corso di   
information Retrival

Traccia A di progetto

A cura di: Marco Valzelli, Matricola 763988 (INF) Leonardo dal Tio, Matricola 766302 (TTC)

Sommario

[Introduzione 2](#_Toc505300617)

[Scopo del progetto 2](#_Toc505300618)

[Tecnologie utilizzate 2](#_Toc505300619)

[Creazione della collezione 2](#_Toc505300620)

[Crawling 3](#_Toc505300621)

[Data selection 3](#_Toc505300622)

[Data enrichment 3](#_Toc505300623)

[Search engine: Solr 4](#_Toc505300624)

[Configurazione 4](#_Toc505300625)

[Interfacciamento con il database 5](#_Toc505300626)

[Personalizzazione 6](#_Toc505300627)

[Topic 6](#_Toc505300628)

[Insert\_tweets 6](#_Toc505300629)

[Calculate\_weights 6](#_Toc505300630)

[Get\_weights 7](#_Toc505300631)

[Boost 7](#_Toc505300632)

[Interfaccia 7](#_Toc505300633)

[Serendipity 7](#_Toc505300634)

[View 7](#_Toc505300635)

[Scelta dell’utente 7](#_Toc505300636)

[Profilo utente 8](#_Toc505300637)

[Ricerca 8](#_Toc505300638)

[Risultati 9](#_Toc505300639)

[Controller 9](#_Toc505300640)

[Valutazione dei risultati 10](#_Toc505300641)

[Comportamento su query specifiche 10](#_Toc505300642)

[Conclusioni e sviluppi futuri 11](#_Toc505300643)

Introduzione

Questo documento descrive le fasi della realizzazione e le scelte implementative riguardanti lo sviluppo di un prototipo di search engine personalizzato per il reperimento di notizie, utilizzando i concetti e le tecnologie apprese durante il corso.

Scopo del progetto

Il progetto richiedere innanzitutto di creare una collezione documenti, che in questo caso sono testi di notizie pubblicate sui social (Twitter e/o Facebook) da almeno 5 organi di stampa diversi. E’ quindi necessario utilizzare le API proprietarie di questi servizi, tenendo conto delle loro limitazioni, per poter creare la collezione.

La fase di indicizzazione e ricerca sono state indicate la libreria Lucene o la piattaforma basata su essa, Solr

I risultati di ogni ricerca vanno però personalizzati a seconda dell’utente che la effettua e dei suoi interessi. E’ quindi necessario creare degli utenti (almeno 10) con del profili multi-layer, in cui ogni livello è riferito a un interesse specifico. Ogni utente deve avere un minimo di 3 interessi diversi.

Per la creazione dei profili utenti, la traccia suggerisce di utilizzare dei testi sottoposti dall’utente, che quindi lui stesso ritiene rappresentativi.

Tecnologie utilizzate

La fase di crawling è stata effettuata con Python, salvando i dati ottenuti su un database SQL. Per la parte di indicizzazione e ricerca è stato utilizzato Solr, in grado di interfacciarsi direttamente con database. La gestione dell’interfaccia web, creata con HTML e CSS, e la personalizzazione dei risultati è stata effettuate sempre tramite python.

Creazione della collezione

La traccia indica l’utilizzo di Facebook e/o Twitter come fonte di documenti (news) con cui creare la propria collezione. Dopo una prima analisi, sono emerse le differenze dei post nei due social:

* Twitter permette solo testi brevi e non visualizza dell’anteprima del link inserito nel post: per questo il testo solitamente inserito è il titolo dell’articolo linkato, o qualcosa che ne è indicativo
* Facebook, al contrario, fornisce una ricca anteprima del link contenuto nel post che include il titolo. Per questo il testo inserito spesso è solo un commento alla notizia, poco indicativo.

Alla luce di queste differenze, è stato preferito Twitter. Per quanto riguarda le fonti di notizie, ne sono state scelte cinque tra quelle presenti sul social:

* La Repubblica
* Il Corriere
* La Stampa
* Agenzia Ansa
* Sole 24 Ore

Crawling

Per utilizzare le API del social è necessario fornire delle chiavi di identificazione, ottenibili dopo aver effettuato un’iscrizione come sviluppatore[[1]](#footnote-1). Dopo questa operazione è possibile estrarre i tweet delle fonti scelte: per questo scopo è stato creato uno script in Python utilizzando la libreria tweepy[[2]](#footnote-2). Dato che Twitter impone delle restrizioni per quanto riguarda il numero di tweet scaricabili e la loro data di pubblicazione, lo script è stato lanciato giornalmente dall’8/12/17 al 31/1/18 ottenendo circa 32.000 tweet.

Per ogni tweet è stati memorizzati nel database SQL i seguenti dati:

* Id originale
* Tweet originale (testo)
* Tweet ripulito
* Id della sorgente
* Nome della sorgente
* Data
* Numero di like
* Numero di retweet
* url contenuto nel tweet

Lo script di crawling è denominato crawler.py . Ad ogni sua esecuzione:

* Vengono reperiti i tweet per ogni fonte
  + Se nuovi, vengono ripuliti da hashtag, emoticons e urls e memorizzati sul database
  + Se già presenti, vengono aggiornati i dati sui like e retweet.
* Viene scritto un file di log con l’esito del crawling

Data selection

Già in fase di crawling è stato necessario adottare delle soluzioni per migliorare la qualità dei dati ottenuti. Per prima cosa, si è evitato di scaricare i tweet che erano in realtà retweet di altri account. Questo per due motivi:

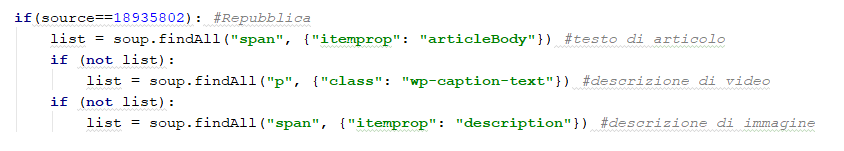
* Spesso non erano vere e proprie notizie, ma solo curiosità o approfondimenti
* Era possibile trovare più retweet dello stesso tweet, creando doppioni

Nonostante questo accorgimento, c’è comunque la possibilità di duplicati: si tratta di tweet il cui testo viene copiato e incollato per creare un nuovo tweet. Questo viene evitato tramite un confronto esatto prima dell’inserimento nel database.

Data enrichment

Per avere una collezione più completa, è stato deciso di integrare i dati ottenuti da Twitter con i testi degli articoli linkati nel tweet. Per questo sono stati creati diversi script python (total\_spyder\_x.py) che, utilizzando la libreria Scrapy[[3]](#footnote-3), creano degli spyder che seguono gli url forniti e restituiscono l’html delle pagine web.

Successivamente, con la libreria di parsing HTML BeautifulSoup[[4]](#footnote-4), veniva estratto il testo significativo. E’ stato necessario configurare il parsing per ogni fonte e per ogni tipo di contenuto: articoli, video, gallery. Qui un esempio nel caso di Repubblica:



Anche dopo questo tipo di selezione non è stato sempre possibile ottenere un testo, per vari motivi. Il caso più tipico è quello del link alla propria Home Page. Per questo, i tweet corredati da articoli solo divenuti 25.800. Anche in questo caso viene salvato un file di log, per monitorare l’esecuzione periodica

Search engine: Solr

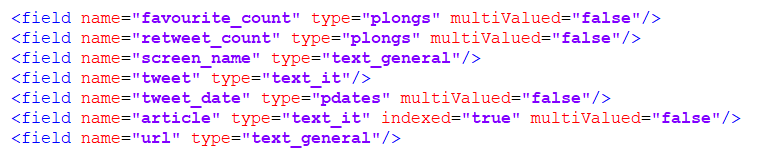
Sol[[5]](#footnote-5)r è una piattaforma che utilizza la libreria Apache Lucene per fornire funzionalità di più altro livello. Permette di:

* Indicizzare grandi quantità di documenti, anche in modalità full-text
* Accetta documenti in XML, Json, CSV ma con le numerose le opzioni diventano molte di più
* E’ utilizzabile con numerosi linguaggi: Java, Python, ecc
* Permette la modalità cloud
* E’ configurabile facilmente tramite XML
* Offre un’interfaccia web per la gestione
* Riceve comandi tramite un’API Rest

Per questi motivi è stata preferita rispetto al solo Lucene. In più, per alcune operazioni su Solr, ci si è serviti della libreria pysolr[[6]](#footnote-6).

Configurazione

Per prima cosa è stato creato il core, l’unità computazione di Solr che ospita la collezione, la sua configurazione e l’indice. All’interno del core si trova il file schema.xml, che va modificato in base ai campi del documento che si vogliono indicizzare. Il file contiene già tutti i tipi di field preconfigurati da Solr: ad ogni tipo di campo posso essere abbinati Tokenizer, Filters e Stemming diversi. Tra questi tipi vengono scelti quelli da abbinare ai field da indicizzare. Nel nostro caso i field corrispondono ai campi del database contenenti l’informazione:



In particolare per i campi tweet e article è stato indicato come tipo di field quello di testo italiano.

Interfacciamento con il database

Per impostare la connessione con il database è necessaria una configurazione[[7]](#footnote-7). Per prima cosa bisogna posizionare i file .jar dei driver JDBC nella cartella solr>contrib>dataimporhandler.

Inoltre all’interno del core sono stati modificati i file solrconfig.xml e data-config.xml. In particolare in quest’ultimo sono stati settati i parametri veri e propri per la connessione al database:



Una volta impostato tutto, per indicizzare è sufficiente eseguire lanciare il comando REST

<http://localhost:8983/solr/demo/dataimport?command=full-import>

Personalizzazione

Al motore di ricerca è richiesta una personalizzazione dei risultati sulla base dell’utente che effettua la query e dei suoi interessi. Il modello che è stato scelto è una bag-of-words, creata per ogni topic di ogni utente a partire da documenti da lui forniti. Queste parole verranno poi impiegate in una query expansion, personalizzando la query dell’utente. In aggiunta a questo, sono state predisposte altri parametri con cui l’utente può variare la query: la freschezza delle notizie e la loro popolarità

Topic

Per prima cosa, sono state create attraverso Solr diverse collezioni: una per ogni interesse di ogni utente (con la sintassi utente\_topic)

Queste collezioni vengono poi gestite attraverso la classe User (nel file user.py) che permette diverse operazioni:

Insert\_tweets

Dato in input un utente, un topic, e una lista di ID, provvede ad estrarne il testo dal database e inserirlo nella collezione opportuna su solr, creando un oggetto JSON e inviandolo tramite una richiesta REST. Questa operazione è stata svolta manualmente, data la natura didattica del progetto.

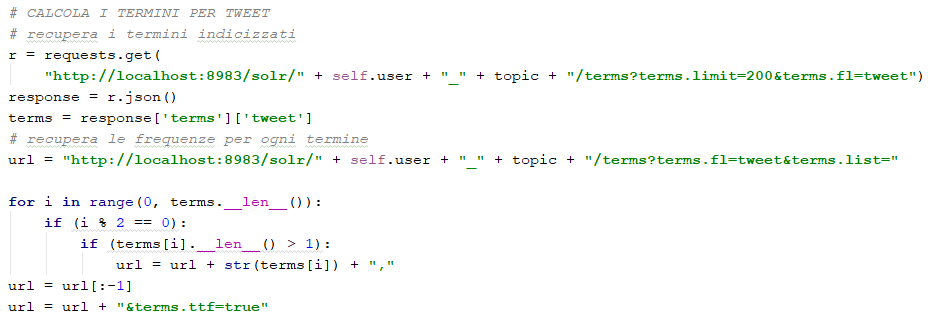
Calculate\_weights

Dati in input un utente, un suo topic, questo metodo estrae i termini indice dai field di Solr.

* Per i tweet richiede a Solr le frequenze con cui gli indici compaiono in tutti i documenti. Verrà poi calcolato un peso per uguale alla frequenza del termine divisa per la frequenza massima di un termine.
* Per i testi degli articoli richiede a Solr le frequenze con cui gli indici compaiono in tutti i documenti e anche la frequenza di documenti in cui compaiono. Il peso di ogni indice sarà calcolato con la formula TF/IDF modificata. Infatti, dato il ristretto numero di documenti in una collezione e la loro natura di tematica omogenea, il fattore di normalizzazione nel logaritmo è uguale a uno.

In entrambi i casi, al termine della computazione, i termini vengono memorizzati sul database.

Nell’esempio seguente si può notare come nel primo url il nome della collezione è generato dinamicamente a partire da topic e username. Successivamente il secondo url viene composto aggiungendo la lista di termini ottenuti con la prima richiesta, al fine di ottenere le loro frequenze nei documenti



Get\_weights

Questa funzione viene utilizzata a runtime, quando un utente effettua una ricerca personalizzata. Indicando l’utente che ha effettuato la query e il topic scelto, verranno caricati dal database e restituiti gli opportuni termini con i loro pesi da aggiungere alla query. I termini della query originaria avranno un peso 3, quelli aggiunti avranno un peso uguale a come calcolato sopra. E’ stato scelto di utilizzare come termini di espansione solo gli indici estratti da tweet, in quanto più efficaci. Infatti il tweet spesso è il titolo dell’articolo linkato.

Boost

Come ulteriore parametro di personalizzazione della query, sono stati aggiunti due possibili boost a scelta dell’utente:

* La data, che sfrutta quella estratta da twitter
* La popolarità. Questa è espressa come numero di like

Interfaccia

Per lo sviluppo dell’interfaccia è stato utilizzato il framework python Flask[[8]](#footnote-8), insieme a una grafica costruita con HTML e CSS.

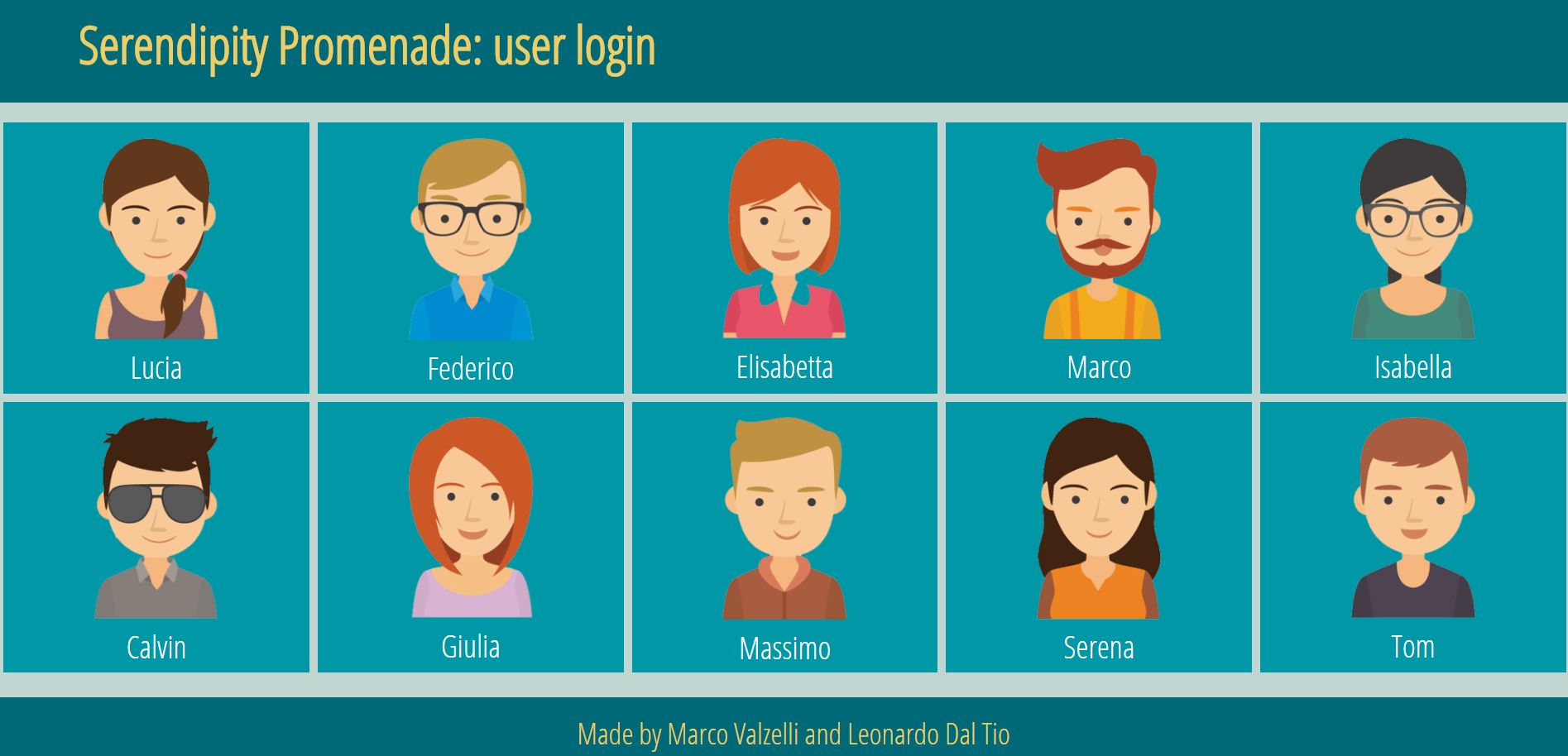
Serendipity

L’idea che caratterizza il progetto è quella di lasciare all’utilizzatore la possibilità di controllare il grado di serendipity della propria ricerca. In tal senso, sotto la barra di ricerca, verrà lasciata all’utente l’opzione di considerare o meno la contingenza temporale o la popolarità dei tweet per modificare il ranking dei risultati.

View

Scelta dell’utente

La schermata iniziale contiene le icone dei profili utenti tra cui è possibile scegliere: cliccando su uno di essi si accede alla schermata successiva



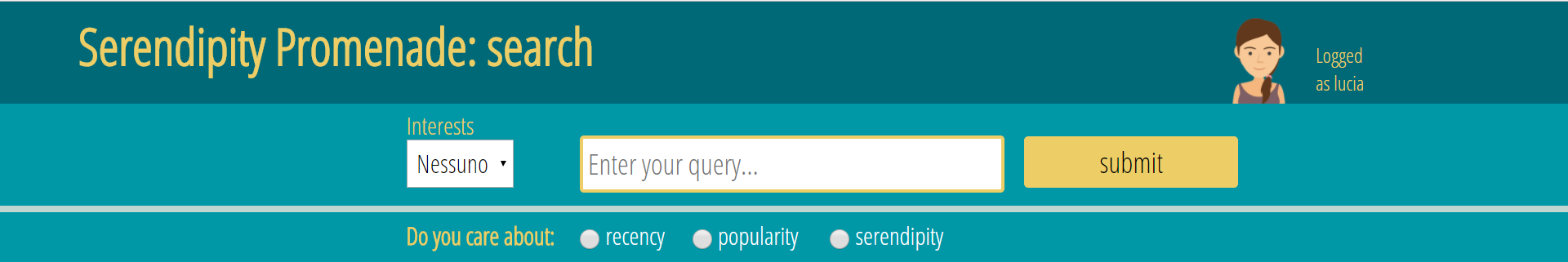
Profilo utente

Dopo aver scelto un utente, viene visualizzato il suo profilo contenente i topic in cui è interessato. In particolare, per ogni topic si visualizza anche la caratterizzazione specifica. Da qui è possibile proseguire alla ricerca o tornare indietro.



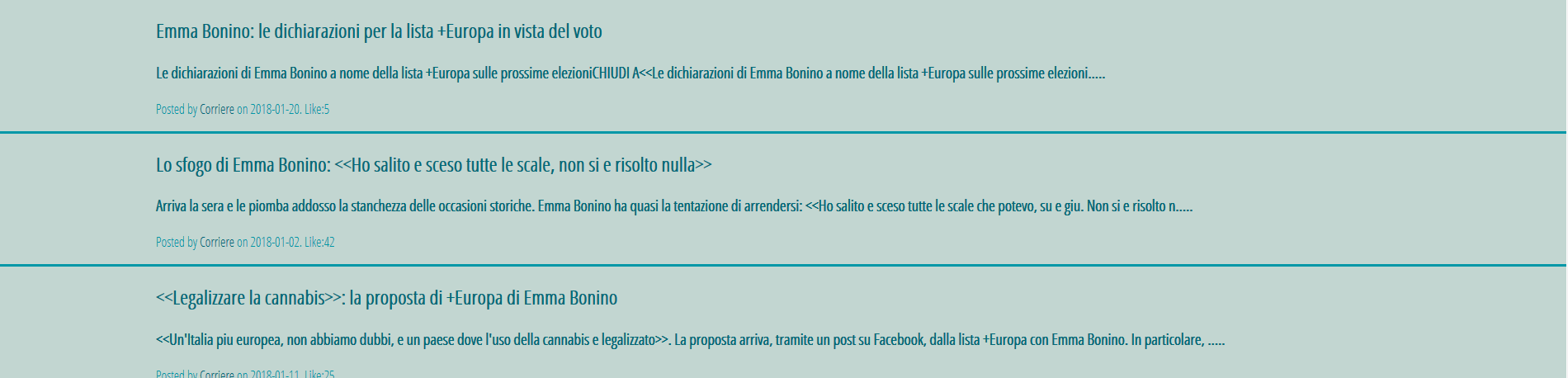
Ricerca

La schermata di ricerca vera e propria comprende un form dove immettere le parole chiave, una tendina dove scegliere l’interesse per la personalizzazione, e dei radio button dove scegliere il tipo di boost



Risultati

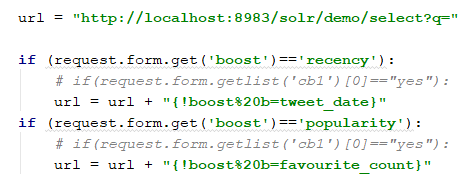
I risultati vengono presentati con titolo, snippet di testo, data di pubblicazione, numero di like e giornale d’origine. Cliccando sul titolo si verrà reindirizzati all’articolo originario.



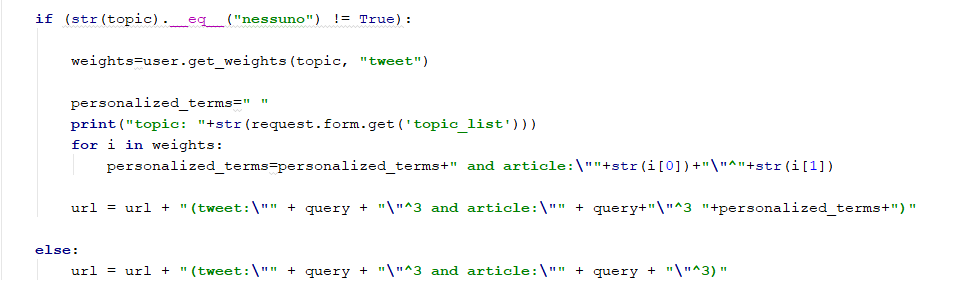
Controller

Il controller è il file client.py. Si occupa di ricevere le richieste dal browser, recuperare i dati necessari e restituire la pagina corretta. In particolare quando viene richiesta la pagina di ricerca, riempie il menu a tendina con i topic dell’utente che l’ha richiesta. Dopo che l’utente ha riempito il form e sottoposto la query, il metodo in client.py si occupa di comporre la query ed inviarla a Solr.

In particolare, prima verifica il boost selezionato:



Poi, se è stato selezionato un topic, espande la query con i termini e i pesi relativi.



E infine visualizza il risultato.

Valutazione dei risultati

Abbiamo effettuato una valutazione della precisione a 4 livelli di richiamo. I cut-off che utilizzeremo sono a 5, 10, 20 e 30 documenti della lista ordinata per rank. Abbiamo usato 3 query per utente e 3 diversi profili utente. Per forza di cose abbiamo deciso noi arbitrariamente quali sono i documenti che i soggetti fittizi, creati tramite i profili per la personalizzazione, avrebbero ritenuto rilevanti o no.

Le query che abbiamo usato sono (indico qui sia la query che il profilo utente che l’interesse selezionato per personalizzare):

1. Utente: Lucia, interesse: musica, query: concerti
2. Utente: Lucia, interesse: politica, query: partito
3. Utente: Lucia, interesse: sport, query: medaglia
4. Utente: Elisabetta, interesse: costume, query: celebrazione
5. Utente: Elisabetta, interesse: cronaca, query: tragedia
6. Utente: Elisabetta, interesse: politica, query: movimento
7. Utente: Isabella, interesse: politica, query: scontri
8. Utente: Isabella, interesse: letteratura, query: autore
9. Utente: Isabella, interesse: musica, query: concerti

Per tutte le 9 query è stata scelta l’opzione “serendipity” per cui il ranking non viene alterato né dalla popolarità né dalla contiguità temporale.



Laprecisione come ci aspettiamo scende all’aumentare dei documenti considerati per calcolarla e generalmente non è altissima, probabilmente per il criterio con cui sono state scelte le query d’esempio come procediamo a spiegare

Comportamento su query specifiche

1. La query numero 8 ha una precisione molto bassa, ma è un risultato che ci aspettavamo, la query consiste nella parola “autore” sull’interesse letteratura e il profilo dell’utente Isabella riguardo a quell’interesse è creato sulla base di tweet e articoli che parlano di fumetti. In generale tweet e articoli su fumetti sono pochi e quei pochi sono stati ritrovati, il primo risultato ad esempio parla di un documentario su un fumettista
2. In generale le query sulla musica non hanno alta precisione, questo forse perché ci sono pochi tweet sull’argomento e ancora di meno per gli specifici interessi dei due utenti generati (che nello specifico preferiscono musica classica e musica rock). Considerando un criterio meno stringente per valutare se un risultato è rilevante la precisione migliora (ad esempio considerano risultati su cantanti pop come rilevanti per l’interesse “musica rock”).
3. Alcune personalizzazioni invece hanno funzionato abbastanza bene. Ad esempio nella query 6, tramite la parola “movimento” sull’interesse politica, otteniamo molti risultati riguardo il movimento 5 stelle che è l’interesse specifico di quel profilo riguardo all’ambito politico. Oppure la query 7 dove con la parola scontri dà risultati su Medio Oriente, Siria, Iran che erano l’interesse specifico di quel profilo.
4. Utente: Marco, interesse: medicina, query: sport. Questa query dà come primo risultato un tweet su sport e diabilità, questo perché l’interesse di Marco nell’ambito medicina è creato su articoli che riguardano le disabilità. Lo stesso tweet in altri profili (o sempre nel profilo di Marco, ma scegliendo un altro interesse) non compare come primo risultato.

Conclusioni e sviluppi futuri

Il progetto ha permesso di implementare i concetti visti a lezione ed affrontare concretamente le problematiche relative al retrival e alla sua personalizzazione.

Lo sviluppo del progetto è stato stimolante e durante la sua realizzazione sono emersi possibili aspetti da approfondire e funzionalità aggiuntive:

* Per completare l’interfaccia, l’ideale sarebbe che l’utente possa indicare un risultato della ricerca come significativo per un topic, e aggiungerlo così alla sua collezione tematica.
* Sarebbe utile implementare il boost sia per data che per popolarità contemporaneamente
* Come ulteriore forma di personalizzazione, si potrebbe tenere conto dei risultati visitati dall’utente
* La selezione degli indici, specialmente nella personalizzazione, è cruciale: in un progetto di queste dimensioni emerge subito come questo aspetto sia di difficile calibrazione.
  + Una possibile azione per migliorare questo aspetto sarebbe la creazione di collezioni tematiche, per avere una pesatura migliore dei termini.
    - Es. Quanto gli articoli sullo sport di un utente si differenziano da tutti gli altri articoli sportivi?
* Si potrebbero utilizzare i tweet scritti dagli stessi utenti anziche un insieme di parole tratte da tweet e articoli che hanno condiviso

1. developer.twitter.com [↑](#footnote-ref-1)
2. tweepy.org [↑](#footnote-ref-2)
3. scrapy.org [↑](#footnote-ref-3)
4. crummy.com/software/BeautifulSoup/ [↑](#footnote-ref-4)
5. lucene.apache.org/solr/features.html [↑](#footnote-ref-5)
6. github.com/django-haystack/pysolr [↑](#footnote-ref-6)
7. gist.github.com/maxivak/3e3ee1fca32f3949f052#file-00-tutorial-md [↑](#footnote-ref-7)
8. flask.pocoo.org/ [↑](#footnote-ref-8)