Information Retrieval
Friends-Enhanced Personalized News Recommendations (for Twitter users)

Andrea Angiolillo Matr. 761678

January 18, 2017

Introduzione

I nostri account dei Social Network spesso riflettono in maniera molto dettagliata i nostri interessi personali:

- Le persone tendono a condividere contenuti di loro interesse;
- Le persone tendono a essere amiche di persone con i loro stessi interessi;
- Le persone tendono a "seguire" persone che condividono i loro stessi interessi.

In questo progetto è stato realizzato un *Content-based Recommender system* che, sfruttando le informazioni presenti nell'account *Twitter* di un utente, ordina una lista di *sources* tematiche in base ai suoi interessi.

Obiettivi del Progetto

- Creare 10 profili utente considerando:
  1. I tweets condivisi dall'utente;
  2. I tweets degli utenti che la persona considerata segue;

3. I tweets dei suoi Friends: un utente (x) è considerato Friend dell'utente considerato (y) sse:

$$Follower(x, y) \wedge Follower(y, x)$$

dove la relazione Follower(x,y) indica che l'utente x è follower dell'utente y;

- Crawl tutte le 70 *sources* da *News API*;
- Usare *Lucene* per gestire i profili e le *sources*;
- In output il sistema dovrà fornire, per ogni utente, una lista di *sources* ordinata in base alle sue preferenze;
- Il sistema deve essere provvisto di una interfaccia utente.

Assunzioni

- **Scelte linguistiche**
  Per quanto riguarda la scelta relativa alla lingua utilizzata nel progetto, si è deciso di utilizzare l'inglese. Questo poichè grammaticalmente più semplice dell'italiano ma soprattutto perchè è la lingua più utilizzata su *Twitter*. Per l'indicizzazione vengono quindi considerati solo i simboli alfabetici (a-z) che formano gli idiomi della lingua inglese, tutti gli altri segni di punteggiatura verranno scartati in quanto non influenti nel processo di calcolo della rilevanza.

- **Scelta degli utenti**
  La scelta dei profili *Twitter* dai quali scaricare i tweets su cui poi creare i profili utenti non è stata casuale. Si è infatti selezionato un insieme di profili che si è ritenuto utilizzassero e scrivessero in un linguaggio corretto, senza uso di abbreviazioni o termini appartenenti allo slang. Inoltre si è preferito scegliere almeno un utente per ogni categoria delle *sources* cosi da vedere per quali categorie il sistema ha migliori performance.
  Le fonti selezionate risultano quindi:
  - *@nytimes* l'account ufficiale del giornale New York Times
  - *@BBCSport* il giornale sportivo della BBC
  - *@BillGates* l'account ufficiale di Bill Gates
  - *@ClassicMusic361* account di musica classica

- *@MTVMusic* l'account ufficiale di MTV
- *@IGN* account ufficiale di IGM
- *@MiniGameReviews* account di recensioni di videogame
- *@realDonaldTrump* l'account ufficiale di Donald Trump
- *@NatGeo* l'account ufficiale di National Geographic
- *@pipTank* account sul trading
- *@NASA* l'account ufficiale della Nasa
- *@ftfinancenews* account che parla prevalentemente di finanza

Scelte Progettuali

Durante il progetto sono state intraprese diverse scelte progettuali, in questa sezione andremo a parlare delle motivazioni che hanno spinto a utilizzare determinate soluzioni e a scartarne altre.

- **Numero di tweets da scaricare**
  La scelta del giusto numero di tweets da scaricare e di come distribuirli nel profilo utente è una scelta molto importante e non banale. È ragionevole pensare che i termini più utilizzati dall'utente siano quelli che meglio rappresentano i suoi interessi. Inoltre un Friend o Following dell'utente avrà molto probabilmente degli interessi differenti e, utilizzare troppi tweets di queste due categorie, potrebbe introdurre del rumore.
  Dopo numerosi tentativi si è deciso di considerare:

  - n. di tweets dell'utente: 200;
  - n. di tweets per friends: 10;
  - n. di tweets per following: 10;
  - n. di friends: 10;
  - n. di following: 10.

  Come si può notare si ha che il profilo sarà composto da 200 tweets dell'utente e 200 tweets dei suoi friends e following.

- **Salvare i tweets in file XML**
  La libreria utilizzata per scaricare i tweets dell'utente è *twitter4j*. Questa libreria impone alcuni limiti a tutte le applicazioni di terze parti che

vogliono utilizzare la piattaforma *Twitter* per i loro scopi: nello specifico ogni operazione della libreria ha un determinato limite che, se superato, blocca per 15 minuti tutte le richieste in uscita dell'applicazione dirette a *Twitter*. Nel nostro caso questi limiti (in particolare quello sui download) non permettono alla piattaforma di scaricare tutti i tweets necessari per la creazione dei profili utenti in un unica sessione. Le soluzioni considerate sono sostanzialmente due:

1. Ridurre il numero di tweets da scaricare per ogni profilo utente:
   - Pros: molto facile da attuare;
   - Cons: il numero dei tweets diventerebbe troppo ridotto per pensare di creare un profilo che riesca a racchiudere gli interessi dell'utente.
2. Scaricare in locale i tweets:
   - Pros: è possibile scaricare un grosso numero di tweets;
   - Cons: risulta una soluzione un pò più impegnativa da implementare.

Analizzando i pros e cons si può notare come la seconda soluzione sia quella più consigliata. Nel nostro caso si è scelto di salvare i tweets in file XML al seguente percorso: otherFile/users. La struttura dei file XML creati è la seguente:

```
<user>
        <description> Descrizione utente </description>
        <id>id Twitter (senza @) </id>\\
        <location> location account </location>
        <name> nome dell'utente </name>
        <following>29018929</following>
        <following>30882524</following>
        <following>32823682</following>
        <following>30143804</following>
        <friends>111809132</friends>
        <friends>1276757451</friends>
        <friends>124202631</friends>
        <friends>98313696</friends>
        <tweets> tweets dell'utente .. </tweets>
        <tweetsFollowing> tweets dei following .. </tweetsFollowing>
        <tweetsFriends> tweets degli amici ..  </tweetsFriends>
<user>
```

Figure 1: Esempio della struttura XML utilizzata per salvare i tweets

- **Struttura progetto**
  Per sviluppare il seguente progetto sono state analizzate due soluzioni possibili:

  1. *Bag of words*: In questa soluzione si effettua una indicizzazione per tutte le sources e una indizizzazione per ogni profilo utente (nel nostro caso 12 utenti = 12 indici). Successivamente si effettua una rappresentazione *bag of words* dell'indice delle sources e dell'indice del profilo utente e si calcola la *cosine similarity*. Nello specifico, nel caso di 12 utenti, si ha: 1 indice per le source, 12 indici per i profili utente e la *cosine similarity* viene calcolata 12 volte (l'indice delle source viene sempre usato nel calcolo e l'indice del profilo utente naturalmente cambia). Questa soluzione ha il vantaggio di utilizzare tutti i termini dell'indice per il calcolo della *cosine similarity* e inoltre permette di avere una sua una maggiore gestione e personalizzazione. Lo svantaggio è che bisogna ridefinire la *cousine similarity* in quanto *Lucene* non dispone di metodi per il confronto indice-indice ma solo indice-query;
  2. *Query Lucene*: In questa soluzione viene ancora generato un indice per le source e uno per ogni utente ma in questo caso vengono selezionati i TOP N termini più utilizzati dall'utente e li si utilizzano per la creazione di una query per cercare nell'indice delle sources. Questa soluzione ha il vantaggio di non dover gestire il calcolo della *cosine similarity* infatti è possibile utilizzare quella già presente di *Lucene*. Lo svantaggio risulta essere l'utilizzo di un numero limitato di termini che compongono la query e inoltre una più difficile gestione dei boost.

  Infine si è scelta la prima soluzione per permettere una maggiore personalizzazione nel calcolo della *cousine similarity*.

- **Calcolo della Cosine Similarity**

  Il sistema, per riuscire a dare in output una lista ordinata in base alle preferenze dell'utente, calcola la *cosine similarity* tra il profilo utente e le *sources*. La *consine similarity* è una misura di similarità tra due vettori non nulli: dati due vettori di attributi numerici , α e β, il livello di similarità tra di loro è espresso con la seguente formula

  $$similarity = \cos(\theta) = \frac{\alpha \cdot \beta}{\|\alpha\| \|\beta\|}$$

  Apache Lucene mette a disposizione un calcolo della *consine similarity* leggermente più sofisticato: data una query α e un documento β lo

score viene calcolato come:

$score(\alpha, \beta) = coord\text{-}factor(\alpha, \beta) \cdot query\text{-}boost(\alpha) \cdot \frac{V(\alpha) \cdot V(\beta)}{\|V(\alpha)\|} \cdot doc\text{-}len\text{-}norm(\beta) \cdot doc\text{-}boost(\beta)$

Dalla formula appena introdotta è possibile derivarne la seguente:

$score(\alpha, \beta) = coord(\alpha, \beta) \cdot \sum_{t \in \alpha}(tf(t \in \beta) \cdot idf(t)^2 \cdot t.getBoost() \cdot norm(t, \beta))$

Dove:

1. $tf(t \in \beta)$: indica il numero di volte che il termine t compare nel documento β; in particolare il metodo ritorna:
$$tf(t \in \beta) = freq^{1/2}$$
2. $idf(t)$: questo indicatore permette ai termini più rari di dare un contributo maggiore (nel calcolo dello score) dei termini più comuni; nello specifico il metodo calcola:
$$idf(t) = 1 + \log(\frac{numDocs}{docFreq+1})$$
3. $coord(\alpha, \beta)$: è uno score basato su quanti termini della query α sono trovati nello specifico documento β.
4. $queryNorm(\alpha)$: è una normalizzazione eseguita per rendere possibile la comparazione tra le query. Essa viene calcolata come:
$$queryNorm(\alpha) = \frac{1}{sumOfSquareWeights^{1/2}}$$
5. $t.getBoost()$ = questo metodo ritorna il *Boost* del termine t;
6. $norm(t, \beta)$ = questa normalizzazione utilizza $doc.getBoost()$ e $field.getBoost()$ che ritornano rispettivamente il *Boost* del documento e del field; la normalizzazione viene calcolata come:
$norm(t, \beta) = doc.getBoost() \cdot lengthNorm(field) \cdot \prod_{f \in \beta}(f.getBoost())$

Nel progetto vengono calcolate entrambe le *cosine similarity* così da permettere un loro confronto.

Implementazione

- **Download tweets**
  La prima parte nell'implementazione del progetto è stata quella del download e del salvataggio dei tweets in un unico file formato XML, tramite la libreria *tweet4j*. Questa procedura viene svolta dai seguenti file (che si possono trovare al seguente percorso: src/twitter):

  - **User.java**: definisce la classe *User* che contiene tutti gli attributi dell'utente *Twitter* che sono utilizzati nei processi successivi.
  - **TwittersUser**: si occupa di scaricare tutti gli elementi che andranno a comporre il profilo utente salvandoli in un file XML.

- **Download sources**
  In questa fase vengono scaricate le *sources*. La libreria *News API* non impone nessun limite a una applicazione di terze parti e quindi non è stato necessario salvare le *sources* in locale.
  Per questo compito vengono utilizzati i seguenti file *.java* (situati al percorso src/source):

  - **Source.java**: definisce la classe *Source* che contiene tutti gli attributi della *source*.
  - **GetSource.java**: si occupa di scaricare tutte le *sources* tramite *News API*.

- **Indicizzazione** In questa fase vengono effettuate le indicizzazione dei profili utente e delle *sources*.

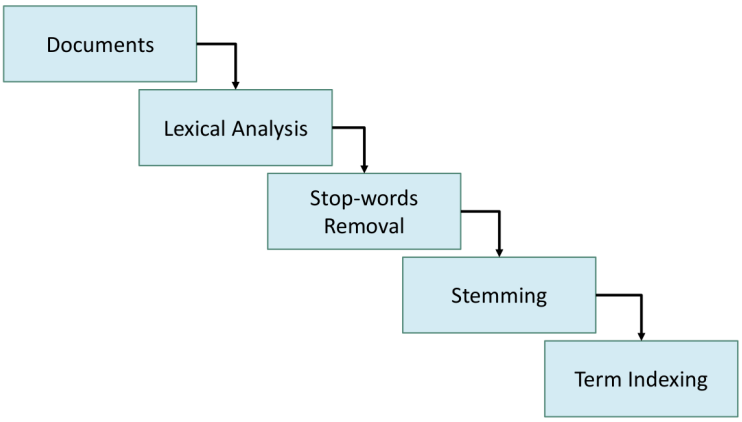

Figure 2: Fasi del processo di indicizzazione effettutato in Index.java

Il processo di indicizzazione è composto da diverse fasi, ognuna eseguita nel file Index.java (situato al seguente percorse: src/lucene). In questo file vengono svolte le seguenti operazioni :

- **Creazione oggetto *lucene.Document***:
  I documenti in *Lucene* sono rappresentati dalla classe *lucene.Document* che è composta da *lucene.Field*. La classe *lucene.Field* permette di gestire vari tipi di dati associati al documento.
  In questa fase, per ogni utente e *sources*, viene creato un oggetto *lucene.Document* contente vari elementi *lucene.Field*.
- **Creazione Analyzer**: La classe *Analyzer* è la più importante della fase di indicizzazione, infatti esegue i seguenti steps:
  1. Pre-tokenization analysis: modifica e/o elimina alcune parti del testo in input applicando dei pattern;
  2. Tokenization: suddivide il testo in input in *token*;
  3. Post-tokenization analyzis:
     (a) Text normalization;
     (b) Stop-words removal;
     (c) Stemming;
     (d) Synonym expansion.

  Nel file *MyAnalyzer.java* (situato in src/lucene) viene definito un nuovo *Analyzer* che utilizza il file CSV al percorso otherFile/stopwords-long.csv che contiene 667 *stopwords* che l'analizzatore eliminerà in quanto non utili per i nostri scopi.
- **Struttura di indicizzazione**:
  Nei metodi *writerUser* e *writerSource* viene creata una struttura di indicizzazione rispettivamente per il profilo utente e per le *sources* attraverso l'utilizzo della classe *IndexWriter*.

- **Cosine Similarity**
  Nel file *Index.java*, più precisamente nei metodi *similarity* e *lucene_similarity*, vengono calcolati i due indici applicando le formule viste precedentemente.

- **Interfaccia utente**
  Finita questa fase di sviluppo, è iniziata quella più sperimentale, in cui si è implementata una semplice interfaccia grafica che permettesse di visualizzare le *sources* ordinate attraverso l'algoritmo appena introdotto. Per eseguire il programma bisogna utilizzare il file *gui.java* al percorso /src/(default package). L'interfaccia mostrata sarà la seguente:

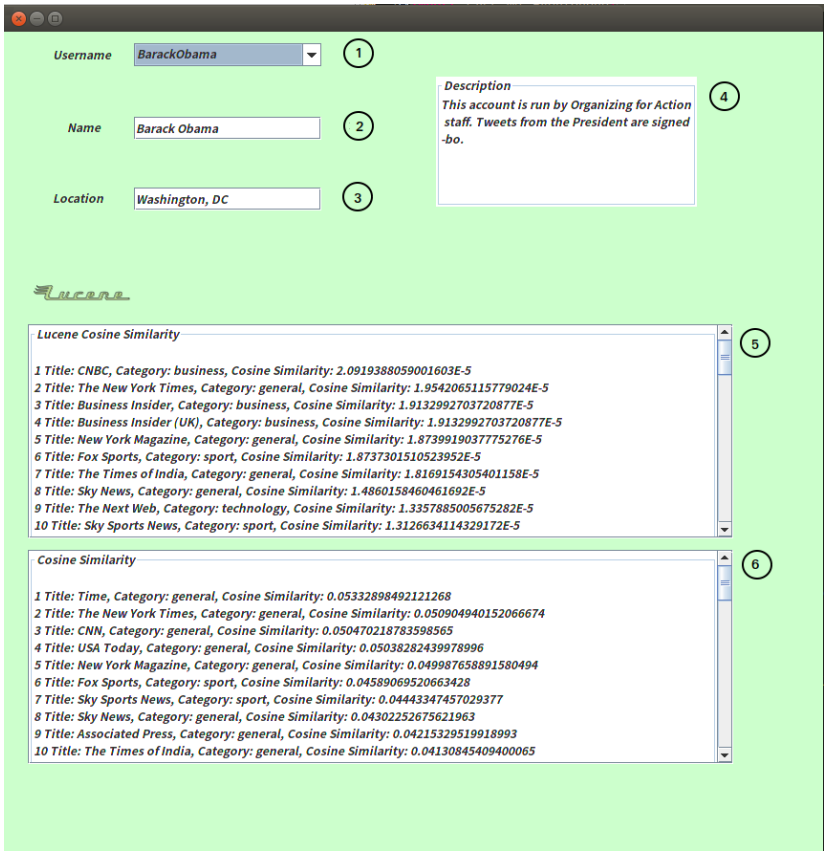

Figure 3: Interfaccia utente

1. **Username**: attraverso questo elemento è possibile scegliere l'utente su cui effettuare lo studio;
2. **Name**: viene mostrato il nome dell'utente selezionato;
3. **Location**: viene mostrata la location dell'utente selezionato;
4. **Description**: viene mostrata una piccola descrizione dell'utente;
5. **Lucene cosine similarity**: viene mostrata la lista delle *sources* ordinata secondo la *cosine similarity* di *Lucene*;
6. **Cosine similarity**: viene mostrata la lista delle *sources* ordinata secondo la normale *cosine similarity*.

Conclusioni

Analizzando le differenze tra le due *cosine similarity* si può notare come l'indice definito da *Lucene* si comporti generalmente meglio: questo perchè tiene conto di molti più fattori come la rarità del termine, il boost, la grandezza del field e la grandezza del documento. Questo risultato può sicuramente variare ed essere migliorato cambiando alcune scelte implementative:

- Un primo fattore che incide nel risultato finale è certamente la quantità di tweets usati come base iniziale per il calcolo degli indici. Al crescere dei profili da cui si vengono scaricati i dati, aumenta infatti la fonte di informazioni su cui fare analisi, e quindi gli indici calcolati risulteranno più precisi e raffinati.

- Infine un miglioramento leggero può essere portato dalla scelta degli utenti di *Twitter*. La nostra decisione di utilizzare determinati utenti potrebbe aver portato a performances diverse da quelle che si avrebbero avuto scegliendo altri profili. Nel caso di una scelta casuale degli utenti si può andare incontro a utenti che adottano un linguaggio ricco di abbrevazioni e che risulta quindi difficile da utilizzare. Contrariamente una scelta accurata può portare a un miglioramento generale del sistema.