'opzione per poter segnalare i tweet di altri utenti come preferiti. Tale possibilità non è stata presa in considerazione in questo studio a causa dell'impossibilità nel ricavare il numero di preferiti relativi a un dato messaggio. Questa funzionalità non è infatti prevista da parte delle API di Twitter ed è quindi stata scartata a priori. Eventuali modifiche future al *social network* in questione e alle API disponibili potrebbero riportare in valutazione questo importante parametro.

3.6 Aggregazione dei ranking di base

Indicati i metodi per calcolare il *ranking* sui vari aspetti di una conversazione, quali gli utenti partecipanti, la rilevanza del testo, la durata nel tempo e la popolarità dei messaggi, si vede ora come è possibile aggregare tali risultati per ottenere un ranking generale, denominato da qui in avanti *score* della conversazione.

Per prima cosa occorre identificare con valore di *score* pari a 0 una conversazione che non include alcun messaggio, ovvero una conversazione inesistente:

$$score() = 0$$

Una conversazione c composta da un singolo messaggio m vedrà il suo *score* come l'aggregazione semplice dei precedenti *ranking*:

$$score(c) = score(m) =$$
$$rilevanza+popolarità\_utente+attualità+densità+popolarità\_messaggio$$

Similmente, potremo calcolare lo *score* di una conversazione che presenti più di un messaggio; in tal caso lo *score* complessivo sarà calcolato come la somma dei singoli *score*:

$$score(c) = score(m_1, ..., m_n) = score(m_1)+...+score(m_n)$$

È bene notare che i valori dei vari ranking dovranno essere forniti utilizzando una stessa scala, di modo che valori potenzialmente molto alti (ad

esempio la popolarità di un utente, espressa in una scala da 0 a 5 milioni circa) non penalizzino i fattori con scala ben più ridotta (ad esempio il ranking sul testo). Per poter ricondurre i valori di questi indici a una scala comune, si è deciso di calcolare i valori dei singoli ranking come il loro rapporto con il valore massimo dello stesso indice all'interno della specifica ricerca, in maniera da avere tutti i valori compresi fra 0 e 1.

Abbiamo precedentemente indicato l'aggregazione dei *ranking* come la somma dei valori stessi. Tuttavia, è possibile migliorare questa forma di aggregazione associando a ciascuno dei parametri un certo peso. Si pensi, ad esempio, di voler effettuare una ricerca in cui non si è interessati tanto alla densità della conversazione, quanto piuttosto alla popolarità degli utenti che vi hanno partecipato. Possiamo quindi associare ai parametri un peso, ad esempio espresso in una scala da 0 a 100, pari a 80 per la popolarità e 30 per la densità. In questa maniera l'intero *score* sarà soggetto a cambiamenti, portando così il riordino dei risultati finali in modo differente rispetto a quello standard.

La possibilità di associare un peso a ciascuno dei parametri di *ranking* potrà essere resa disponibile o meno direttamente alla persona interessata alla ricerca. Per utenti inesperti sarà invece opportuno impostare con valori di default tali pesi. Un possibile esempio potrebbe essere il seguente:

rilevanza: 70% - popolarità conversazione: 50% - popolarità utenti: 50%
attualità: 100% - densità: 30% - popolarità messaggi: 70%

In questo esempio viene data più rilevanza al testo della conversazione e alla popolarità dei messaggi, quindi a quella degli utenti partecipanti e infine alla densità. Si vogliono avere in particolare i messaggi più recenti. I risultati attesi sono quindi quelli inseriti da pochissimo tempo, con più occorrenze del termine ricercato e ritenuti più interessanti dagli utenti del *social network*. Il peso degli utenti che hanno partecipato alla conversazione è mediamente importante, mentre i tempi di risposta sono ritenuti poco interessanti.

La scelta di impostare dei valori di default non può essere realizzata casualmente, ma dovrà essere oggetto di studio. In particolare essa dovrebbe pervenire da una ricerca sugli usi del sistema da parte degli utenti stessi: valutando le impostazioni manuali dei vari parametri, potrebbe essere possibile determinare a cosa essi sono più interessati. Nella presente tesi tale studio non è stato affrontato, in quanto esso dovrebbe partire esattamente da dove tale tesi termina. Si rimanda dunque l'argomento ad eventuali articoli futuri.

In conclusione possiamo indicare la formula per il calcolo dello *score* di un singolo messaggio in questa maniera:

$$
\begin{gathered}
score(m) = \\
((r/max\_r) \times peso\_r) + \\
((f/max\_f) \times peso\_f) + \\
((p/max\_p) \times peso\_p) + \\
((a/max\_a) \times peso\_a) + \\
((d/max\_d) \times peso\_d) + \\
((m/max\_m) \times peso\_m)
\end{gathered}
$$

Dove sono stati posti: r = rilevanza, f = popolarità sui follower, p = popolarità sulle conversazioni per utente, a = attualità, d = densità, m = popolarità dei messaggi (retweet)

Capitolo 4

Demo

Si presenta in questo capitolo l'intero sistema realizzato per poter recuperare le conversazioni presenti su Twitter ed effettuare ricerche su di esse. Nel primo paragrafo vengono presentati gli strumenti utilizzati per la sua realizzazione, nel secondo come è stata implementata l'estrazione dei dati e la loro indicizzazione, nel terzo il calcolo del ranking sulle conversazioni e nell'ultimo un prototipo *web-based* che mostri come è possibile rendere disponibile agli utenti un sistema così composto.

4.1 Strumenti usati

Si introducono ora gli strumenti utilizzati per la realizzazione del sistema: il database, l'IR engine per l'indicizzazione e infine una libreria per l'utilizzo delle API di Twitter.

4.1.1 MySQL

Come introdotto nel secondo capitolo, si è scelto di memorizzazione le informazioni riguardanti tweet, utenti e conversazioni all'interno di un database. Per motivi di disponibilità, ci si è orientati verso l'utilizzo di MySQL [9].

MySQL è un RDBMS (relational database management system)*open source* sviluppato dalla metà degli anni '90 e disponibile sia in ambiente Unix che Windows. La popolarità di MySQL è cresciuta negli anni grazie al suo impiego all'interno di piattaforme di sviluppo come LAMP ed è tutt'oggi utilizzato da alcuni importanti siti come Google, Wikipedia e Facebook grazie alla sua scalabilità. MySQL supporta la sintassi SQL ed è previsto il pieno supporto ad ANSI nelle prossime release.

La versione utilizzata nello sviluppo della presente tesi è la 5.0.51a, scelta per motivi di compatibilità con il sistema operativo in uso.

4.1.2 Lucene

Per l'indicizzazione delle conversazioni, ci si è orientati verso l'utilizzo di Lucene [10], un *IR engine open source* gratuito e implementato in Java. Sviluppato a partire dal 2000 da Doug Cutting, fa parte della Apache Software Foundation ed è rilasciato sotto omonima licenza. È attualmente uno degli strumenti più utilizzati per l'implementazione di ricerche *full-text* nonché per la realizzazione di motori di ricerca, tanto sul web quanto in ambienti locali.

L'idea alla base di questa libreria è il considerare i documenti come composti da campi, tramite cui l'indice creato sarà direttamente interrogabile. Lucene permette inoltre l'indicizzazione di moltissimi formati, quali PDF, HTML, Microsoft Word, OpenDocument, nonché la possibilità di indicizzare i contenuti di un database semplicemente estraendoli tramite query.

La versione utilizzata nello sviluppo della presente tesi è la 3.0.2, l'ultima stabile rilasciata dagli sviluppatori.

4.1.3 Twitter4J

Per l'interrogazione delle API di Twitter è stata utilizzata Twitter4J [5], una libreria non ufficiale scritta in Java che permette l'utilizzo delle API fornite dal *social network*. Sviluppata da Yusuke Yamamoto a partire dal 2007 è rilasciata sotto licenza BSD.

La versione utilizzata nello sviluppo della presente tesi è la 2.1.7, l'ultima stabile rilasciata dallo sviluppatore.

4.2 Il sistema nel dettaglio

Presentati gli strumenti utilizzati nella realizzazione del sistema, passiamo ora all'implementazione dell'estrazione dati e della loro indicizzazione. Viene a questo scopo mostrata in Figura 4.1 una rappresentazione più dettagliata di quella concettuale presentata nel secondo capitolo.

Il sistema realizzato vede, a partire dall'alto, il social network e le API a disposizione con cui è possibile recuperare da Twitter informazioni su utenti e messaggi. Tali informazioni sono recuperate da una rete di calcolatori, il cui scopo e i cui compiti saranno in seguito spiegati, e immagazzinate all'interno di un database, ospitato su un'altra macchina. Su questa stessa macchina viene inoltre svolto il ruolo di indicizzare le conversazioni e di permettere, tramite un'interfaccia (*web-based*, ma disponibile anche tramite console), la ricerca delle stesse. Nei seguenti paragrafi verranno trattati nello specifico i tre aspetti principali del sistema: l'estrazione dei dati, la loro memorizzazione e l'indicizzazione degli stessi. Partendo da quest'ultimo passo, nel capitolo successivo verrà presentato il meccanismo di ranking alla base del motore di ricerca.

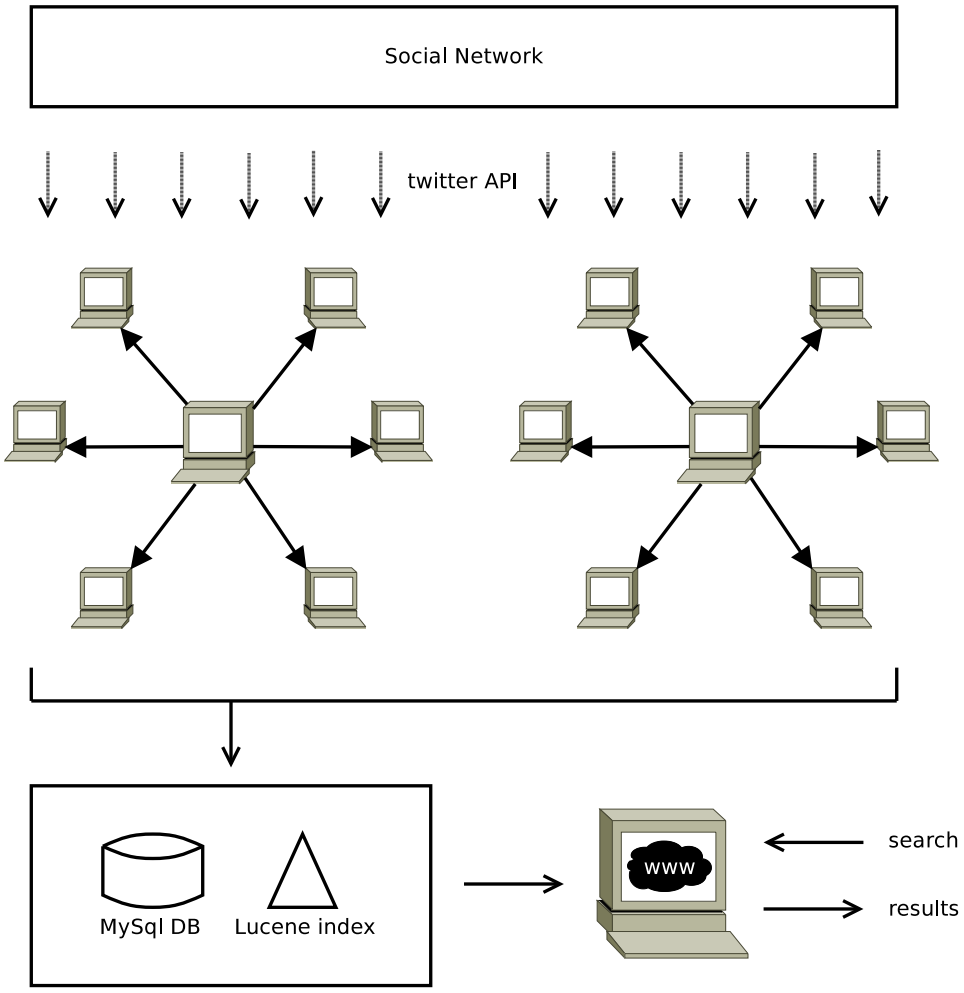

Fig. 4.1: Il sistema realizzato per l'estrazione, la memorizzazione e la ricerca delle conversazioni.

4.2.1 Estrazione dei dati

Nel realizzare un sistema per l'estrazione delle informazioni dal *social network* in questione è stata posta molta attenzione nelle performance, in particolare nel numero di tweet estraibili rapportati ai tempi e al numero di richieste tramite API. Per far ciò ci si è affidati a una serie di macchine, nell'ordine della cinquantina, coordinate fra loro per permettere simultaneamente l'estrazione delle informazioni. In questa maniera è stato possibile recuperare in breve tempo un campione significativo di dati senza essere legati al limite imposto di 350 richieste per indirizzo IP delle API.

Il sistema, come è visibile nella figura precedente, è stato realizzato ponendo una macchina come server centrale e gli altri computer come client direttamente connessi ad esso. È stato inoltre pensato di predisporre due reti server-client separate fra loro, in maniera da poterle utilizzare singolarmente in caso di guasti o indisponibilità dei calcolatori.

Presentiamo ora i due ruoli assunti dalle macchine del sistema: server e client.

Server

Il server centrale ha il compito di ricavare, mediante API, la *public timeline* disponibile su Twitter. Per far ciò si autenticherà tramite il metodo OAuth, quindi effettuerà una richiesta per la *public timeline*. Estratti da essa i 20 *user id* corrispondenti ai 20 tweet, si porrà in attesa di connessioni entranti da parte dei client. Ricevuta una richiesta, consegnerà uno degli id utente, eliminandolo quindi dalla sua lista e inserendo tutte le informazioni di tale utente all'interno del database. Terminati i 20 id utente, il server provvederà ad effettuare nuovamente la richiesta di una *public timeline* e riprenderà il ciclo sopra descritto.

Per ottimizzare il numero di richieste effettuate, sono state introdotte alcune migliorie. Per prima cosa, prima di consegnare un id, viene verificato se l'utente da esso contraddistinto è stato recentemente trattato. In tal caso è possibile che la maggior parte dei tweet siano già stati scaricati e può essere quindi scartato l'utente in favore di altri mai trattati. Il periodo settato nel nostro caso è di due giorni, ma l'intervallo è chiaramente adattabile per migliorare le prestazioni.

Per ricavare fin da subito un campione molto significativo in termini di utenti rilevanti, è stato deciso di utilizzare una lista di id utenti di celebrità registrate su Twitter. Facilmente ricavabile da uno dei tanti siti internet dedicati a Twitter, è stato possibile in questa maniera recuperare fin da subito i dati di oltre 6 mila utenti fra i più seguiti sul *social network*.

Client

I client hanno lo scopo di effettuare tutte le richieste riguardanti il recupero dei tweet e le relative conversazioni. Il primo passo di un client consiste nel ricevere dal server un id utente. Ricavato l'id, effettuerà una richiesta per la sua timeline, al cui interno sono inclusi gli ultimi 200 tweet di tale utente. Scorrendo man mano ognuno di essi, si verificherà il loro essere parte di una conversazione o il loro status di tweet singoli. In quest'ultimo caso, essi saranno inseriti direttamente all'interno del database e il numero di conversazioni dell'utente aumentato di una unità (il numero di reply rimarrà invece invariato, poichè nessuno ha risposto al suo messaggio). Nel caso il tweet sia invece parte di una conversazione, occorrerà verificare che parte di essa non sia già presente nel database, poichè già estratta precedentemente. In tal caso la conversazione verrà aggiornata con gli ultimi tweet e allo stesso modo verranno aumentati il numero di risposte alle conversazioni dell'utente. Nel caso invece la conversazione non sia mai stata estratta, essa verrà interamente recuperata tramite le API e inserita nella tabella delle conversazioni, nonché aggiornato lo status dell'autore iniziale dell'utente.

Per ottimizzare il compito dei client, sono state adottate diverse tecniche, in modo da garantire il funzionamento di tante macchine connesse contemporaneamente allo stesso database. Per prima cosa, occorre segnalare che è possibile che un utente sia già stato trattato precedentemente e quindi molti dei suoi messaggi siano già stati scaricati. Occorrerebbe quindi interrogare, prima dell'inserimento di ogni tweet, il database per verificare che esso non sia già presente. Per evitare queste interrogazioni, viene effettuata a priori l'estrazione degli id di tutti i tweet già memorizzati per tale utente. In questa maniera, con una sola richiesta invece che una per ciascuno dei 200 tweet, abbiamo già l'elenco di tutti i messaggi da non inserire.

Molto simile è quanto viene fatto per le conversazioni: in questo caso è possibile trovare nuovi tweet in risposta a conversazioni già parzialmente memorizzate. È necessario quindi verificare passo passo che il tweet padre dell'attuale (e quindi la relativa conversazione) non sia già stato memorizzato precedentemente, in modo da evitare nuove richieste. In tal caso non è possibile conoscere a priori la presenza di una determinata conversazione, ma trovato il padre dell'attuale tweet, possiamo recuperare facilmente l'intera conversazione e semplicemente aggiornarla con i nuovi messaggi.

Per l'inserimento dei tweet e delle conversazioni viene inoltre utilizzato INSERT DELAYED, una estensione del linguaggio SQL che permette l'esecuzione di INSERT senza il bisogno di attendere i tempi di completamento da parte del database. In questa maniera viene infatti demandato l'inserimento direttamente al db, riducendo i tempi di esecuzione sia da parte del db stesso, che effettua l'inserimento in blocco degli statement, sia da parte dei client.

4.2.2 Indicizzazione delle conversazioni

Per l'indicizzazione dei dati, realizzata tramite Lucene, è stato deciso di implementare un meccanismo incrementale che permettesse di aggiungere man mano le conversazioni inserite nel database. In questa maniera è stato possibile costruire un indice di dimensioni molto grandi senza preoccuparsi né dei tempi necessari per realizzarlo né dell'uso estensivo di CPU e RAM. La possibilità inoltre di poter aggiungere in qualsiasi momento nuove conversazioni è considerata importante in un sistema che nel tempo continuerà ad essere utilizzato.

L'indicizzazione viene dunque effettuata a intervalli regolari, basata sulla quantità media di conversazioni estratte al minuto, e permette l'inserimento delle nuove conversazioni nel frattempo inserite nel db. Ogni documento è identificato dall'id della conversazione e dal testo della stessa. È stato inoltre aggiunto il numero di reply alla conversazione per avere fin dall'interrogazione dell'indice l'elenco delle vere conversazioni e quello dei tweet singoli, permettendo all'utente di scegliere cosa visualizzare nei risultati.

4.3 Ranking dei risultati

Analizzato il sistema di recupero delle informazioni necessarie alla realizzazione delle conversazioni, si presenta ora il meccanismo di ranking per i risultati finali di una ricerca.

Il primo passo della ricerca viene effettuato tramite l'interrogazione dell'indice creato tramite Lucene per trovare l'elenco delle conversazioni che includono la parola chiave ricercata. Questa interrogazione ci restituisce, oltre che l'id iniziale della conversazione, un valore di ranking sul testo, calcolato automaticamente dall'IR engine, e il numero di risposte alla discussione. In questa maniera abbiamo già a disposizione il primo valore necessario per il ranking, ovvero la rilevanza del testo.

Come secondo passaggio possiamo dunque ordinare i risultati in base a questo primo ranking, in modo da ricavare da un numero molto alto di risultati un numero ristretto, ad esempio i 20 risultati più rilevanti per testo e numero di risposte. Infatti i risultati forniti da Lucene sono ordinati in base all'ordine di inserimento del documento nell'indice: questo lo porterà a restituire sempre e solo i primi documenti relativi alla parola chiave cercata, a meno di non richiedere un numero abbastanza alto di risultati da riordinare successivamente in base alle nostre preferenze.

Per ogni conversazione presente fra i risultati viene dunque interrogato il database per conoscere tutte le informazioni necessarie al calcolo degli altri ranking: numero di retweet, densità della conversazione e tempo di inizio. Vengono inoltre ricercate nel db le informazioni relative agli utenti che hanno partecipato alla conversazione: il loro numero di follower, il numero di

discussioni iniziate e il numero di risposte ottenute.

Effettuata questa ricerca per tutte le conversazioni presenti nei risultati, vengono estratti i massimi per ciascuno degli indici elencati e viene calcolato il totale di ogni conversazione, tenendo conto dei pesi associati a ogni indice.

Di seguito vengono indicati i calcoli per ciascun indice:

- Rilevanza del testo: $(score/max\_score) \times peso$
  lo score viene calcolato dall'IR engine e moltiplicato per il numero di risposte alla conversazione. Max_score è il massimo di questo score fra tutti i risultati;

- Popolarità in base ai follower: $(\frac{followers}{reply+1}/max(\frac{followers}{reply+1})) \times peso$
  in questo caso $\frac{followers}{reply+1}$ è il rapporto fra la somma dei follower di tutti gli utenti partecipanti alla conversazione e il numero di messaggi della stessa. È da notare che il numero di reply da noi memorizzato non tiene conto del primo messaggio della conversazione: occorre dunque incrementare di uno tale valore;

- Popolarità in base alle conversazioni: $(\frac{n\_reply}{n\_story}/max(\frac{n\_reply}{n\_story})) \times peso$
  dove $\frac{n\_reply}{n\_story}$ è il rapporto fra la somma del numero di risposte alle conversazioni degli utenti partecipanti e il numero di conversazioni iniziate dagli stessi;

- Attualità della conversazione: $\frac{time\_start}{max\_time\_start} \times peso$
  in questo caso time_start è il tempo di inizio della conversazione espresso in millisecondi nel formato Unix/epoch;

- Densità della conversazione: $\frac{time\_density}{max\_time\_density} \times peso$
  dove $\frac{time\_density}{max\_time\_density}$ è il rapporto fra la densità della conversazione e il massimo della stessa;

- Popolarità della conversazione: $(\frac{n\_retweet}{reply+1}/max(\frac{n\_retweet}{reply+1})) \times peso$
  in questo caso $\frac{n\_retweet}{reply+1}$ è il rapporto fra la somma del numero di retweet e il numero di messaggi della conversazione.

Riordinati dunque i risultati in base al totale, ottenuto sommando tutti gli indici precedentemente calcolati, vengono presentati all'utente le conversazioni in risposta alla sua ricerca. Sarà per lui possibile selezionare il risultato più interessante e visualizzare l'intera conversazione.

4.4 Uso pratico: the online Twitter Conversation Retrieval

In questo paragrafo viene mostrata un'applicazione *web-based* esemplificativa del sistema realizzato. Accedendo al servizio viene mostrato un form di ricerca simile a quello presente in Figura 4.2. Nel campo di testo è possibile inserire una o più parole chiave che vogliono essere ricercate. Sotto ad esso è possibile assegnare dei pesi differenti ai vari indici di ranking descritti nei precedenti capitoli. In base a queste impostazioni, i risultati saranno ordinati in modo da offrire i risultati più rilevanti in base alle scelte dell'utente. Vi è inoltre la possibilità di richiedere che nei risultati siano incluse unicamente le vere conversazioni e non i tweet singoli. Per parole chiave molto utilizzate questa scelta è ininfluente, poichè è più probabile che le conversazioni con più risposte abbiano uno score più alto di quelle senza. Per parole invece poco utilizzate, possiamo in questa maniera richiedere che ci vengano restituite solamente le conversazioni con risposte. Questo può naturalmente portare a ricavare zero risultati nel caso in cui non esistano vere conversazioni ma solamente tweet singoli.

Fig. 4.2: Interfaccia per l'interrogazione del sistema.

Avviando la ricerca vengono mostrati i risultati della stessa, come è possibile vedere nella Figura 4.3. Già ordinati in base ai vari parametri impostati, i risultati mostrano il tweet iniziale della conversazione, il nome dell'autore e il suo avatar, la data di inizio, il numero di risposte e di retweet che ha ricevuto. In basso a destra, per soli scopi di debug, è visibile inoltre il punteggio totale del ranking ricevuto dalla particolare conversazione. Tutti i dati visualizzati sono estratti direttamente dal db e non richiesti tramite le API a disposizione, rendendo quindi il sistema scollegato dal funzionamento effettivo del *social network*.

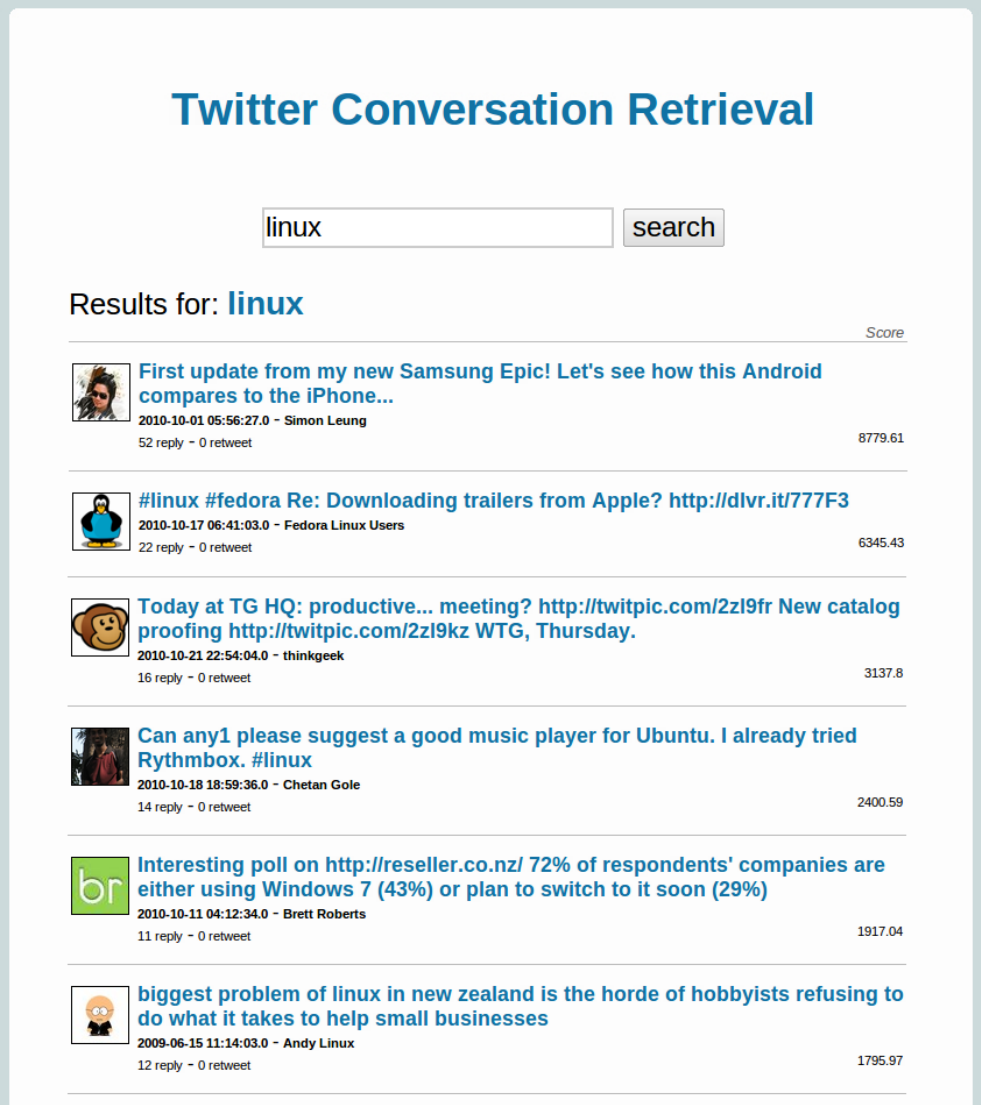

Fig. 4.3: Risultati di una interrogazione.

Cliccando su uno dei risultati si avrà la possibilità di visualizzare l'intera conversazione con le relative risposte, così come mostrato in Figura 4.4. I tweet sono posizionati ad albero così come si è sviluppata la conversazione, permettendo di leggere ogni possibile ramo separatamente e senza perdere il senso della stessa.

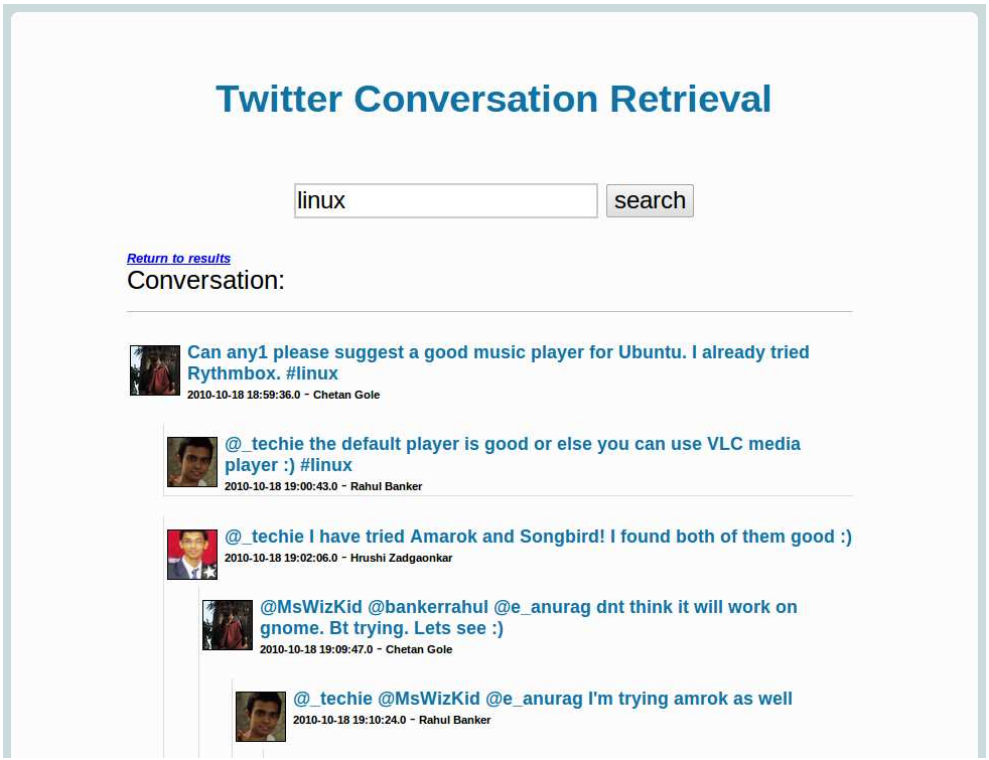

Fig. 4.4: Esempio di conversazione.

Capitolo 5

Conclusioni

Come conclusione del lavoro svolto, in questo capitolo vengono riassunte le analisi sui risultati ottenuti e presentati due esempi di ricerca. Nell'ultimo paragrafo vengono inoltre indicati spunti per ulteriori sviluppi dell'applicazione, a partire dal recupero delle conversazioni fino a possibili integrazioni del sistema per ricerche più complesse.

5.1 Analisi dei dati raccolti

I dati estratti per il test sul sistema realizzato sono i seguenti:

- 12˙817˙704 tweet, di cui 8˙642˙098 senza risposta e i rimanenti 4˙175˙606 facenti parte di conversazioni

- 9˙749˙257 conversazioni totali

- 618˙055 utenti totali

In media ogni conversazione è composta da 2.5 tweet, mentre il massimo numero di risposte ricevute da una conversazione è 234. Sul totale degli utenti trattati circa il 4.2% non ha mai ricevuto risposte ai propri tweet mentre il 30.1% supera la media dei 2.5 tweet in risposta. Il numero di follower e il numero di risposte ricevute non sono sempre correlati fra loro: elencando i primi utenti per numero di risposte e quelli con numero di follower, solo il 2% di essi sono in comune. Questo mostra che non sempre gli utenti più seguiti sono anche quelli che realizzano conversazioni più seguite, rendendo dunque necessario il ranking calcolato anche in base al numero di conversazioni iniziate e il numero di risposte totali.

Vengono di seguito presentati due esempi di ricerca per mostrare come i vari parametri influenzino i risultati ottenuti.

Esempio 1

Viene ricercato all'interno dell'indice la parola "soccer", includendo unicamente conversazioni con almeno una risposta. I risultati forniti in questo caso dall'indice sono 344. Ricavate le informazioni di base, sono riordinati una prima volta in base allo score calcolato da Lucene moltiplicato per il numero di risposte e vengono presi in considerazione solamente i primi 20 risultati con score più alto. Di essi sono cercate le varie informazioni contenute nel database, calcolato lo score finale e riordinati nuovamente.

La prima ricerca viene effettuata assegnando ai pesi valori equivalenti e diversi da zero, in modo da ricavare un ordinamento di base dei risultati senza che i pesi influiscano su di esso. I risultati forniti, ponendo i pesi tutti equivalenti a 1 e, per motivi di praticità, limitati ai primi sei, sono mostrati nelle Tabelle 5.1 e 5.2. La prima tabella mostra le informazioni estratte direttamente dal database, ovvero quelle sulla conversazione (id, numero di retweet, data di inizio, densità, numero di risposte) e sugli utenti che vi hanno partecipato (numero totale di follower, di conversazioni iniziate e di risposte ricevute), oltre allo score calcolato direttamente da Lucene. Nella seconda tabella vengono invece calcolati i vari parametri per il ranking, già moltiplicati per i rispettivi pesi, e il risultato finale.

Effettuiamo la stessa identica ricerca ponendo i pesi come di seguito riportati:

rilevanza: 90% - popolarità conversazione: 80% - popolarità utenti: 20%
attualità: 100% - densità: 10% - popolarità messaggi: 70%

I risultati vengono mostrati nelle Tabelle 5.3 e 5.4. Come è possibile vedere, l'ordinamento in questo caso risulta differente. Il primo risultato, sebbene presenti un numero di follower relativamente basso e la conversazione sia iniziata diverso tempo prima rispetto alle altre, ha un altissimo rapporto fra conversazioni iniziate e numero di risposte ricevute da parte degli utenti che vi hanno partecipato, nonché un valore altrettanto alto di rilevanza del testo. Questo fa di essa una conversazione ritenuta più importante di altre più recenti e con più risposte, anche in base ai pesi assegnati in fase di ricerca.

Come ultima prova, vengono assegnati pesi completamente opposti ai valori del precedente caso, per dimostrare come i risultati possano presentare conversazioni differenti al variare significativo degli stessi. Nelle Tabelle 5.5 e 5.6 sono posti i risultati con pesi uguali ai seguenti:

rilevanza: 10% - popolarità conversazione: 1% - popolarità utenti: 100%
attualità: 0% - densità: 100% - popolarità messaggi: 5%

I risultati ricavati evidenziano che il 40% delle conversazioni, confrontate con il precedente esempio, sono diverse. Inoltre l'ordinamento dei risultati è significativamente differente rispetto a quello mostrato nel secondo caso.

	conversazione						utenti		
n	**id**	**score**	**retweet**	**inizio**	**densità**	**risposte**	**conversazioni**	**risposte**	**follower**
1	27206420583	28.533	0	13/10/2010 05:28:53	0.366	20	20	120	5893
2	1336018330460160	15.836	0	07/11/2010 19:11:46	0.238	37	3018	2953	6371478
3	29420126084	17.791	0	02/11/2010 01:11:11	0.073	9	10	50	380
4	28458596671	17.754	0	23/10/2010 03:29:58	0.486	22	12	12	37411
5	354439982485504	27.237	0	05/11/2010 02:11:19	0.746	54	1763	1934	37233
6	27809667594	18.158	0	19/10/2010 08:44:55	0.129	18	498	2351	28728
7	28607807941	29.034	0	24/10/2010 18:05:13	0.537	47	1138	1154	68593
8	27596380075	24.823	0	17/10/2010 04:12:37	0.645	58	3624	4864	12937
9	27682107741	25.465	0	18/10/2010 01:26:33	0.22	51	2496	7280	58422
10	27989333975	21.003	0	21/10/2010 04:37:29	0.271	17	1071	567	7110

Tabella 5.1: Risultati della ricerca: id della conversazione, dati relativi ad essa e all'utente iniziale.

n	**score**	**follower**	**conversazioni**	**densità**	**retweet**	**attualità**	**totale**
1	0.983	0.002	1.000	0.825	0.0	0.998	3.807
2	0.545	1.000	0.162	0.296	0.0	1.000	3.004
3	0.613	0.000	0.833	0.347	0.0	1.000	2.793
4	0.612	0.010	0.167	1.000	0.0	0.999	2.787
5	0.938	0.004	0.183	0.642	0.0	1.000	2.767
6	0.625	0.009	0.787	0.321	0.0	0.999	2.74
7	1.000	0.009	0.169	0.530	0.0	0.999	2.706
8	0.855	0.001	0.224	0.517	0.0	0.999	2.596
9	0.877	0.007	0.486	0.200	0.0	0.999	2.569
10	0.723	0.002	0.088	0.711	0.0	0.999	2.524

Tabella 5.2: Risultati del calcolo del ranking sui vari parametri con pesi equivalenti a 1.

	conversazione						utenti		
n	**id**	**score**	**retweet**	**inizio**	**densità**	**risposte**	**conversazioni**	**risposte**	**follower**
1	27206420583	28.533	0	13/10/2010 05:28:53	0.366	20	20	120	5893
2	29420126084	17.791	0	02/11/2010 01:11:11	0.073	9	10	50	380
3	27809667594	18.158	0	19/10/2010 08:44:55	0.129	18	498	2351	28728
4	27682107741	25.465	0	18/10/2010 01:26:33	0.22	51	2496	7280	58422
5	28607807941	29.034	0	24/10/2010 18:05:13	0.537	47	1138	1154	68593
6	354439982485504	27.237	0	05/11/2010 02:11:19	0.746	54	1763	1934	37233
7	27596380075	24.823	0	17/10/2010 04:12:37	0.645	58	3624	4864	12937
8	1336018330460160	15.836	0	07/11/2010 19:11:46	0.238	37	3018	2953	6371478
9	27273610678	19.973	0	13/10/2010 22:25:17	0.635	56	3392	4552	12507
10	25539744632	11.984	0	26/09/2010 01:27:27	0.109	24	400	1276	438089

Tabella 5.3: Risultati della ricerca: id della conversazione, dati relativi ad essa e all'utente iniziale.

n	**score**	**follower**	**conversazioni**	**densità**	**retweet**	**attualità**	**totale**
1	88.445	0.033	80.000	8.246	0.0	99.828	276.553
2	55.149	0.005	66.667	3.475	0.0	99.961	225.256
3	56.286	0.180	62.945	3.206	0.0	99.869	222.487
4	78.937	0.134	38.889	2.000	0.0	99.861	219.821
5	90.000	0.170	13.521	5.296	0.0	99.906	208.893
6	84.429	0.081	14.627	6.419	0.0	99.982	205.538
7	76.947	0.026	17.896	5.171	0.0	99.855	199.895
8	49.087	20.000	12.980	2.964	0.0	100.000	185.031
9	61.912	0.026	17.893	5.268	0.0	99.833	184.932
10	37.147	2.090	42.533	2.065	0.0	99.713	183.549

Tabella 5.4: Risultati del calcolo del ranking sui vari parametri con pesi differenti.

	conversazione						utenti		
n	**id**	**score**	**retweet**	**data inizio**	**densità**	**n risposte**	**conversazioni**	**risposte**	**follower**
1	1336018330460160	15.836	0	07/11/2010 19:11:46	0.238	37	3018	2953	6371478
2	28458596671	17.754	0	23/10/2010 03:29:58	0.486	22	12	12	37411
3	27206420583	28.533	0	13/10/2010 05:28:53	0.366	20	20	120	5893
4	27989333975	21.003	0	21/10/2010 04:37:29	0.271	17	1071	567	7110
5	29653356801	10.7	0	04/11/2010 10:26:46	0.246	15	842	582	2568
6	354439982485504	27.237	0	05/11/2010 02:11:19	0.746	54	1763	1934	37233
7	28607807941	29.034	0	24/10/2010 18:05:13	0.537	47	1138	1154	68593
8	27789595546	13.125	0	19/10/2010 03:22:17	0.283	23	857	1416	6575
9	27596380075	24.823	0	17/10/2010 04:12:37	0.645	58	3624	4864	12937
10	27273610678	19.973	0	13/10/2010 22:25:17	0.635	56	3392	4552	12507

Tabella 5.5: Risultati della ricerca: id della conversazione, dati relativi ad essa e all'utente iniziale.

n	**score**	**follower**	**conversazioni**	**densità**	**retweet**	**attualità**	**totale**
1	5.454	100.000	0.162	29.643	0.0	0.0	135.260
2	6.115	0.970	0.167	100.000	0.0	0.0	107.252
3	9.827	0.167	1.000	82.458	0.0	0.0	93.453
4	7.234	0.236	0.088	71.118	0.0	0.0	78.676
5	3.685	0.096	0.115	72.725	0.0	0.0	76.622
6	9.381	0.404	0.183	64.192	0.0	0.0	74.160
7	10.000	0.852	0.169	52.961	0.0	0.0	63.983
8	4.521	0.163	0.275	55.716	0.0	0.0	60.675
9	8.550	0.131	0.224	51.714	0.0	0.0	60.619
10	6.879	0.131	0.224	52.680	0.0	0.0	59.914

Tabella 5.6: Risultati del calcolo del ranking sui vari parametri con pesi differenti.

Esempio 2

Viene effettuata ora una ricerca sulle parole “ubuntu 10.10”, includendo unicamente conversazioni con almeno una risposta. Vengono posti i pesi come di seguito riportato, in modo da ricavare le conversazioni più rilevanti per il testo come prime e lasciare per il momento da parte l'influenza degli altri ranking:

rilevanza: 100% - popolarità conversazione: 20% - popolarità utenti: 20%
attualità: 20% - densità: 5% - popolarità messaggi: 20%

I risultati forniti in questo caso dall'indice sono 32, di cui sono trattati solamente i primi cinque. Di seguito vengono riportati, in ordine di ranking totale decrescente, i testi delle varie conversazioni:

- Instalando #ubuntu 10.10 cc @and3rz0n @chicogeek
  @chicopolari Me cuentas que te parece ;)
  @CocoGinebra Ya lo haba usado y est chido :) pero ahora lo instalare en otra particin
  @chicopolari recien men?? ya va asalir el 11.04 haha xD
  @And3rz0n Es enserio?
  @chicopolari jheje si el otro ao en abril :D

- @riskand: dist-upgrading from lucid to maverick ≫ ubuntu 10.10???
  @AroonPardede yoi pakcik. mumpung sistem blm naik production
  @riskand server kaskus pake ubuntu??
  @AroonPardede bukan itu server google :p

- A alguien ms se le crashea #chrome en #ubuntu al usar twitter?
  @axoloteDeAccion Te aviso al rato, hasta ahorita yo no he tenido problemas
  @AleksAlpizar oks. Se cierra cuando quiero twittear desde el sitio de twitter.com
  @axoloteDeAccion De hecho estoy twitteando desde Chrome con Ubuntu 10.10
  @AleksAlpizar Y no se te crashea? Que onda conmigo jajaj. Puedo hacer todo, pero al hacer login a twitter despus de 10 segundos se cierra.
  @axoloteDeAccion eso es raro... unicamente te pasa con Chrome?

- uhmm creo que sale doble fresh install, y ya que estoy le doy una oportunidad a Fedora 14
  @ea00d009 fedora jeje bueno y si pruebas ubuntu 10.10?
  @merckmf tengo instalado ubuntu 10.10 xD y lo voy a sacar para probar fedora 14
  @ea00d009 oye que bien ahi nos dices que tal esta el fedora yo la verdad voy con el ubuntu

- Wow, @uupc is getting a fair amount of unity based feedback this week :)
  @popey: unity sucks / is fantastic and this is the best / worst decision by Ubuntu. That's it I'm leaving / moving to Ubuntu! Or something
  @popey: actually I am trying it out at work and at home. Some things I like, some things I don't but going to give it a few weeks
  @Xoke: I'm happy to carry on using 10.04 until they release the compizificated Unity, and give it a go, or just leave it till next year. :)
  @popey: I'm on 10.10. and unity is an apt-get away. Gotta love Ubuntu (and Debian etc etc) for that :)

Come è possibile notare, le conversazioni sono tutte rilevanti alla parola chiave ricercata, in particolare espongono problemi della distro in oggetto, consigli su di essa o semplici annotazioni da parte dei suoi utilizzatori.

Cambiamo ora i parametri come segue per ricavare le conversazioni più interessanti sulla parola chiave:

rilevanza: 100% - popolarità conversazione: 90% - popolarità utenti: 100%
attualità: 20% - densità: 5% - popolarità messaggi: 100%

I risultati ottenuti, limitati a cinque e di cui mostriamo unicamente il testo, sono i seguenti:

- I guess I was put off Android with the Nexus One. Not to say I'd never go back, but Google/manufacturers need to sort out firmware updates.
  @mbdrake I was put off Android by the HTC Desire... I'm not sure there's any going back from that. ;)
  @steveabraham The Desire is all but a Nexus One rebranded and replacing the trackball and buttons with something sturdier.
  @mbdrake Ah, I thought the Desire was a better spec. Still, wasn't impressed, and that was me comparing it to an old iPhone 3G.
  @steveabraham I had to rotate the Nexus One to get the keyboard to accept any input. That's when I had enough.
  @mbdrake I never encountered issues like that, but the whole thing just felt a bit disjointed. Plus, email was just a nasty experience.
  @steveabraham BTW, looks like we *may* have tracked down the problem relating to 'screen' and non-priv users.
  @mbdrake Ah, cool, fixable, or requires a reinstall?
  @steveabraham Should have a working solution to this. Do you mind if I log into your server and make changes as necessary?
  @mbdrake Crack on, thanks! Reboot if necessary, nothing live on the box.
  @steveabraham Just to confirm: this is steveah3 I'm working on?

@mbdrake Yep, that's the 10.10 box.
@steveabraham I'm going to give it a reboot..
@mbdrake No probs.
@steveabraham Sadly this does not appear to have worked - have referred this back to rest of the technical team..
@mbdrake I've just logged in as non-priv user and managed to run screen without any problem...
@steveabraham In my test case, I logged in as root and su'ed down to 'steve' and tried running screen. Perhaps not be way to test..
@mbdrake should be fine as a means of testing. su'ing down will drop all your privs, after all.
@nevali That's what I thought, but we think permissions for pty are still attached to root even when su'ing downwards..
@mbdrake plus, I think su must do something funny with controlling terminal permissions otherwise it wouldn't work :)
@mbdrake permissions on *your* PTY (created by SSH) != permissions on the one screen allocates :)

- Wow, @uupc is getting a fair amount of unity based feedback this week :)
  @popey: unity sucks / is fantastic and this is the best / worst decision by Ubuntu. That's it I'm leaving / moving to Ubuntu! Or something
  @popey: actually I am trying it out at work and at home. Some things I like, some things I don't but going to give it a few weeks
  @Xoke: I'm happy to carry on using 10.04 until they release the compizificated Unity, and give it a go, or just leave it till next year. :)
  @popey: I'm on 10.10. and unity is an apt-get away. Gotta love Ubuntu (and Debian etc etc) for that :)

- wow, #ubuntu 10.10 beta very impressive! Just installed #FF 4.0 beta, also nice. Running like cut cat in #virtualbox on 10.04 laptop.
  @lightweight I'm curious how much memory you're running on your virtual box with 10.10? How's the performance? @markrais 512 MB, but think it'd do fine with 256, Mark. Don't use it for much beyond testing.

- Ahora descargando Ubuntu 10.04, alguien sabe si para el porttil me bajo mejor la desktop o la netbook edition?
  @idcs la netbook es mejor para esos equipos pequeos, si es portatil normal desktop es mejor #ubuntu

- przecie? piszecie o katastrofie znamienne to stwierdzenie Bayer. Wg blondynki samo pisanie o 10.04 zwi?zane jest wida? z jazd? po bandzie.
  @Nurni11 Czy REM protestowa?a przeciwko temu o?wiadczeniu?

http://janinajankowska.salon24.pl/202515,uchwala-rady-programowej-tvp

È da notare in questo caso che il primo risultato, che ci attenderemmo il più attinente di tutti, risulti invece non esattamente incentrato sulla parola chiave ricercata. Il risultato viene però posto come primo per due motivi: prima di tutto la presenza della parola "10.10" inclusa anche nella chiave di ricerca, secondo per l'alto numero di risposte, che fa di essa una conversazione molto importante. La sua presenza o meno nei risultati è quindi strettamente legata al valore dato ai vari ranking e allo score del testo, in questa ricerca posto al 100%. Gli altri risultati, ad esclusione dell'ultimo, il quale compare unicamente per la presenza della parola "10.10", risultano invece attinenti alla ricerca effettuata.

5.2 Spunti per ulteriori sviluppi

Il sistema realizzato ha puntato molto sull'efficienza nella raccolta dei dati, in modo da ottimizzare il recupero di numerose conversazioni in tempi brevi. Non si è dunque tenuto molto conto degli argomenti dei tweet estratti, così come degli utenti trattati (sebbene, come presentato nel precedente capitolo, siano stati in origine recuperati gli utenti più popolari del *social network*). È dunque chiaramente possibile migliorare il sistema in modo che i dati estratti dal SN siano in qualche modo ritenuti più interessanti.

Un possibile esempio è quello di ricercare ed estrarre i tweet relativi ad alcune parole chiave, in modo da interrogare successivamente il sistema con tali parole e ottenere conversazioni ad esse relative. Un'altra possibile applicazione consiste nell'estrarre da Twitter l'elenco dei tag più utilizzati in un preciso istante e quindi effettuare delle ricerche su di essi. In questa maniera è possibile collezionare tweet e conversazioni relative agli argomenti più discussi, e quindi ritenuti più interessanti dal pubblico, in quel momento.

Quest'ultima interessante applicazione potrebbe essere implementata con un sistema molto simile a quello già realizzato e composto da una rete di client connessi ad un server centrale. In questo caso il server avrà il compito non di richiedere la *public timeline*, ma l'elenco dei tag più utilizzati in quel momento (si vedano le API relative ai *trends* presenti in [4]). Una volta estratta la lista, consegnerà a ciascun client un tag da ricercare ed essi, tramite le API di ricerca (si veda in [4] *search*), scaricheranno tutti i risultati relativi, memorizzandoli e ricostruendo le eventuali conversazioni così come presentato precedentemente.

Per quanto riguarda l'indicizzazione, potrà essere implementata in maniera più rigorosa. In particolare sarebbe interessante realizzare indici differenti per ciascuna delle lingue utilizzate (in maniera da rendere possibile la

ricerca solamente in una di esse) oppure sui tempi di inizio delle conversazioni (in maniera da poter effettuare ricerche correlate ai tempi di sviluppo delle stesse, ad esempio una ricerca che includa unicamente i risultati più recenti di una certa data). Queste indicizzazioni, che mirano a creare indici differenti e scollegati fra di loro, risultano facilmente ottenibili semplicemente estraendo le conversazioni dal db e inserendole nell'indice appropriato in base alle loro caratteristiche: la lingua, individuata con opportuni meccanismi, o la data. Il problema che sorge nel realizzare più indici è la difficoltà in fase di interrogazione. Nel caso infatti si decida di cercare un risultato in più di una lingua o per qualsiasi data, non avremo un solo indice in cui ricercare, ma dovremo, in base alle scelte dell'utente e/o a quelle progettuali, orientarci sull'interrogazione di uno o più di essi, quindi visualizzare i risultati in modo trasparente per l'utente. Queste ricerche prevedono dunque interrogazioni a più indici e, dai risultati da essi ottenuti, scegliere opportunamente quali di essi risultano più o meno importanti.

Infine, possibili sviluppi futuri dell'applicazione potrebbero tenere conto delle ulteriori informazioni forniteci da Twitter su utenti e messaggi. Un interessante esempio potrebbe essere quello relativo alla geo-localizzazione, un servizio che permette di associare ad ogni tweet una posizione geografica più o meno precisa relativa all'utente. Inserendo questa posizione, diventa interessante effettuare ricerche che includano solo alcune zone geografiche. Si pone ad esempio di voler ricercare una parola chiave che ha un significato particolare solamente in una zona circoscritta, che potrebbe essere ad esempio uno stato o ancor più una regione di esso. Senza considerare la provenienza geografica, potrebbero comparire fra i risultati messaggi inseriti fuori dalla nostra area di interesse e quindi non attinenti alla ricerca, poichè riguardanti altri aspetti della parola cercata. Associando quindi pesi differenti alle conversazioni in base alla loro provenienza geografica avremo dunque un nuovo metodo per ricavare le conversazioni più interessanti per l'utente.

Bibliografia

[1] Chirp. Twitter User Statistics. URL: http://www.huffingtonpost.com/2010/04/14/twitter-user-statistics-r_n_537992.html, 2010.

[2] Chirp. Twitter User Statistics 2. URL: http://www.businessinsider.com/twitter-stats-2010-4, 2010.

[3] Ryan Kelly. Twitter Study - August 2009. URL: http://www.pearanalytics.com/blog/wp-content/uploads/2010/05/Twitter-Study-August-2009.pdf, 2009.

[4] Twitter API Documentation. URL: http://dev.twitter.com, 2010.

[5] Yusuke Yamamoto. Twitter4J: unofficial Java library for the Twitter API. URL: http://twitter4j.org, 2010.

[6] Twitter. Announcing Snowflake. URL: http://engineering.twitter.com/2010/06/announcing-snowflake.html, June 1, 2010.

[7] Christian Middleton and Ricardo Baeza-Yates. A Comparison of Open Source Search Engines. 2006.

[8] Matteo Magnani, Danilo Montesi, and Luca Rossi. Conversation retrieval in social network sites, comunicazione privata. 2010.

[9] Mysql. URL: http://www.mysql.com.

[10] Lucene. URL: http://lucene.apache.org.

Appendice A

Codici sorgente

A.1 Server

Classe per la realizzazione del server nell'implementazione mostrata in Figura 4.1. Il server ha il compito di distribuire ai client gli id degli utenti da trattare. Questi id vengono estratti o da un apposito file oppure ricavati direttamente dalla *public timeline*.

Nella prima parte del programma viene assicurato l'accesso, tramite OAuth, al servizio API di Twitter. Viene dunque verificato se è necessario leggere il file contenenti gli id o meno (segnalato all'avvio dello script con l'inserimento del parametro *-f*). Nel caso vada utilizzato il file, da esso viene letto ogni singolo id e assegnato ai client che ne fanno richiesta. Nel caso il file non vada letto, oppure sono stati utilizzati tutti gli id in esso presenti, il server effettua una richiesta per la *public timeline*, estrae da essa gli id utente e li consegna ai client. Il server si occupa anche di verificare che un utente non sia stato recentemente trattato.

```
/*******************************************
 * ConvRetrieval : Server
 */

import java.io.*;
import java.net.*;
import java.sql.ResultSet;
import java.util.Vector;

import twitter4j.RateLimitStatus;
import twitter4j.ResponseList;
import twitter4j.Status;
import twitter4j.Twitter;
import twitter4j.TwitterException;
import twitter4j.TwitterFactory;
import twitter4j.User;
import twitter4j.http.AccessToken;
import twitter4j.http.RequestToken;
```

```
public class Server extends Thread
{
   /* Variabili per connessione server-client */
   private static ServerSocket Server;
   URLConnection conn;
   static int port;          /* porta per connessioni entranti */

   /* Variabili per oggetti twitter e database */
   static Twitter twitter;   /* Oggetto twitter per Twitter4J */
   static boolean use_file;  /* true per usare file celebrita' */
   Database db;              /* DB connection */

   /**
    * Metodo principale per autenticazione alle API e l'avvio al ↩
      recupero degli id utente
    * @param args[0] porta del server su cui attende le connessioni ↩
      dei client
    * @param args[1] -f: se presente, non legge il file degli id ↩
      celebrita'
    */
   public static void main(String args[]) throws Exception
   {
      /* Impostazioni per autenticazione OAuth */
      RequestToken requestToken;
      AccessToken accessToken = null;
      String consumer_key = "wxu9PvPEVnf1k9QuSr9liw";
      String consumer_secret = "private_key";

      /* Legge da input impostazioni porta connessioni e
       * uso del file celebrita' */
      port = Integer.parseInt(args[0]);
      use_file = true;
      if( args.length > 1 )
      {
         if( args[1].equals("-f") )
            use_file = false;
      }

      System.out.println("Twitter Conversation Retrieval Server ↩
          avviato");

      /* Si autentica con OAuth per le richieste alle Twitter API */
      twitter = new TwitterFactory().getInstance();
      twitter.setOAuthConsumer(consumer_key, consumer_secret);
      requestToken = twitter.getOAuthRequestToken();
      BufferedReader br = new BufferedReader(
                           new InputStreamReader(System.in) );

      while( accessToken == null )
      {
         System.out.println("Accedi al seguente URL e permetti l'↩
             accesso al tuo account:");
         System.out.println(requestToken.getAuthorizationURL());
         System.out.print("Inserisci il PIN: ");
         String pin = br.readLine();

         try {
            if(pin.length() > 0)
               accessToken = twitter.getOAuthAccessToken(↩
                   requestToken, pin);
            else
               accessToken = twitter.getOAuthAccessToken();
```

```
        } catch (TwitterException te) {
            if(401 == te.getStatusCode())
                System.out.println("Impossibile accedere.");
            else
                te.printStackTrace();
        }
    }

    Server = new ServerSocket(port);
    System.out.println("In attesa sulla porta " + port);

    new Server();
}


/*
 * Metodo per il recupero degli id utente
 */
public Server() throws Exception
{
    /* Variabili per lettura file con id user */
    String file = "celebrities_id.txt";
    FileInputStream fstream = new FileInputStream(file);
    BufferedReader br = new BufferedReader(
          new InputStreamReader( new DataInputStream(fstream) ) );

    /* id utente */
    String idUser;

    /* Vettore contenente gli id utente da utilizzare */
    Vector<User> id_users = new Vector<User>();

    /* False per assegnare l'id al client */
    boolean continue_user;

    /* Variabile per intervallo degli aggiornamenti user */
    /* default 2 giorni: 1000msec x 60sec x 60min x 24h x 2day */
    int interval_user = 1000*60*60*24*2;

    /* Variabili per connessione database */
    String IP_db = "194.116.73.79";
    String db = "twitter";
    String user = "user";
    String password = "password";

    /* Apre la connessione con il database */
    db = new Database(IP_db, db, user, password);
    if( !db.open() )
        System.out.println("Errore connessione database: "
                           + db.getErrore());

    /* Ciclo per ricavare e consegnare gli id utente */
    while(true)
    {
        try {
            /* In attesa di una connessione da parte dei client */
            Socket client = Server.accept();
            System.out.println("* Connessione accettata da: "
                               + client.getInetAddress());
            continue_user = true;

            /* Primo passaggio: recupera gli utenti piu' seguiti di ↩
```

```
    Twitter, leggendoli da file. Se sono ancora presenti↩
     id, li utilizza */
if( use_file && ((idUser = br.readLine()) != null) )
{
 /* Verifica gli id alla ricerca del primo disponibile*/
   do {
      System.out.println("ID utente: " + idUser);

      /* Controlla se e' possibile continuare (sono ↩
          presenti ancora richieste) */
      controlContinue(twitter);

      /* Verifica (leggendo il DB) che l'utente non sia ↩
          gia' stato scaricato recentemente: in questo ↩
          caso e' possibile che numerosi tweet siano gia↩
          ' stati estratti, quindi e' meglio saltarlo ↩
          per risparmiare request. */

      /* Recupera l'ora dell'ultimo update utente */
      ResultSet rs = db.executeQuery("SELECT last_update↩
          FROM user WHERE id='" + idUser + "'");
      if( rs!=null && rs.next() )
      {
         /* Se l'utente e' presente, verifica se e' ↩
             aggiornato piu' tardi di 2 giorni */
         if (System.currentTimeMillis()-rs.getTimestamp↩
             (1).getTime() >= interval_user)
         {
            /* Utente gia' presente, ma non aggiornato ↩
                da molto: aggiorna i suoi dati (tutto ↩
                tranne l'id) ed esce dal ciclo di ↩
                ricerca */

            /* Richiede le informazioni dell'utente */
            User user = twitter.showUser(Integer.↩
                parseInt(idUser));

            /* Esegue l'update delle sue informazioni */
            db.executeUpdate("UPDATE user SET name='" + ↩
                formatString(user.getName()) + "', ↩
                screen_name='" + formatString(user.↩
                getScreenName()) + "', " + "image='"+↩
                user.getProfileImageURL() + "', ↩
                followers='" + user.getFollowersCount() ↩
                + "', last_update=now() " + "WHERE id='"↩
                 + user.getId() + "'");

            /* Pone a false per indicare che e' stato ↩
                trovato l'id */
            continue_user = false;
         }
      }
      else
      {
         /* Utente non presente: lo aggiunge al DB ed ↩
             esce dal ciclo di ricerca */

         /* Richiede le informazioni dell'utente */
         User user = twitter.showUser(Integer.parseInt(↩
             idUser));

         db.executeUpdate("INSERT INTO user (id, name, ↩
```

```
                screen_name, image, followers, last_update)↩
                 " + "VALUES ('"+idUser+"', '"+formatString↩
                (user.getName())+"', '"+formatString(user.↩
                getScreenName())+"', '"+user.↩
                getProfileImageURL()+"', '"+user.↩
                getFollowersCount()+"', now())");

            continue_user = false;
        }

    } while(continue_user
            && (idUser = br.readLine()) != null);

    /* Se e' stato trovato un ID, avvia un thread per ↩
        estrarre i tweet dell'utente */
    if ( idUser != null )
    {
        Connect c = new Connect(client, Integer.parseInt(↩
            idUser));
    }
    else
    {
        /* File id terminati: passa a utilizzare la public↩
            timeline */
        use_file = false;
    }
}
else
{
    /* File id da non usare o terminato: passa a ↩
        utilizzare la public timeline */
    use_file = false;
}


/* Se non usiamo il file (o lo abbiamo terminato), ↩
    estrae gli utenti dalla public timeline */
if( !use_file )
{
    do {
        /* Se non sono ancora stati scaricati gli id o ↩
            sono gia' finiti, ne ricava di nuovi */
        while( id_users.size() == 0 )
        {
            /* Controlla se e' possibile continuare */
            controlContinue(twitter);

            /* Recupera la public timeline */
            ResponseList<Status> statuses =
                            twitter.getPublicTimeline();

            System.out.println("Public timeline:");
            for (Status status : statuses)
            {
                System.out.println("- [" +status.getUser().↩
                    getId()+"] "+ status.getUser().getName()↩
                     + ": " + status.getText().replaceAll("\↩
                    n", " "));

                /* Inserisce gli id utente nel vettore ↩
                    id_users */
                id_users.add(status.getUser());
```

```
            }
        }

        /* Inizia ad esaminare il primo utente in lista */
        /* Verifica (leggendo il DB) che l'utente non sia gia' stato scaricato recentemente */

        /* Recupera l'ora dell'ultimo update utente */
        ResultSet rs = db.executeQuery("SELECT last_update FROM user WHERE id='"+id_users.get(0).getId()+"'");
        if( rs!=null && rs.next() )
        {
            /* se e' presente l'utente verifica se e' aggiornato piu' tardi di 2 giorni */
            if (System.currentTimeMillis()-rs.getTimestamp(1).getTime()<interval_user)
            {
                /* Utente aggiornato recentemente: lo elimina e passa all'utente successivo */
                id_users.remove(0);
            }
            else
            {
                /* Utente gia' presente, ma non aggiornato da molto: aggiorna i suoi dati (tutto tranne l'id) ed esce dal ciclo di ricerca */
                db.executeUpdate("UPDATE user SET name='"+formatString(id_users.get(0).getName())+"', screen_name='"+formatString(id_users.get(0).getScreenName())+"', " + "image='"+id_users.get(0).getProfileImageURL()+"', followers='"+id_users.get(0).getFollowersCount()+"', last_update=now() " + "WHERE id='"+id_users.get(0).getId()+"'");

                continue_user = false;
            }
        }
        else
        {
            /* Utente non presente: lo aggiunge al DB ed esce dal ciclo di ricerca */
            db.executeUpdate("INSERT INTO user (id, name, screen_name, image, followers, last_update) " +
                "VALUES ('"+id_users.get(0).getId()+"', '"+formatString(id_users.get(0).getName())+"', '"+formatString(id_users.get(0).getScreenName())+"', '"+id_users.get(0).getProfileImageURL()+"', '"+id_users.get(0).getFollowersCount()+"', now())");

            continue_user = false;
        }

    } while( continue_user );
```

```
                /* Avvia un thread per il singolo client */
                Connect c = new Connect(client, id_users.get(0).getId↩
                    ());

                /* Rimuove l'id utente appena utilizzato */
                id_users.remove(0);
            }
        } catch(Exception e) {
            e.printStackTrace();
        }
    }
}


/* Funzione per controllare se e' possibile effettuare nuove ↩
    richieste o se occorre attendere */
private void controlContinue(Twitter twitter) throws Exception
{
    /* Numero di richieste minime da preservare */
    int remaining_hits = 3;

    /* Richiede il numero di req ancora disponibili per l'IP */
    RateLimitStatus status = twitter.getRateLimitStatus();
    System.out.println("^ Richieste rimanenti: "
                       + status.getRemainingHits());

    /* Controlla se le richieste sono in numero sufficiente */
    if( status.getRemainingHits() < remaining_hits )
    {
        /* Richieste rimanenti insufficienti */
        System.out.println("Le richieste rimanenti sono poche: "
                    + status.getRemainingHits()
                    + " - Rimarr  in dormiveglia per: "
                    + (status.getSecondsUntilReset()+5) +" sec");

        /* Il thread rimane in pausa per i secondi necessari al ↩
            reset delle request (+5sec di scarto) */
        Thread.sleep( (twitter.getRateLimitStatus().↩
            getSecondsUntilReset() * 1000) + (5 * 1000) );
    }
}


/* Funzione per sistemare testo con caratteri particolari */
public static String formatString(String txt)
{
    String text = txt.replace("\\","\\\\").replace("\n"," ");
    text = text.replace("\t"," ").replace("'","\\'");
    text = text.replace("\"","\\\"").trim();

    return text;
}
}


/* Classe per la gestione dei singoli client */
class Connect extends Thread
{
    /* Variabili per connessione server-client */
    URLConnection conn;
    private Socket client = null;
    BufferedReader in = null;
```

```
    PrintStream out = null;

    long id_user;

    /*
     * Metodo principale per la connessione con il client
     * @param clientSocket socket da utilizzare
     * @param id_u id utente da assegnare al client
     */
    public Connect(Socket clientSocket, int id_u)
    {
        id_user = id_u;
        client = clientSocket;

        try
        {
            /* Crea i canali input e output */
            in = new BufferedReader( new InputStreamReader( client.↩
                getInputStream() ) );
            out = new PrintStream(client.getOutputStream(), true);
        }
        catch(Exception e1)
        {
            try { client.close(); }
            catch(Exception e) { System.out.println(e.getMessage()); }
            return;
        }

        /* Avvia il client per l'invio dell'id utente */
        this.start();
    }


    public void run()
    {
        /* Invia l'id al client */
        System.out.println("* Richiesta da effettuare per "
                           + client.getInetAddress() + ": "
                           + id_user);

        out.println(id_user);
        out.flush();

        /* Chiude la connessione e termina il client */
        try {
            out.close();
            in.close();
            client.close();
        } catch (Exception e) {
            e.printStackTrace();
        }
    }
}
```

A.2 Client

Classe per la realizzazione del client nell'implementazione mostrata in Figura 4.1. Il client ha il compito di estrarre la timeline dell'id utente assegnatogli dal server e ricostruire eventuali conversazioni.

Nella prima parte del programma viene richiesto al server l'id utente da trattare. Il client effettuerà una richiesta al database per estrarre gli eventuali tweet già inseriti per esso, quindi richiederà tramite API la *public timeline* dell'utente. Scorrerà ogni tweet di essa e verificherà se sono in risposta o meno ad altri messaggi. Nel caso non lo siano, e non siano già presenti nel db, effettuerà il loro inserimento. Nel caso siano invece in risposta, recupererà i tweet precedenti fino a ricreare l'intero ramo della conversazione. Terminata la *public timeline*, chiederà al server un nuovo id utente da trattare.

```
/*******************************************
 * ConvRetrieval : Client
 */

import java.io.*;
import java.net.*;
import java.sql.Connection;
import java.sql.ResultSet;
import java.sql.Statement;
import java.sql.SQLException;
import java.text.DateFormat;
import java.text.SimpleDateFormat;
import java.util.Calendar;
import java.util.Date;
import java.util.StringTokenizer;
import java.util.Vector;
import java.util.ArrayList;

import twitter4j.Paging;
import twitter4j.RateLimitStatus;
import twitter4j.ResponseList;
import twitter4j.Status;
import twitter4j.Twitter;
import twitter4j.TwitterFactory;


public class Client
{
   static int port;
   static String search, server;

   /* DB connection */
   static Database db;

   /**
    * Metodo principale per il recupero dei tweet
    * @param args[0] porta tramite cui connettersi al server
    * @param args[1] IP del server
    */
   public static void main(String[] args) throws SQLException
```

```
{
   /* Legge da input impostazioni della porta connessioni e ↩
       indirizzo del server */
   port = Integer.parseInt(args[0]);
   server = args[1];

   System.out.println("Conversation Retrieval Client avviato");

   BufferedReader in = null;
   Socket socket = null;

   /* Variabili per connessione database */
   String IP_db = "194.116.73.79";
   String db = "twitter";
   String user = "user";
   String password = "password";

   /* Apre la connessione con il database */
   db = new Database(IP_db, db, user, password);
   if( !db.open() )
      System.out.println("Errore connessione database: "
                         + db.getErrore());

   while(true)
   {
      try
      {
         /* Instaura la connessione con il server */
         socket = new Socket(server, port);
         in = new BufferedReader(
                new InputStreamReader(socket.getInputStream()) );
      }
      catch(Exception e) {
         System.out.println( e.getMessage() );
         break;
      }

      try {
         /* Legge dal server lo user id da cercare */
         search = in.readLine();
         System.out.println("* ID Utente : " + search);

         if( search != null && search != "" )
         {
            /* Recupera la timeline dell'utente */
            Twitter twitter = new TwitterFactory().getInstance();
            int id_user = Integer.parseInt(search);
            int count_tweet = 0; /* Totale retweet            */
            int tot_n_story = 0; /* Totale conversazioni e    */
            int tot_n_reply = 0; /* reply da aggiungere a user */

            /* Recupera la timeline dell'utente selezionato */
            ResponseList<Status> statuses;
            try {
               /* Controlla sia possibile fare nuove request */
               controlContinue(twitter);

               /* Paginazione per richiedere gli ultimi tweet ↩
                   dell'utente */
               Paging paging = new Paging(1, 200);
               statuses = twitter.getUserTimeline(id_user,
                                                  paging);
```

```
                    /* Ricava tutti i tweet dell'utente gia' presenti ↩
                        nel DB */
                    ResultSet rs = db.executeQuery("SELECT id FROM ↩
                        tweet WHERE id_user='"+id_user+"'");
                    ArrayList<String> tweet_user =
                                        new ArrayList<String>();
                    while( rs!=null && rs.next() )
                    {
                       tweet_user.add(rs.getLong("id")+"");
                    }

                    System.out.println("Timeline utente:");
                    for (Status status : statuses)
                    {

System.out.println("- ["+status.getUser().getId()+"] " +
   status.getUser().getName()+":" +
   " ["+status.getId()+"] "+formatString(status.getText()) +
   " [reply: "+status.getInReplyToStatusId()+"]" +
   " [retweet: " + status.getRetweetCount()+"]");

/* Verifica che il tweet non sia gia' presente: se non lo e', lo ↩
    inserisce e ricostruisce l'eventuale conversazione */
boolean trovato = false;
for(int i=0;i<tweet_user.size();i++)
{
   if( tweet_user.get(i).equals(status.getId()) )
   {
      trovato = true;
      System.out.println("Tweet gia' presente nel DB: "
                         + status.getId());
   }
}

if( !trovato )
{
   /* Tweet non presente: lo inserisce */
   tweet_user.add( status.getId() );

   /* Controlla se il tweet fa parte di una discussione */
   if( status.getInReplyToStatusId() != -1 )
   {
      /* Il tweet e' in risposta a un altro: ricostruiamo la ↩
          discussione */

      /* numero reply alla discussione */
      int count_reply = 0;
      /* numero retweet totali */
      int n_retweet = 0;
      /* densita' della discussione */
      double time_density = 0;
      /* testo completo della discussione */
      String text = "";
      /* copia dello stato di partenza */
      Status origin_status = status;
      /* status del padre del tweet */
      Status father_status = null;
      /* id del padre della discussione */
      long father_origin = 0;
      /* id dell'utente iniziale della discussione */
      long id_user_story = 0;
```

```
/* true se il padre e' gia' nel DB */
boolean father_is_present = false;
/* false per dire che e' stata terminata la ricerca */
boolean parent_continue = true;
/* id di tutti i reply */
Vector<String> status_child = new Vector<String>();
/* id dell'utente precedente */
long previous_user = 0;
/* stringa per inserimento tweet conversazione */
String query_tweet_c = "";

/* Variabile per intervallo degli aggiornamenti user */
/* default 2 giorni: 1000msec x 60sec x 60min x 24h x 2day */
int interval_user = 1000*60*60*24*2;

/* Ricerca il tweet padre e gli antenati ricorsivamente */
do {
   /* Controlla che sia possibile fare nuove richieste */
   controlContinue(twitter);

   /* Verifica (leggendo il DB) che il padre non sia gia' ↩
       stato scaricato: in tal caso recupera il padre del ↩
       padre e aggiorna la discussione, altrimenti continua a ↩
       cercare fino al superiore */
   ResultSet rs4 = db.executeQuery("SELECT id, origin, time ↩
       FROM tweet WHERE id='"+status.getInReplyToStatusId()+"'↩
       ");
   if( rs4!=null && rs4.next() )
   {
     /* Antenato gia' presente nel DB: ricava l'id del padre ↩
         superiore a tutti (il tweet iniziale della ↩
         discussione), ricavato dal tweet padre del presente (↩
         che ha gia' memorizzato nel campo "origin" il padre ↩
         della discussione) */
     int count_reply_2 = count_reply + 1;

     /* Recupera le informazioni da aggiornare della ↩
         discussione */
     ResultSet rs3 = db.executeQuery("SELECT reply, text, ↩
         n_retweet, time_density FROM story WHERE id_start='"+↩
         rs4.getLong("origin")+"' ");

     if( rs3!=null && rs3.next() )
     {
        /* Aumenta il numero di figli della discussione ↩
            sommando i nuovi ai vecchi */
        count_reply += rs3.getInt("reply") + 1;
        n_retweet += rs3.getInt("n_retweet");

        /* Somma alla vecchia densita' la nuova differenza fra↩
            i due status */
        DateFormat formatter =
             new SimpleDateFormat("yyyy-MM-dd HH:mm:ss");
        double last_density =
             differenceDate(rs4.getTimestamp("time"),
             (Date)formatter.parse(formatDate(status.↩
                 getCreatedAt().toString())) );
        time_density +=
             rs3.getDouble("time_density")+last_density;

        /* Aggiorna il testo complessivo */
        text = formatString(rs3.getString("text")) + " " + ↩
```

```
                formatString(status.getText()) +" "+formatString(↩
                text);

            /* Aggiorna la discussione con il corretto numero di ↩
                reply e il tempo di ultima risposta */
            db.executeUpdate("UPDATE story SET reply='"+↩
                count_reply+"', text='"+text+"', time_last_post='"↩
                +formatDate(origin_status.getCreatedAt().toString↩
                ())+"', " +"n_retweet='"+n_retweet+"', ↩
                time_density='"+time_density+"' WHERE id_start='"+↩
                rs4.getLong("origin")+"'");

            /* Assegniamo a father_origin l'id del padre dell'↩
                intera discussione */
            father_origin = rs4.getLong("origin");
            father_is_present = true;
        }
        else
        {
            father_origin = 0;
            father_is_present = true;
        }

        /* Aggiorna le info sull'utente iniziale della conversaz.↩
            */
        /* Cerca l'id_user del tweet iniziale della conversaz. */
        ResultSet rs_user2 = db.executeQuery("SELECT id_user FROM↩
            tweet WHERE id='"+father_origin+"' ");
        if( rs_user2!=null && rs_user2.next() )
        {
            /* Aggiunge il numero di nuovi reply all'utente */
            db.executeUpdate("UPDATE user SET n_reply=n_reply+"+↩
                count_reply_2+" WHERE id='"+rs_user2.getLong("↩
                id_user")+"' ");
        }

        /* setta a false la variabile per continuare */
        parent_continue = false;
    }
    else
    {
        /* Tweet non ancora presente nel DB */
        try {
            /* ricava il padre del tweet esaminato */
            father_status = twitter.showStatus(status.↩
                getInReplyToStatusId());

            if(t) System.out.println("- [" + father_status.getUser↩
                ().getId() + "] " + father_status.getUser().↩
                getName() + ": "
                + "[" + father_status.getId() + "] " + ↩
                    father_status.getText().replaceAll("\n", " ").↩
                    replaceAll("'","\'")
                + " [reply to: " + father_status.↩
                    getInReplyToStatusId() + "]");


            /* Aggiorna il numero di reply alla discussione, il ↩
                numero di retweet totali e la densita' */
            if( father_status.getRetweetCount()>0 )
                n_retweet += father_status.getRetweetCount();
            count_reply++;
```

```
/* Calcola la nuova densita' della discussione (↩
    differenza fra tempo figlio e tempo padre) */
time_density += differenceDate(father_status.↩
    getCreatedAt(), status.getCreatedAt());

/* Aggiorna il testo della discussione */
text = father_status.getText().replaceAll("\n", " ") +↩
    " " + text;

/* Copia lo stato del padre nello stato attuale */
status = father_status;

/* Inserisce il tweet nel DB: il padre generale per ↩
    ora non e' settato, lo faremo alla fine dell'↩
    intero ciclo */
if( status.getRetweetCount() >= 0 )
   query_tweet_c += "('"+status.getId()+"', '"+status.↩
       getUser().getId()+"', '"+formatString(status.↩
       getText())+"', '"+formatDate(status.↩
       getCreatedAt().toString())+"', '"+status.↩
       getInReplyToStatusId()+"', 'twt-origin-?-6tr32↩
       ', '"+status.getRetweetCount()+"'), ";
else
   query_tweet_c += "('"+status.getId()+"', '"+status.↩
       getUser().getId()+"', '"+formatString(status.↩
       getText())+"', '"+formatDate(status.↩
       getCreatedAt().toString())+"', '"+status.↩
       getInReplyToStatusId()+"', 'twt-origin-?-6tr32↩
       ', '0'), ";
status_child.add( String.valueOf( status.getId() ) );

/* Salviamo le informazioni di questo nuovo utente (se↩
    non e' l'utente iniziale della timeline ne il ↩
    penultimo) */
if( status.getUser().getId() != origin_status.getUser↩
    ().getId() && status.getUser().getId() != ↩
    previous_user )
{
   /* Salva l'id dell'utente, per evitare nei botta e ↩
       risposta di cercare sempre info sul secondo ↩
       utente */
   previous_user = status.getUser().getId();

   /* Controlla se l'utente e' gia' presente nel DB */
   ResultSet rs_user = db.executeQuery("SELECT ↩
       last_update FROM user WHERE id='"+status.↩
       getUser().getId()+"'");
   if( rs_user!=null && rs_user.next() )
   {
      /* Utente gia' presente: nel caso abbia dati ↩
          vecchi, li aggiorniamo */
      if (System.currentTimeMillis()-rs_user.↩
          getTimestamp(1).getTime() >= interval_user)
      {
         /* Utente gia' presente, ma non aggiornato da↩
             molto: aggiorna i suoi dati (tutto ↩
            tranne l'id) */
         db.executeUpdate("UPDATE user SET name='"+↩
            formatString(status.getUser().getName())+↩
            "', screen_name='"+formatString(status.↩
            getUser().getScreenName())+"', image='"+↩
```

```
                        status.getUser().getProfileImageURL()+"',↩
                         followers='"+status.getUser().↩
                        getFollowersCount()+"', last_update=now()↩
                         WHERE id='"+status.getUser().getId()+"'"↩
                        );
                }
            }
            else
            {
                /* Utente non presente: lo aggiungiamo al DB */
                db.executeUpdate("INSERT INTO user (id, name, ↩
                    screen_name, image, followers, last_update) ↩
                    VALUES ('"+status.getUser().getId()+"', '"+↩
                    formatString(status.getUser().getName())+"',↩
                     '"+formatString(status.getUser().↩
                    getScreenName())+"', '"+status.getUser().↩
                    getProfileImageURL()+"', '"+status.getUser()↩
                    .getFollowersCount()+"', now())");
            }
        }
        else if( status.getUser().getId() == origin_status.↩
            getUser().getId() )
        {
            /* Inserisce il nuovo tweet nell'array contenente i↩
                tweet dell'utente in esame */
            tweet_user.add(status.getId());
        }

        /* salva l'attuale id come il padre di tutti: nel caso↩
            verra' sovrascritto successivamente */
        father_origin = status.getId();
        id_user_story = status.getUser().getId();

    } catch (Exception e) {
        /* Errore nel recupero di un tweet (ad esempio e' ↩
            settato privato e quindi non accessibile) */
        System.out.println("Errore nel download dei padri.");

        /* imposta la discussione al meglio, limitandola all'↩
            ultimo tweet ricavato come padre */
        father_origin = status.getId();
        id_user_story = status.getUser().getId();

        /* setta a false la variabile per continuare, in modo ↩
            che finisca questo ciclo */
        parent_continue = false;
    }
  }
} while(parent_continue && father_status.getInReplyToStatusId()↩
    != -1);


if( !status_child.isEmpty() )
{
  /* Se il padre non ha gia' la sua discussione, ne crea una ↩
      nuova */
  if( !father_is_present )
  {
    /* Crea il testo completo (tutti i post concatenati) */
    text = text + " " + origin_status.getText().replaceAll("\↩
        n", " ");
```

```
            /* Inserisce la nuova discussione nel DB */
            db.executeUpdate("INSERT DELAYED INTO story (id_start, ↩
                reply, text, time_first_post, time_last_post, ↩
                n_retweet, time_density) VALUES ('"+father_origin+"',↩
                 '"+count_reply+"', '"+formatString(text)+"', '"+↩
                formatDate(status.getCreatedAt().toString())+"', '"+↩
                formatDate(origin_status.getCreatedAt().toString())+"↩
                ', '"+n_retweet+"', '"+time_density+"')");

            /* Aggiorna le info sull'utente iniziale della ↩
                discussione */
            db.executeUpdate("UPDATE user SET n_story=n_story+1, ↩
                n_reply=n_reply+"+count_reply+" WHERE id='"+↩
                id_user_story+"' ");
        }

        if( !status_child.isEmpty() )
        {
            query_tweet_c = query_tweet_c.replaceAll( "', $", "" );
            query_tweet_c = query_tweet_c.replace( "twt-origin-?-6↩
                tr32", father_origin+"" );

            db.executeUpdate("INSERT DELAYED INTO tweet (id, id_user,↩
                 text, time, reply_to, origin, n_retweet) VALUES "+↩
                query_tweet_c);
        }
    }

    /* Inserisce il tweet da cui e' iniziato tutto nel DB */
    if( father_origin == origin_status.getId() )
    {
        /* se vi e' stato un problema nel recupero del padre, setta ↩
            a -1 il reply del tweet */
        if( origin_status.getRetweetCount() >= 0 )
        {
            db.executeUpdate("INSERT DELAYED INTO tweet (id, id_user,↩
                 text, time, reply_to, origin, n_retweet) VALUES ('"+↩
                origin_status.getId()+"', '"+origin_status.getUser().↩
                getId()+"', '"+formatString(origin_status.getText())+↩
                "', '"+formatDate(origin_status.getCreatedAt().↩
                toString())+"', '-1', '"+father_origin+"', '"+↩
                origin_status.getRetweetCount()+"')");
        } else {
            db.executeUpdate("INSERT DELAYED INTO tweet (id, id_user,↩
                 text, time, reply_to, origin, n_retweet) VALUES ('"+↩
                origin_status.getId()+"', '"+origin_status.getUser().↩
                getId()+"', '"+formatString(origin_status.getText())+↩
                "', '"+formatDate(origin_status.getCreatedAt().↩
                toString())+"', '-1', '"+father_origin+"', '0')");
        }

        /* Inserisce la nuova discussione nel DB */
        if( origin_status.getRetweetCount() >= 0 )
        {
            db.executeUpdate("INSERT DELAYED INTO story (id_start, ↩
                reply, text, time_first_post, time_last_post, ↩
                n_retweet, time_density) VALUES ('"+origin_status.↩
                getId()+"', '0', '"+formatString(origin_status.↩
                getText())+"', '"+formatDate(origin_status.↩
                getCreatedAt().toString())+"','"+formatDate(↩
                origin_status.getCreatedAt().toString())+"', '"+↩
                origin_status.getRetweetCount()+"', '0')");
```

```
            } else {
                db.executeUpdate("INSERT DELAYED INTO story (id_start, ↩
                    reply, text, time_first_post, time_last_post, ↩
                    n_retweet, time_density) VALUES ('"+origin_status.↩
                    getId()+"', '0', '"+formatString(origin_status.↩
                    getText())+"', '"+formatDate(origin_status.↩
                    getCreatedAt().toString())+"','"+formatDate(↩
                    origin_status.getCreatedAt().toString())+"', '0', ↩
                    '0')");
            }

            /* Aumenta di uno le conversazioni dell'utente iniziale ↩
                della discussione */
            tot_n_story += 1;
        }
        else
        {
            /* altrimenti salva normalmente il tweet */
            if( origin_status.getRetweetCount() >= 0 )
            {
                db.executeUpdate("INSERT DELAYED INTO tweet (id, id_user,↩
                     text, time, reply_to, origin, n_retweet) VALUES ('"+↩
                    origin_status.getId()+"', '"+origin_status.getUser().↩
                    getId()+"', '"+formatString(origin_status.getText())+↩
                    "', '"+formatDate(origin_status.getCreatedAt().↩
                    toString())+"', '"+origin_status.getInReplyToStatusId↩
                    ()+"', '"+father_origin+"', '"+origin_status.↩
                    getRetweetCount()+"')");
            } else {
                db.executeUpdate("INSERT DELAYED INTO tweet (id, id_user,↩
                     text, time, reply_to, origin, n_retweet) VALUES ('"+↩
                    origin_status.getId()+"', '"+origin_status.getUser().↩
                    getId()+"', '"+formatString(origin_status.getText())+↩
                    "', '"+formatDate(origin_status.getCreatedAt().↩
                    toString())+"', '"+origin_status.getInReplyToStatusId↩
                    ()+"', '"+father_origin+"', '0')");
            }
        }

        System.out.println("- Numero reply alla discussione: "+↩
            count_reply);
    }
    else
    {
        /* Il tweet non e' in risposta (ma potrebbe essere il padre di ↩
            una nuova discussione): lo memorizziamo come singolo, ↩
            contando eventuali retweet */

        /* Inserisce il tweet nel DB */
        if( status.getRetweetCount() >= 0 )
        {
            db.executeUpdate("INSERT DELAYED INTO tweet (id, id_user, ↩
                text, time, reply_to, origin, n_retweet) VALUES ('"+↩
                status.getId()+"', '"+status.getUser().getId()+"', '"+↩
                formatString(status.getText())+"', '"+formatDate(status.↩
                getCreatedAt().toString())+"', '-1', '"+status.getId()+"↩
                ', '"+status.getRetweetCount()+"')");
         } else {
            db.executeUpdate("INSERT DELAYED INTO tweet (id, id_user, ↩
                text, time, reply_to, origin, n_retweet) VALUES ('"+↩
                status.getId()+"', '"+status.getUser().getId()+"', '"+↩
                formatString(status.getText())+"', '"+formatDate(status.↩
```

```
            getCreatedAt().toString())+"', '-1', '"+status.getId()+"↩
            ', '0')");
    }

    /* Crea la nuova conversazione */
    if( status.getRetweetCount() >= 0 )
     {
       db.executeUpdate("INSERT DELAYED INTO story (id_start, reply↩
            , text, time_first_post, time_last_post, n_retweet, ↩
            time_density) VALUES ('"+status.getId()+"', '0', '"+↩
            formatString(status.getText())+"', '"+formatDate(status.↩
            getCreatedAt().toString())+"', '"+formatDate(status.↩
            getCreatedAt().toString())+"', '"+status.getRetweetCount↩
            ()+"', '0')");
    } else {
       db.executeUpdate("INSERT DELAYED INTO story (id_start, reply↩
            , text, time_first_post, time_last_post, n_retweet, ↩
            time_density) VALUES ('"+status.getId()+"', '0', '"+↩
            formatString(status.getText())+"', '"+formatDate(status.↩
            getCreatedAt().toString())+"', '"+formatDate(status.↩
            getCreatedAt().toString())+"', '0', '0')");
  }

  /* Aumenta il numero di conversazioni iniziate dall'utente */
  tot_n_story += 1;
 }

 count_tweet++;
}
                    }

                    /* Aggiorna le info dell'utente */
                    System.out.println("Aggiornamento discussioni ↩
                        utente "+id_user+": story "+tot_n_story);
                    db.executeUpdate("UPDATE user SET n_story=n_story+↩
                        "+tot_n_story+" WHERE id='"+id_user+"' ");

                } catch (Exception e1) {
                    e1.printStackTrace();
                }
            }
        }
        catch (Exception ex) {
            System.err.println("[Errore] " + ex.getMessage());
            ex.printStackTrace();
            continue;
        }
     }

     /* chiude il socket */
     try {
        in.close();
     } catch (IOException e) { e.printStackTrace(); }

     System.out.println("Conversation Retrieval Client terminato");
  }


  /* Metodo per controllare se e' possibile effettuare nuove ↩
      richieste o se occorre attendere */
  private static void controlContinue(Twitter twitter) throws ↩
      Exception
```

```
{
   /* Numero di richieste minime da preservare */
   int remaining_hits = 2;

   /* Richiede il numero di req ancora disponibili per l'IP */
   RateLimitStatus status = twitter.getRateLimitStatus();

   /* Controlla se le richieste rimanenti sono in numero ↩
       sufficiente */
   if( status.getRemainingHits() < remaining_hits )
   {
      /* Richieste rimanenti insufficienti */
      System.out.println("Le richieste rimanenti sono poche: " + ↩
          status.getRemainingHits()
            + " - Rimarro' in dormiveglia per: " + (status.↩
                getSecondsUntilReset()+5) +" sec ("+(status.↩
                getSecondsUntilReset()/60)+" min) dalle "+now());

      db.disconnetti();

      /* Il thread rimane in pausa per i secondi necessari al ↩
          reset delle request (+5sec di scarto) */
      Thread.sleep( (twitter.getRateLimitStatus().↩
          getSecondsUntilReset()*1000)+(5*1000) );

      if( !db.open() )
         System.out.println("Errore connessione database: " + db.↩
             getErrore());
   }
}


/* Formatta la data nel formato YYYY-MM-DD HH:MM:SS a partire dal↩
    formato GG MM HH:MM:SS YY CEST */
private static String formatDate(String date)
{
   String date_format;
   StringTokenizer st = new StringTokenizer(date, " ");
   String[] date_token = new String[st.countTokens()];

   int i = 0;
   while(st.hasMoreTokens())
   {
     String s = st.nextToken();
     date_token[i] = s;
     i++;
   }

   /* Sostituisce il nome del mese con il suo numero */
   date_token[1] = date_token[1].replace("Jan", "01");
   date_token[1] = date_token[1].replace("Feb", "02");
   date_token[1] = date_token[1].replace("Mar", "03");
   date_token[1] = date_token[1].replace("Apr", "04");
   date_token[1] = date_token[1].replace("May", "05");
   date_token[1] = date_token[1].replace("Jun", "06");
   date_token[1] = date_token[1].replace("Jul", "07");
   date_token[1] = date_token[1].replace("Aug", "08");
   date_token[1] = date_token[1].replace("Sep", "09");
   date_token[1] = date_token[1].replace("Oct", "10");
   date_token[1] = date_token[1].replace("Nov", "11");
   date_token[1] = date_token[1].replace("Dec", "12");
```

```
        /* Compone la data nel formato YYYY-MM-DD HH:MM:SS */
        date_format = date_token[5]+"-"+date_token[1]+"-"+date_token[2]+" "+date_token[3];

        return date_format;
    }


    /* Calcola la differenza fra due date e restituisce la densita' fra le due in secondi */
    private static double differenceDate(Date start, Date finish)
    {
        /* calcola la differenza in secondi fra le due date */
        double difference = (finish.getTime() - start.getTime())/1000;

        return (1/difference);
    }


    /* Funzione per orario attuale */
    public static String now()
    {
        String DATE_FORMAT_NOW = "yyyy-MM-dd HH:mm:ss";
        Calendar cal = Calendar.getInstance();
        SimpleDateFormat sdf = new SimpleDateFormat(DATE_FORMAT_NOW);

        return sdf.format(cal.getTime());
    }


    /* Funzione per sistemare testo con caratteri particolari */
    public static String formatString(String txt)
    {
          String text = txt.replace("\\","\\\\").replace("\n"," ");
          text = text.replace("\t"," ").replace("'","\\'");
          text = text.replace("\"","\\\"").trim();

          return text;
    }
}
```

A.3 Indexer

Classe utilizzata per la realizzazione dell'indice di Lucene. Avviato lo script, sono estratte dal database le informazioni sulle nuove conversazioni da indicizzare: il loro id, il numero di risposte e il testo dell'intera conversazione.

Per rendere l'indicizzazione incrementale, viene salvato ad ogni esecuzione dello script il numero di conversazioni salvate. In questa maniera è possibile la volta successiva partire da essa e non reinserire quindi conversazioni già presenti.

```
/*****************************************
 * ConvRetrieval : Indexer
 * Classe per la realizzazione dell'indice di Lucene
 */

import java.io.BufferedReader;
import java.io.File;
import java.io.FileReader;
import java.io.FileWriter;
import java.io.IOException;
import java.io.InputStreamReader;
import java.io.PrintWriter;
import java.sql.ResultSet;
import java.sql.SQLException;
import java.text.DateFormat;
import java.text.ParseException;
import java.text.SimpleDateFormat;
import java.util.Date;
import java.util.Calendar;

import org.apache.lucene.analysis.standard.StandardAnalyzer;
import org.apache.lucene.document.Document;
import org.apache.lucene.document.Field;
import org.apache.lucene.index.IndexWriter;
import org.apache.lucene.store.FSDirectory;
import org.apache.lucene.util.Version;


public class Indexer {

    static Database db;
    static ResultSet rs;
    static String INDEX_DIR;

    static String storeIdPath;

    /**
     * Metodo principale per l'indicizzazione incrementale
     * delle conversazioni
     */
    public static void main(String[] args) throws Exception
    {
        /* File per salvataggio numero conversazioni indicizzate */
        storeIdPath = "store_id.log";
        long storeId = Long.parseLong( getStoreId(storeIdPath) );

        /* Directory dove salvare l'indice */
        INDEX_DIR = "/twitter";
```

```
/* Si connette al database */
db = new Database("194.116.73.79", "twitter", "user", "↩
    password");

if( !db.open() )
    System.out.println("Errore connessione database: "
            + db.getErrore());

/* Recupera l'ID memorizzato nel file per capire se e' la ↩
    prima indicizzazione o una incrementale */
boolean isEmpty = true;
try {
    File file = new File(storeIdPath);
    if (!file.exists())
            file.createNewFile();

    FileReader fr = new FileReader(storeIdPath);
    BufferedReader br = new BufferedReader(fr);
    if( br.readLine()!= null )
        isEmpty = false;

    br.close();
    fr.close();
} catch (IOException e) {
        e.printStackTrace();
}


/* Avvia Lucene IndexWriter */
IndexWriter writer = new IndexWriter(
    FSDirectory.open( new File(INDEX_DIR) ),
    new StandardAnalyzer(Version.LUCENE_CURRENT), isEmpty, ↩
        IndexWriter.MaxFieldLength.UNLIMITED);


/* Effettua la query per estrarre il testo e indicizzarlo */
long n_st = 0;
boolean indexFlag = false;
rs = db.executeQuery("SELECT id_start, reply, text FROM ↩
    story LIMIT "+storeId+",2000000");

System.out.println("["+now()+"] Start add stories");

while( rs!=null && rs.next() )
{
    Document doc = new Document();

    doc.add(new Field("id", rs.getLong("id_start"),
            Field.Store.YES, Field.Index.NO));
    doc.add(new Field("reply", rs.getInt("reply"),
            Field.Store.YES, Field.Index.ANALYZED));
    doc.add(new Field("text", rs.getString("text"),
            Field.Store.YES, Field.Index.ANALYZED));

    writer.addDocument(doc);

    n_st++;
    indexFlag = true;
}

db.disconnetti();
```

```
        System.out.println("["+now()+"] Added " + n_st + "stories");

        if( d>0 )
        {
            System.out.println("["+now()+"] Start optimization");
            writer.optimize();
        }

        writer.close();

        if(indexFlag)
        {
            /* Salva il numero di ID gia' indicizzati */
            long total = d + storeId;
            writeStoreId(storeIdPath, Long.toString(total));
            System.out.println("["+now()+"] Added: "+d+" stories - ↩
                Total: "+total+" stories");
        }

        System.out.println("["+now()+"] Terminated");
    }


    /* Recupera da file il numero di ID indicizzati */
    public static String getStoreId(String path)
    {
        String storeId = "";
        try {
            File file = new File(path);

            if ( !file.exists() )
                file.createNewFile();

            FileReader fr = new FileReader(path);
            BufferedReader br = new BufferedReader(fr);
            storeId = br.readLine();

            if ( storeId == null || storeId == "" )
                storeId = "0";

            br.close();
            fr.close();
        } catch (Exception e) {
            e.printStackTrace();
        }

        return storeId;
    }

    /* Scrive il numero di ID indicizzati su file */
    public static boolean writeStoreId(String path, String storeId)
    {
        boolean b = false;
        try {
            File file = new File(path);

            if ( !file.exists() )
                file.createNewFile();

            FileWriter fw = new FileWriter(path);
            PrintWriter out = new PrintWriter(fw);
            out.write(storeId);
```

```
            out.close();
            fw.close();
            b = true;
        } catch (IOException e) {
            e.printStackTrace();
        }
        return b;
    }


    /* Funzione per ricavare l'orario attuale */
    public static String now()
    {
        String DATE_FORMAT_NOW = "HH:mm:ss dd-MM-yyyy";
        Calendar cal = Calendar.getInstance();
        SimpleDateFormat sdf =
                new SimpleDateFormat(DATE_FORMAT_NOW);

        return sdf.format(cal.getTime());
    }
}
```

A.4 Searcher

Classe utilizzata per la ricerca di una parola chiave nel Conversation Retrieval. La classe viene chiamata passandogli in input la parola chiave da ricercare. È possibile inoltre specificare i pesi da assegnare ai vari parametri del ranking. In tal caso la sintassi è la seguente:

parola=[rilevanza]-[popolarità messaggi]-[attualità]-[densità]-[popolarità conversazioni]-[popolarità follower]

```
/*****************************************
 * ConvRetrieval : Searcher
 */

import java.io.File;
import java.io.IOException;
import java.sql.ResultSet;
import java.sql.SQLException;
import java.sql.Timestamp;
import java.util.ArrayList;
import java.util.Collections;
import java.util.Date;
import java.text.DecimalFormat;

import org.apache.lucene.analysis.standard.StandardAnalyzer;
import org.apache.lucene.document.Document;
import org.apache.lucene.queryParser.ParseException;
import org.apache.lucene.queryParser.QueryParser;
import org.apache.lucene.search.IndexSearcher;
import org.apache.lucene.search.Query;
import org.apache.lucene.search.ScoreDoc;
import org.apache.lucene.search.TopScoreDocCollector;
import org.apache.lucene.store.FSDirectory;
import org.apache.lucene.util.Version;


public class Searcher {

    static Database db;

    /**
     * Metodo principale per la ricerca di conversazioni
     * @param parola da ricercare
     * @throws IOException
     * @throws ParseException
     * @throws SQLException
     */
    public static void main(String[] args) throws IOException, ↩
        ParseException, SQLException, java.text.ParseException
    {
        /* Variabili per massimi della ricerca */
        double max_score = 0;
        double max_time_density = 0;
        int max_reply = 0;
        int max_n_story = 0;
        int max_n_reply = 0;
        int max_n_retweet = 0;
        int max_followers = 0;
```

```
double nn = 0;       /* max n_reply/n_story */
double fr = 0;       /* max followers/reply */
double tr = 0;       /* max time/reply      */
double rr = 0;       /* max retweet/reply   */
Date max_time = new Date(0000,0,0,0,0,0);
Timestamp max_time_2 = null;
long max_time_start = 0;

/* Cerca dall'indice creato */
String s = args[0];
String querystr;

/* Se ricerca e' nel formato parola=pesi, estra la parola da↩
     cercare */
if( s.contains("=") )
    querystr = s.split("=")[0];
else
    querystr = s;

/* Numero risultati massimi da estrarre */
int hitsPerPage = 5000;
IndexSearcher searcher = new IndexSearcher(
        FSDirectory.open( new File("/twitter") ) );
TopScoreDocCollector collector =
        TopScoreDocCollector.create(hitsPerPage, true);

/* Effettua la ricerca */
Query q = new QueryParser(Version.LUCENE_CURRENT, "text", ↩
    new StandardAnalyzer(Version.LUCENE_CURRENT)).parse(↩
    querystr);
searcher.search(q, collector);

ScoreDoc[] hits = collector.topDocs().scoreDocs;
System.out.println("Trovati " + hits.length + " risultati ↩
    per '"+querystr+"' (su "+hitsPerPage+" max)");

/* Si connette al database */
db = new Database("194.116.73.79", "twitter", "user", "↩
    password");
if( !db.open() )
{
    System.out.println("Errore connessione database: "
                + db.getErrore());
}
else
{
    ArrayList<Risultato> l = new ArrayList<Risultato>();
    String query_str = "";

    /* Ranking per testo */
    for(int i=0;i<hits.length;++i)
    {
        int k = 0;
        int docId = hits[i].doc;
        Document d = searcher.doc(docId);
        Risultato ris = new Risultato(k);

        /* Recupera id, n reply e score del risultato */
        ris.id_start = d.get("id");
        ris.reply = Integer.parseInt(d.get("reply"));
        ris.score = hits[i].score * (double) ris.reply;
```

```
        if((hits[i].score * (double)ris.reply) > max_score)
            max_score = (hits[i].score * (double)ris.reply);

        l.add(ris);
    }

    System.out.println("Score Ok");

    /* Riordina i risultati in base allo score sul testo + ↩
        reply e prende solo i primi 20 */
    Collections.sort(l);

    int risultati = l.size();
    for(int i=0; i<risultati; i++)
    {
        if( i < 20 )
            query_str +=
                "id_start='"+l.get(i).id_start+"' OR ";
    }
    /* Rimuove i risultati oltre i primi 20 */
    for(int i=20; i<risultati; i++)
    {
        l.remove(20);
    }
    query_str = query_str.replaceAll( "OR $", "" );
    System.out.println("Primi risultati: "+l.size());

    /* Cerca le info su tutte le conversazioni */
    ResultSet rs = db.executeQuery("SELECT * FROM story ↩
        WHERE "+query_str);

    while( rs!=null && rs.next() )
    {
        for(int i=0;i<l.size();i++)
        {
            if( l.get(i).id_start.equals(rs.getLong("↩
                id_start")) )
            {
                /* Setta variabili del singolo risultato */
                l.get(i).n_retweet = rs.getInt("n_retweet");
                l.get(i).time_density = rs.getDouble("↩
                    time_density");
                l.get(i).reply = rs.getInt("reply");
                l.get(i).time = rs.getTimestamp("↩
                    time_first_post");

                if( rs.getInt("n_retweet") > max_n_retweet )
                    max_n_retweet = rs.getInt("n_retweet");
                if( rs.getDouble("time_density") > ↩
                    max_time_density )
                    max_time_density = rs.getDouble("↩
                        time_density");
                if( rs.getInt("reply") > max_reply )
                    max_reply = rs.getInt("reply");
                if( rs.getTimestamp("time_first_post").↩
                    compareTo(max_time) > 0 )
                    max_time = rs.getTimestamp("↩
                        time_first_post");

                if( l.get(i).reply > 0 )
                    if( rs.getDouble("time_density")/(double↩
                        )(l.get(i).reply+1) > tr)
```

```
                    tr = rs.getDouble("time_density")/(↩
                        double)(l.get(i).reply+1);

                if( l.get(i).reply > 0 )
                    if( rs.getInt("n_retweet")/(double)(l.↩
                        get(i).reply+1) > rr)
                        rr = rs.getInt("n_retweet")/(double)↩
                            (l.get(i).reply+1);
            }
        }
    }


    System.out.println("Ricerca in corso...");
    for(int i=0;i<l.size();i++)
    {
        /* Calcola il totale dei followers, n_story e ↩
            n_reply degli utenti della conversazione */
        rs = db.executeQuery("SELECT SUM(user.followers) AS ↩
            followers, SUM(user.n_story) AS n_story, SUM(↩
            user.n_reply) AS n_reply FROM tweet JOIN user ON↩
             tweet.id_user=user.id WHERE origin='"+l.get(i).↩
            id_start+"'");
        while( rs!=null && rs.next() )
        {
            l.get(i).followers = rs.getInt("followers");
            l.get(i).n_story = rs.getInt("n_story");
            l.get(i).n_reply = rs.getInt("n_reply");

            if( rs.getInt("followers") > max_followers )
                max_followers = rs.getInt("followers");
            if( rs.getInt("n_story") > max_n_story )
                max_n_story = rs.getInt("n_story");
            if( rs.getInt("n_reply") > max_n_reply )
                max_n_reply = rs.getInt("n_reply");
            if( rs.getInt("n_story") > 0)
                if( (rs.getInt("n_reply")/(double)rs.getInt(↩
                    "n_story")) > nn )
                    nn = rs.getInt("n_reply")/(double)rs.↩
                        getInt("n_story");

            if( l.get(i).reply > 0 )
                if( rs.getInt("followers")/(double)(l.get(i)↩
                    .reply+1) > fr)
                    fr = rs.getInt("followers")/(double)(l.↩
                        get(i).reply+1);
        }
    }

    System.out.println("...terminata");

    for(int i=0;i<l.size();i++)
    {
        //Setta le variabili max
        l.get(i).set_max(max_score, max_n_retweet, ↩
            max_time_density, max_reply, max_time, ↩
            max_followers, max_n_story, max_n_reply, nn, fr,↩
             tr, rr);

        //Calcola il totale di questo risultato
        String value;
        if( s.contains("=") )
```

```
                {
                    /* pesi inseriti dall'utente */
                    value = s.split("=")[1];
                    l.get(i).total(Double.parseDouble(value.split("-↩
                        ")[0]), Double.parseDouble(value.split("-")↩
                        [1]), Double.parseDouble(value.split("-")↩
                        [2]), Double.parseDouble(value.split("-")↩
                        [3]), Double.parseDouble(value.split("-")↩
                        [4]), Double.parseDouble(value.split("-")↩
                        [5]));
                }
                else
                {
                    /* Pesi di default */
                    /* int p_score, int p_retweet, int p_timeliness,
                     * int p_density, int p_user, int p_followers */
                    l.get(i).total(70, 80, 80, 30, 80, 1);
                }
            }

            /* Chiude connessione al database */
            db.disconnetti();

            /* Ordinamento di l */
            Collections.sort(l);

            if( !l.isEmpty() )
            {
                System.out.println("Risultati della ricerca:");

                for(int i=0; i<30; i++)
                {
                  if( !l.isEmpty() )
                  {
                     Risultato r = l.remove(l.size()-1);

                     /* Stampa i valori della conversazione */
                     System.out.print(r.id_start);
                     System.out.print("- text: "+r.score);
                     System.out.print("- retweet: "+r.n_retweet);
                     System.out.print("- start: "+r.time_start);
                     System.out.print("- dens: "+r.time_density);
                     System.out.print("- reply: "+r.reply);
                     System.out.print("- n_story: "+r.n_story);
                     System.out.print("- n_reply: "+r.n_reply);
                     System.out.print("- followers: "+r.followers);

                     /* Stampa i valori del ranking */
                     System.out.print("\n");
                     System.out.print(" text: "+r.totalS[0]);
                     System.out.print("- follow: "+r.totalS[1]);
                     System.out.print("- user_conv: "+r.totalS[2]);
                     System.out.print("- densita': "+r.totalS[3]);
                     System.out.print("- retweet: "+r.totalS[4]);
                     System.out.print("- timeliness: "+r.totalS[5]);
                     System.out.print("= "+r.total+" \n");
                  }
                }
            }
        }
    }
}
```