

UNIVERSITÀ DEGLI STUDI DI FIRENZE
FACOLTÀ DI INGEGNERIA
DIPARTIMENTO DI INGEGNERIA DELL'INFORMAZIONE
CORSO DI LAUREA TRIENNALE IN
INGEGNERIA INFORMATICA



ALGORITMI DI PROFILAZIONE SU SOCIAL
NETWORK PER INTERFACCE DI RICERCA
WEB MULTIMEDIALI

Profiling algorithms on Social Network for multimedia
web search interface

Studente:
TOMMASO MARIGO

Docente:
Prof. ALBERTO DEL BIMBO

Correlatori:
ANDREA FERRACANI

ANNO ACCADEMICO 2017-2018

INTRODUZIONE

Già da diversi anni ormai alcuni rapporti affidabili come quello pubblicato da Parse.ly [1] hanno mostrato come i Social Network battano i motori di ricerca, in quanto attualmente costituiscono le principali fonti riferimento verso le reti di editori. Ciò significa che gli utenti web tendono a cercare informazioni (testi, notizie ma anche immagini e video) sempre più sui social network rispetto ai tradizionali motori di ricerca come Google, Bing o Yahoo. Questo sorpasso pone nuove sfide e opportunità per migliorare i sistemi di recupero di informazioni online, considerando anche che non esiste ancora un sistema condiviso per la ricerca full-text e il recupero multimediale su questa enorme quantità di dati.

L'obiettivo della tesi consiste nella realizzazione di un sistema che permetta, sfruttando i dati dei Social Network degli utenti, di aiutare i motori di ricerca nel loro lavoro di espansione di query, ordinamento dei risultati, suggerimenti di ricerca e tutte quelle cose che hanno come scopo quello di andare in contro il più possibile agli interessi dell'utente. È stata creato un motore di ricerca su contenuti Facebook di utenti che, attraverso una profilazione, restituisce per prima i contenuti che possono essere più utili per l'utente stesso.

L'idea alla base che si è voluto implementare è la seguente: se due utenti sono simili allora i contenuti associati ad uno possono essere di interesse per l'altro. Dunque se un utente A e un utente B sono simili, o hanno un qualche tipo di similarità, allora possiamo pensare che ciò che interessa ad A potrebbe interessare anche a B.

Capitolo 1

TECNOLOGIE

1.1 Django

Per la creazione del motore di ricerca è stato scelto Django. Django è un Web Framework di alto livello scritto in Python, che consente lo sviluppo di siti web sicuri e scalabili rapidamente. Tra i suoi utilizzatori celebri, Django vanta: Instagram, Disqus, Pinterest, National Geographic, BitBucket, PBS, Washington Town, Mozilla e tanti altri.

Considerando la sua grandissima popolarità, e il fatto che si tratta di un Web Framework Open Source, possiamo contare su puntuali e continui aggiornamenti del software, una comunità di sviluppatori fantastica e molto supporto. Il fatto che sia scritto utilizzando Python inoltre ne permette una facile integrazione col parco librerie del linguaggio. Django porta con sé tantissimi vantaggi, che lo rendono uno dei Web Framework più apprezzati al mondo.

- **Completezza:** seguendo la filosofia “batterie incluse”, il framework mette a disposizione degli sviluppatori tantissime componenti pronte all’uso, il che permette di concentrarsi sullo sviluppo software senza dover reinventare la ruota.
- **Sicurezza:** Django è stato concepito con un forte accento sulla sicurezza e fornisce tanti accorgimenti utili ad evitare gran parte degli errori più comuni. Tra le tante, fornisce protezione automatica per vulnerabilità quali SQL injection, cross-site scripting e cross-site request forgery.

- **Versabilità:** Django può essere ed è stato utilizzato per la creazione di qualsiasi tipologia di sito, CMS e WIKI, Social Network, siti di eCommerce, News e altro. Può fornire contenuto in tantissimi formati, come HTML, JSON, RSS, ed essere esteso da componenti esterne come Django Rest Framework. Inoltre supporta nativamente database quali MySQL, PostgreSQL e SQLite.
- **Scalabilità:** per via dell'architettura utilizzata, è possibile far scalare un sito scritto in Django fino ad accogliere centinaia di milioni di utenti.
- **Mantenibilità:** Django è stato scritto rispettando il principio di scrittura software D.R.Y. ovvero “Don’t Repeat Yourself”. Questo, abbinato alla sua architettura modulare, ci consente di mantenere il nostro codice robusto e aggiornato.
- **Portabilità:** Django è scritto utilizzando Python e ne eredita quindi la natura portabile, permettendoci di utilizzarlo senza problemi in tutti i sistemi operativi più comunemente diffusi.

Il pattern architetturale proprio di Django è il modello MTV [2].

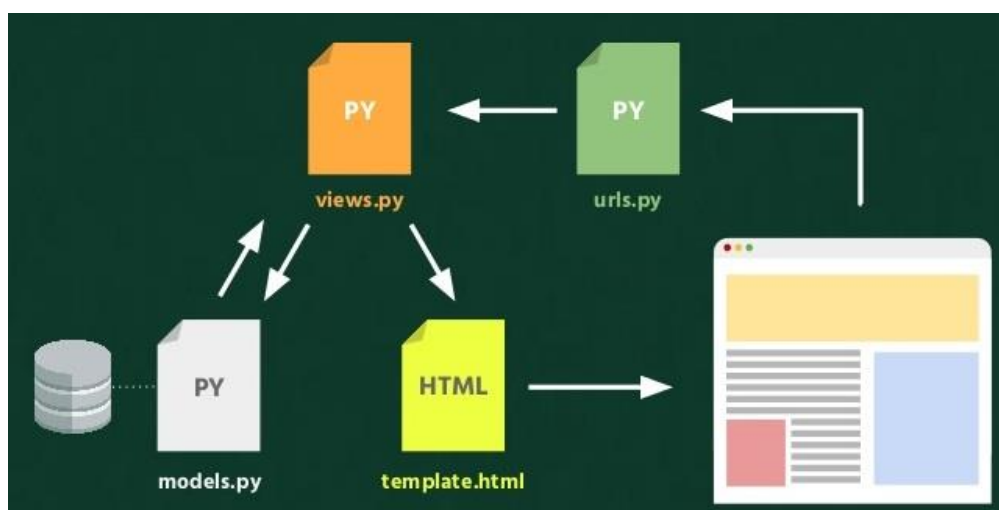


Figura 1.1 Modello MTV

In questo modello, rappresentato in figura 1.1, una richiesta ad un URL viene inviata ad una View (componente che in questo caso riceve i comandi dell'utente e li attua modificando lo stato degli altri componenti). Questa View richiama il Model (componente centrale che fornisce i metodi per accedere ai dati utili all'applicazione), esegue le manipolazioni e prepara i dati per l'output. I dati vengono passati a poi un Template che viene restituito come risposta.

1.2 OAuth2

Il termine OAuth [3] si riferisce a un protocollo generico di autenticazione aperta. Lo scopo che tale protocollo si prefigge è quello di fornire un framework per la verifica delle identità delle entità coinvolte in transazioni sicure.

Esistono, al momento, due versioni di questo protocollo: OAuth 1.0 e OAuth 2.0. Entrambe le versioni supportano l'autenticazione two-legged (letteralmente, “a due gambe”), in cui un server viene garantito circa l'identità dell'utente, e l'autenticazione three-legged, in cui un server è garantito da un'applicazione (o, più generalmente, da un content provider) circa l'identità dell'utente. Quest'ultimo tipo di autenticazione richiede l'utilizzo degli access token ed è quella comunemente implementata dai Social Network (e.g., Facebook e Twitter) attualmente.

1.2.1 Autenticazione three-legged

Il punto focale della specifica OAuth è che il content provider (e.g., un'applicazione Facebook) deve garantire al server che il client possieda un'identità. L'autenticazione three-legged offre tale funzionalità senza che il client o il server necessitino mai di conoscere i dettagli di tale identità (i. e., username e password).

Il processo di autenticazione three-legged funziona tramite i seguenti passi:

1. Il client effettua una richiesta di autenticazione al server, il quale controlla che il client sia un utente legittimo del servizio che esso offre.
2. Il server indirizza il client verso il content provider affinché possa richiedere l'accesso alle sue risorse.
3. Il content provider valida l'identità del client e (spesso) richiede i permessi necessari ad accedere ai suoi dati.
4. Il content provider indirizza il client verso il server, notificando il successo o il fallimento della sua operazione. Questa operazione, che è anch'essa una richiesta, include un codice di autorizzazione nel caso di successo dell'operazione precedente.
5. Il server effettua una richiesta out-of-band al content provider, scambiando il codice di autorizzazione ricevuto con un access token.

Si noti come il server verifichi sia l'identità dell'utente (i. e., client), sia quella del consumatore (i. e., content provider).

Ogni scambio (i. e. client verso server e server verso content provider) include la validazione di una chiave segreta condivisa.

La differenza tra OAuth 1.0a e OAuth 2.0 si palesa nelle modalità in cui avviene tale validazione. Infatti, mentre nel caso della versione 2.0, dovendo la comunicazione avvenire necessariamente su SSL, la chiave segreta viene validata direttamente dal server, nel caso della versione 1.0, tale chiave viene firmata sia dal client sia dal server (con varie complicazioni quali l'ordine degli argomenti). Quindi il protocollo OAuth 2.0 semplifica i passi 1, 2 e 5 precedentemente illustrati poiché, essendo implementato su SSL, elimina il bisogno che client e server accedano anch'essi ai servizi forniti dal protocollo OAuth.

A questo punto il server possiede un access token equivalente alla coppia username e password dell'utente. Esso potrà quindi effettuare richieste al content provider da parte dell'utente passando tale access token come parte della richiesta (e.g., come parametri di query, nell'intestazione HTTP o nei dati associati a una richiesta POST).

Se il content provider può essere contattato solo tramite SSL, allora l'implementazione OAuth è completa. In caso contrario, invece, è necessario prevedere dei meccanismi di protezione dell'access token.

Spesso quest'ultimo problema è risolto con l'utilizzo di un nuovo access token (chiamato refresh token), equivalente di una password permanente, utilizzato solo per ottenere in cambio degli access token con scadenza temporale. Tale approccio, comunque, è utilizzato solo per fornire maggiore sicurezza nel caso in cui l'accesso effettuato non sia criptato tramite connessione SSL.

Infine, la fig. 1.2 mostra il flusso di OAuth 2.0:

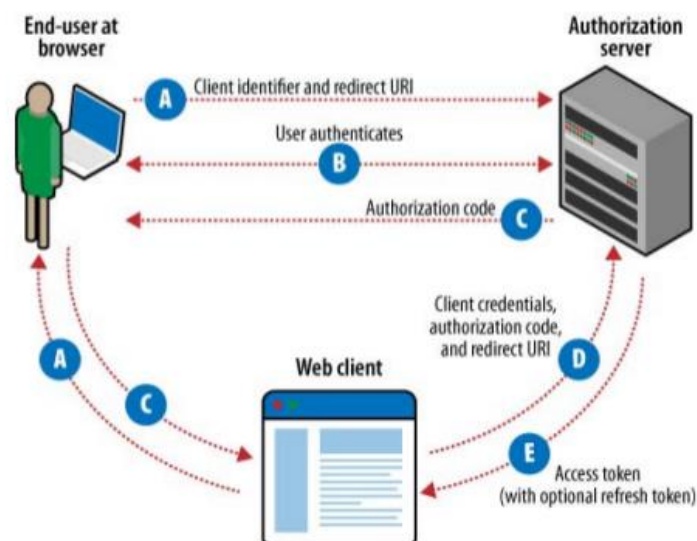


Figura 1.2 Il flusso del framework OAuth 2.0

In conclusione, al di là dei tecnicismi implementativi, il protocollo OAuth 2.0 semplifica tutte le comunicazioni fra client, server e content provider. Infatti, dal punto di vista implementativo, il vantaggio principale è la sua ridotta complessità: tale protocollo non richiede procedure di registrazione, riduce la quantità di lavoro

necessario ad agire come client di un servizio e riduce la complessità della comunicazione tra server e content provider (permettendo perciò una maggiore scalabilità).

1.3 Facebook Graph API

Al centro di Facebook c'è il social graph: un grafo i cui nodi rappresentano le entità (i. e., persone, pagine, applicazioni) e i cui archi rappresentano le connessioni di tali entità. Qualsiasi entità, o oggetto, che opera su Facebook è un nodo di tale grafo sociale. Ogni azione che un'entità compie su tale piattaforma identifica un arco (etichettato) uscente dal nodo ad esso correlata. L'etichetta di tale arco è comunemente chiamata verbo.

Esiste una sola modalità di interazione con tale grafo sociale, cioè tramite chiamate HTTP alle API di Facebook. L'interazione è divisa in due componenti:

- Graph API API REST per la lettura e scrittura del grafo sociale.
- Open Graph protocol (OGP) Meccanismo che permette di inserire qualsiasi oggetto (e.g., pagina Web) nel grafo sociale di Facebook semplicemente inserendo in esso dei meta-dati RDFa. Tale piattaforma fornisce anche un insieme di API HTTP per l'estensione del grafo sociale anche dal punto di vista delle connessioni che esso supporta.

Per rimarcare l'estensibilità del grafo sociale tramite le Object API (permettono la creazione di nuovi verbi e oggetti, previa approvazione), Facebook si riferisce ad esso anche utilizzando il termine Open Graph.

Le Graph API rappresentano un metodo consistente di ottenere una vista uniforme sul grafo sociale di Facebook attraverso delle semplici chiamate HTTP. Esse, infatti, permettono di ottenere un sottoinsieme di nodi di tale grafo (e.g., i profili, le foto, gli eventi) e le connessioni che intercorrono tra essi (e.g., le relazioni di amicizia, i contenuti condivisi, e i tag nelle foto). Costituiscono perciò il metodo

di lettura e scrittura di dati da Facebook. Su di esse è basato tutto il funzionamento di Facebook stesso.

1.4 Apache Solr

Apache Solr è un server di ricerca open-source basato sulle librerie Apache Lucene e distribuito sotto licenza “Apache License, Version 2.0”.

1.4.1 *Principi basilari*

Apache Solr gestisce i dati al proprio interno impiegando un repository basato su una struttura ad indice invertito, ovvero composta da due elementi: il vocabolario e le occorrenze. Il vocabolario è costituito dall'insieme di tutte le parole del testo a cui vengono associate la lista delle posizioni nel testo dove esse compaiono. Tali liste costituiscono le occorrenze. In Solr e Lucene, un indice è costituito da uno o più Document ed ognuno consiste di uno o più Field. Un Field è costituito da un nome, il contenuto ed una serie di parametri che specificano al sistema come trattarne il contenuto. Ad esempio i campi possono contenere stringhe, numeri, valori booleani, date, ecc..., ed ogni campo può essere descritto usando determinate opzioni che specificano al sistema come trattarne il contenuto durante le fasi di indicizzazione e ricerca (processo di analisi).

1.4.2 *Processo di analisi*

Se opportunamente specificato, Solr avvia il processo di analisi sui campi di un documento durante le fasi di indicizzazione e/o ricerca. Esempio di processi di analisi sono:

- Eliminazione delle stopwords: articoli, congiunzioni ecc.
- De-hyphenation: divisione in più token di parole contenenti un trattino

- **Stemming**: riduzione delle parole alla loro radice grammaticale
- **Thesauri**: gestione dei sinonimi

1.4.3 Operazioni di indicizzazione

L'indicizzazione è il processo durante il quale Solr riceve in input Document inviati attraverso messaggi HTTP POST. É possibile inviare quattro differenti tipi di richieste di indicizzazione:

- **add/update** consente di aggiungere o modificare documenti in Solr. Le modifiche al sistema non saranno disponibili per la ricerca finchè non viene effettuato il commit.
- **commit** specifica che le ultime modifiche apportate al sistema devono essere rese disponibili per la ricerca.
- **optimize** ristruttura i file di Lucene al fine di migliorare le performance durante la ricerca.
- **delete** elimina un document specificato da id o un insieme di documenti risultati da una query.

1.4.4 Comandi di ricerca

Solr accetta messaggi HTTP GET ed HTTP POST per le query di ricerca che vengono processate da apposite istanze dell'interfaccia SolrRequestHandler. La sintassi [4] per le query in Solr è la stessa supportata da LuceneQueryParser, con l'aggiunta di funzionalità per l'ordinamento.

Un esempio di semplice query Solr è la seguente:

```
http://localhost:8983/solr/select?q=myField:Java AND  
otherField:developerWorks; date asc
```

con la quale si ricercano i termini “Java” e “developerWorks” rispettivamente nei campi “myField” e “otherField” e si ordina il risultato sulla base del campo “date”. Una volta sottomessa la query, in caso di match con i documenti indicizzati, Solr restituisce una risposta XML contenente i risultati. Solr consente inoltre di effettuare ricerche sfaccettate su specifici Fields opportunamente configurati.

1.4.5 Solr schema

Solr gestisce le funzionalità descritte sulla base dell'apposito file di configurazione schema.xml. Lo schema è fondamentale per la definizione della struttura che devono avere i documenti trattati all'interno del server ed è suddiviso in due sezioni principali:

- Types
- Field

Nella sezione fields vengono definiti i campi che compongono i documenti, specificando per ognuno di essi un nome, la tipologia (types) e se devono essere indicizzati e memorizzati all'interno del sistema. Nella sezione types vengono definite le tipologie che possono assumere i fields. Tali tipologie possono essere semplici stringhe, valori interi, ecc..., oppure tipi di dati più complessi in cui viene espressamente indicato il processo di analisi che deve essere applicato al contenuto dei campi appartenenti a quella tipologia per l'estrazione dei singoli termini. Il processo di analisi può essere definito indipendentemente per le due diverse fasi di indicizzazione e ricerca e prevede la definizione di un tokenizer per

l'estrazione dal contenuto di token rispondenti a determinate regole, più tutta una serie di filtri quali eliminazione di stopword, stemming, etc. Il sistema quindi indicizzerà per ogni campo tutti i termini risultanti dal processo di analisi definito nello specifico tipo definito per quel campo ed analogamente effettuerà le interrogazioni con i termini risultanti dall'altro processo di analisi definito per la fase di ricerca.

Capitolo 2

L'APPLICAZIONE

In questo capitolo andremo a descrivere meglio l'applicazione vera e propria, il suo scopo e la sua architettura. Come è stato spiegato nell'introduzione, è stato creato un motore di ricerca dove i contenuti sono i dati di Facebook degli utenti, che analizza le query e, attraverso una ricerca per similarità restituisce per prima quei contenuti che possono essere più di interesse per l'utente.

2.1 Architettura dell'Applicazione

In questo paragrafo verrà definita l'architettura generale dell'applicazione.

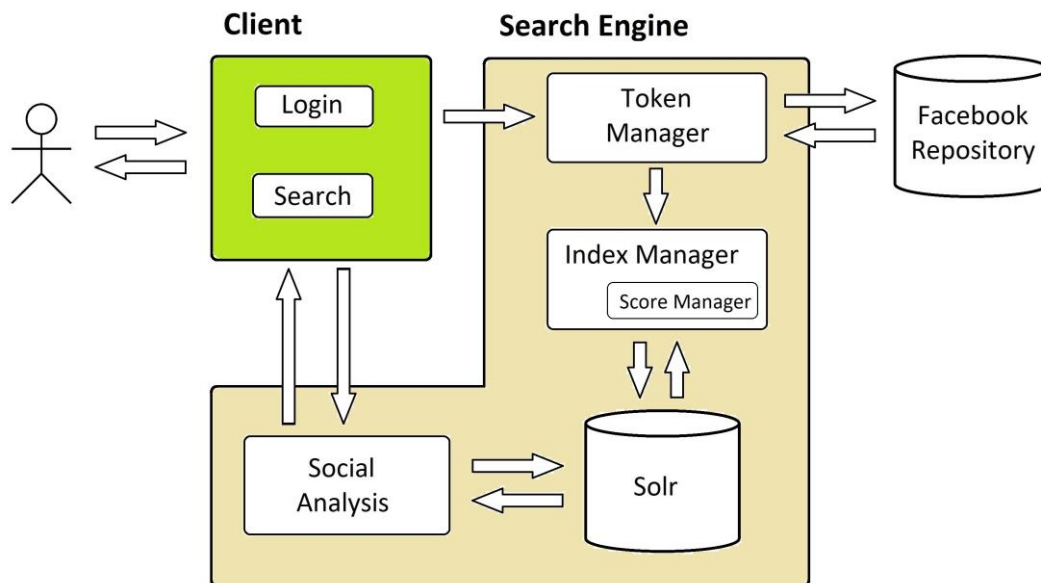


Figura 2.1 Architettura

Nella Figura 2.1 vediamo la struttura e i vari moduli pensati.

Il Client costituisce l'interfaccia dell'applicazione stessa, dove l'utente può utilizzare le sue funzioni che sono classiche di un motore di ricerca: quindi abbiamo la funzione di login e quella di search.

Il Search Engine contiene un insieme di moduli che comunicano tra loro. Quello che succede è che se l'utente decide di loggarsi, attiva il primo modulo, chiamato Token Manager, che recupera il token dell'utente (con la sua autorizzazione) e inizia a parlare con Facebook scaricando i dati dell'utente stesso.

Dopo di che i dati sono passati ad un secondo modulo, detto Index Manager, che ha il compito di formattare e organizzare tali dati prima di indicizzarli nell'istanza di Apache Solr contenuta nel motore di ricerca.

Il modulo chiamato Social Analysis invece comunica in modo bidirezionale con l'interfaccia (quindi con l'utente). Prende le richieste e, interfacciandosi con Solr e analizzando la similarità con gli altri utenti, restituisce i risultati in richiesti in un

ordine deciso dal modulo stesso. Guardiamo adesso questi moduli un po' più nel dettaglio.

2.2 Token Manager

Dopo che l'utente effettua il "login with Facebook", attraverso l'applicazione django-allouth installata all'interno di django, i dati dell'utente (nome, email e token Facebook) vengono memorizzati all'interno del database integrato in django SQLite. Successivamente (prima di reindirizzare l'utente nella pagina di ricerca) viene istanziato un thread che fa partire le funzioni all'interno del Token Manager prendendo in ingresso il token dell'utente in questione.

Il Token Manager è scritto in python e sfruttando le librerie Facebook [5] vengono scaricati i dati mediante le funzioni: *Graph.API('token')* e *get_object('fields')*.

La figura 2.2 mostra un esempio di script python dove vengono scaricati i post, con riferimenti al luogo e alla data, di un utente:

```
# 1) SCARICO I DATI IN UN FILE JSON
graph = facebook.GraphAPI (access_token = token)
json = graph.get_object(id='me', fields='posts{place,created_time})
```

Figura 2.2 Scarico i post da python

I dati scaricati per ogni utenti sono: età, luoghi visitati, posts, lista amici e pagine Facebook piaciute.

2.3 Index Manager

Dopo aver scaricato i dati, il Token Manager li passa in formato json all'Index Manager (anch'esso scritto in python) il quale formatta e organizza i dati come Document da indicizzare all'interno di Solr. Ciascun document non è altro che un dizionario che popola la repository di Solr.

2.3.1 Formattazione dei dati

Ciascun dato di Solr dell'applicazione ha un campo che ne identifica la tipologia di nome "doc_type"; per esempio se il dato in questione è un contenuto di Facebook (dove poi verrà effettuata la ricerca) allora sarà "doc_type: content".

Di seguito vengono elencati tutti i campi dei contenuti indicizzati per ogni tipologia:

- Likes di pagina (doc_type: page): user_id (id dell'utente), page_id, name, category_list, fan_count (numero totale di likes della pagina, indica la popolarità), data, genre, type (se si tratta di musica, film, libri ecc... oppure se è una pagina generica).
- Posts dell'utente (doc_type: post): post_id, user_id, message (il testo del post scritto dall'utente), type (se si tratta di una foto, un video, un link ecc...), link (url se il post è un link), full_picture (se è una foto o un video contiene il link dove dal quale scaricare il contenuto), place_id, data
- Luoghi visitati dall'utente (doc_type: place): user_id, place_id, city, country, created_time, zip.
- Etá (doc_type: age): user_id, age (l'anno di nascita).
- Lista amici (doc_type: friend_list): user_id, friends_count, friends_id (una lista di tutti gli id degli amici che hanno installato l'applicazione).
- Immagine del profilo (doc_type: profile_picture): user_id, link, link2 (url dove scaricare l'immagine del profilo con una risoluzione dimezzata).
- Contenuti su cui effettuare la ricerca (doc_type: content): user_id, user_profile_picture, type, image, message, link, height, width, aspect_ratio (se è una foto).

La figura 2.3 mostra un esempio: una foto postata da Tommaso Salerno indicizzata su Solr.

```
"doc_type":["content"],
"user_profile_picture":["https://scontent.xx.fbcdn.net/v/t1.0-0/p130x:
"type":["photo"],
"image":["https://scontent.xx.fbcdn.net/v/t1.0-9/37959476_12284738864:
"height":[157],
"width":[637],
"data":["2018-07-28T10:21:00Z"],
"aspect_ratio":[4.05732484076],
"user_id":["115599246033206"],
"message":["__null__"],
"user_name":["Tommaso Salerno"],
"id":["6cac4016-4ddf-4ce7-a054-43ff0c0f64c7"],
"_version_":1613044992846069762},
```

Figura 2.3 Foto indicizzata su Solr

Dunque “doc_type: content” sono i dati su cui viene fatta la ricerca, tutti gli altri servono esclusivamente per il modulo che verrà illustrato in seguito.

Ciascun contenuto è catalogato per tipologia:

- Photo
- Video (video postati dall’utente)
- Video_link (video condivisi, ad esempio da youtube)
- Link
- Facebook_page (anche le pagine di Facebook fanno parte dei contenuti)

2.3.2 *Score Manager*

All’interno di Index Manager è contenuto un altro modulo scritto in python che ha il compito di usare i dati passati dal Token Manager per calcolare uno score di similarità tra utenti ed indicizzare anch’esso su Solr. Questo dato verrà poi utilizzato dal modulo di Social Analysis durante la ricerca.

Lo score è calcolato per ogni coppia di utente e per ogni categoria.

Le categorie di similarità utilizzate sono:

- Luoghi
- Età
- Pagine di Facebook generiche (senza la categoria musica, film, ecc...)
- Musica
- Libri
- TV
- Film

L'algoritmo di score utilizzato è descritto nel paragrafo successivo.

2.3.3 Algoritmo di score

Il metodo consiste in un'analisi statistica ed è stato pensato per essere più generico possibile, quindi applicabile ad ogni tipologia.

L'idea è che ogni dato (luogo, pagina Facebook, ecc...) può essere sotto-categorizzato e visualizzato per semplicità come dei cerchi concentrici dove il cerchio più esterno rappresenta la categoria più generica. Per esempio la figura 2.4 mostra questo tipo di rappresentazione per un luogo:

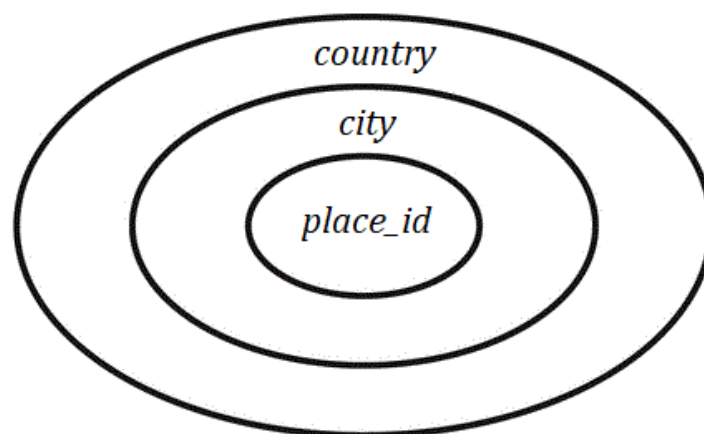


Figura 2.4 Categorie luogo

Questo dato è stato implementato come un dizionario.

In sostanza all'utente A e all'utente B (di cui vogliamo calcolare lo score) è associato un vettore con questi elementi (un vettore con tutti i luoghi visitati) e lo score è calcolato confrontando ogni coppia di elementi e controllando le varie intersezioni e occorrenze in modo il più possibile intelligente e sensato.

Quello che si fa è calcolare uno score per ogni sotto-categoria pesato per importanza, con dei pesi costanti predeterminati. Ciascuno di questi score rappresenta un contributo per lo score finale.

Diamo una descrizione di ciascun algoritmo di score.

Per i luoghi abbiamo già detto come sono stati rappresentati i dati.

Inizialmente abbiamo i due vettori contenenti tutti i luoghi visitati dagli utenti.

Per ogni sotto-categoria (a partire dalla più importante) come detto è impostato un coefficiente costante che indica il peso dello score (nel caso dei luoghi abbiamo 1.0 per i place_id, 0.5 per city e 0.25 per country), dopo aver inizializzato a 0 la variabile corrispondente allo score della sotto-categoria (il contributo) viene effettuato un ciclo:

- Si creano due vettori associati ai due utenti, dove ciascun elemento contiene il nome del luogo e il numero di volte in cui quel luogo è stato visitato. I vettori contengono l'unione di tutti i luoghi negli insiemi dei luoghi visitati dagli utenti, pertanto hanno pari dimensione.
- Si visitano i vettori e ogni volta si trova una corrispondenza (quindi se entrambi hanno visitato quel luogo almeno una volta) si incrementa lo score del minimo tra i due valori che indicano il numero di visite, inoltre si raddoppia il coefficiente. Altrimenti si incrementa una variabile "*norm*" inizialmente impostata a 0.
- Alla fine del ciclo il contributo sarà dato da:

$$coeff * \frac{score}{score+norm}$$

- Si tolgono dai vettori iniziali dei due utenti tutti i luoghi per cui è stata trovata una corrispondenza e si ricomincia il ciclo per la sotto-categoria successiva.

Alla fine ciascun contributo viene sommato e diviso per la somma di tutti e tre i coefficienti. Facciamo ora alcune considerazioni sul perché di questa implementazione:

- Lo score è incrementato con il minimo tra i due valori che indicano il numero di volte che gli utenti hanno visitato tale luogo perché ha senso pensare che l'affinità tra utenti aumenti se entrambi sono soliti frequentare lo stesso posto più volte. È scelto il valore minimo per mantenere lo score equilibrato (e sensato) nel caso in cui ci sia un forte gap tra i due valori. Immaginiamo per esempio che l'utente A si è geo localizzato nello stesso luogo una cinquantina di volte (perché magari ci va tutti i giorni) e invece l'utente B una volta sola (perché magari c'è passato per caso).
- Il coefficiente viene raddoppiato per fare sì che aumenti con il numero di corrispondenze trovate. Se in una sotto-categoria ottengo uno score massimo ma l'insieme è piccolo, è giusto che abbia un peso minore e viceversa.

Per i likes delle pagine Facebook l'approccio è identico con delle differenze. Considerando come sotto-categorie il `page_id`, il genere e la `category_list` della pagina (ordinati per importanza), nella prima iterazione (`page_id`) il contributo viene calcolato con un indice di Jaccard [6] dove ogni elemento (pagina) è rappresentato da un valore numerico corrispondente al reciproco dei "*fan_count*" della pagina. Questo per implementare l'idea che una corrispondenza di una pagina poco popolare dia un contributo maggiore allo score piuttosto che quella di una pagina più popolare.

L'indice di Jaccard è stato utilizzato per le page_id poiché sono uniche e non si contano occorrenze, per le restanti sotto-categorie l'algoritmo è analogo a quello dei luoghi.

Infine lo score per l'età è un semplice valore assoluto della differenza di età.

2.4 Social Analysis

Questo è scritto in Javascript, implementa la ricerca ed è il responsabile dell'ordinamento dei risultati. Esso comunica in maniera bidirezionale con l'interfaccia e possiede una lista contenente tutti gli score associati all'utente che sono stati precalcolati durante la fase di indicizzazione.

2.4.1 La ricerca

La ricerca viene effettuata sui contenuti di Facebook degli utenti indicizzati su solr, quei documenti per cui "doc_type: content". Essa è di tipo full text ed è effettuata su ciascun campo dei contenuti dando maggior importanza al campo più significativo del contenuto stesso. Infatti per esempio se trovo una corrispondenza nel titolo (campo "title") di un link, avrà maggior importanza di una corrispondenza trovata nella descrizione. L'interrogazione a solr è effettuata dal modulo "Social Analysis" attraverso una chiamata AJAX utilizzando la classe xmlhttpRequest() in JQuery. I pesi per dare importanza ai campi, vengono inseriti direttamente nella query da inviare a Solr.

La figura 2.5 mostra un esempio la porzione di richiesta a Solr per la ricerca full text:

```
console.log(score_weight);
xmlhttp.open("GET", 'http://localhost:8983/solr/demo/select?
q=doc_type:content AND ' + query_filter + ' (name:*' + query + '^10 and
place_name:*' + query + '^9 and genre:*' + query + '^8 and
category_list:*' + query + '^7 and description:*' + query + '^6 and
message:*' + query + '^5 and city:*' + query + '^4 and country:*' +
query + '^3) ' + score_weight + ' &start=' + start + '&rows=13', true);
xmlhttp.send();
```

Figura 2.4 Categorie luogo

Ogni chiamata restituisce i primi 12 contenuti, che saranno renderizzati nella pagina. Ovviamente è possibile visualizzare i contenuti restanti tramite un bottone “LOAD MORE” nell’interfaccia, che effettuerà un’altra chiamata AJAX restituendo i primi 12 contenuti e così via.

2.4.2 Analisi per similarità

Dopo aver ricevuto la query dall’utente, Social Analysis aggiorna gli score memorizzati moltiplicandoli per dei pesi che l’utente può impostare nell’interfaccia; un peso per categoria, dando più importanza alla similitudine per musica piuttosto che per luoghi per esempio (l’interfaccia è descritta nel prossimo capitolo).

Quindi interroga Solr aggiungendo alla ricerca full text una ricerca per corrispondenza di utente dove ciascuna corrispondenza è pesata con i rispettivi pesi appena aggiornati.

Di conseguenza Solr restituirà per prima quei contenuti associati all’utente con score più alto e Social Analysis li suggerirà all’utente.

Capitolo 3

L 'INTERFACCIA

L'interfaccia è stata realizzata facendo use dei seguenti linguaggi/librerie:

- HTML 5.0
- Javascript
- JQuery
- AJAX
- Bootstrap 4

In sostanza l'interfaccia è formata da una navbar fissa e una griglia di contenuti.

3.1 Navbar

Per un utente non loggato la navbar è costituita esclusivamente, oltre alla barra di ricerca, da un bottone di “*login with Facebook*” e un dropdown menu che dà la possibilità di impostare i filtri.

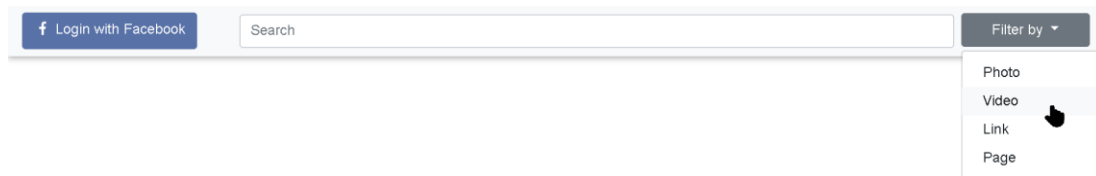


Figura 3.1 Navbar

La funzione di login porta l'utente sulla home page Facebook dove gli saranno richieste le credenziali per l'accesso.

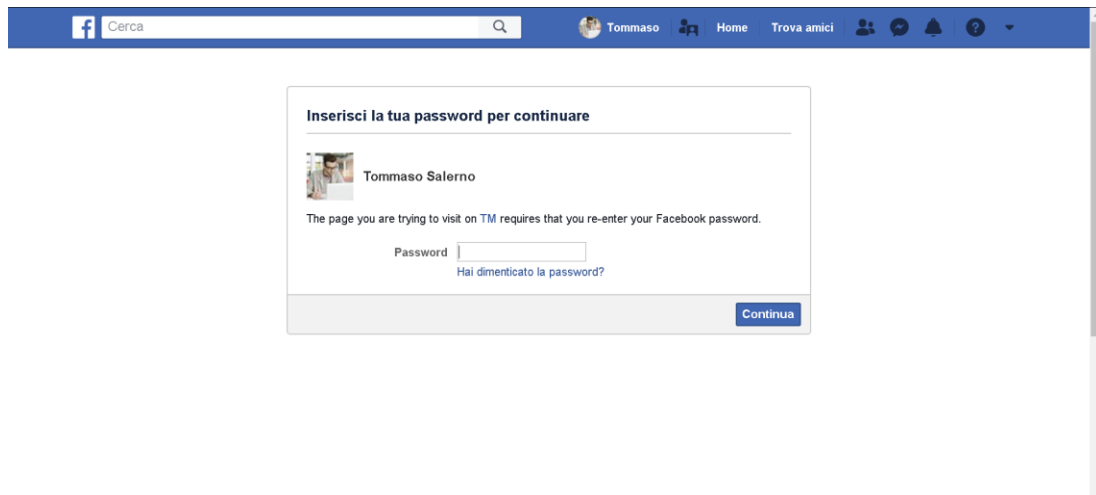


Figura 3.2 Home page Facebook

Dopo aver inserito email e password, l'utente verrà riportato sulla homepage dell'applicazione, dove al posto del tasto di login sarà visualizzata la sua immagine del profilo.

La figura 3.2 mostra la navbar dopo il login dell'utente:

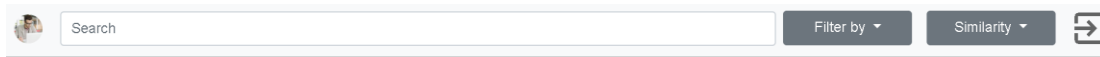


Figura 3.3 Navbar

Come si può notare oltre alla funzione di logout (in fondo a destra), è comparso il bottone “*Similarity*”. Si tratta di un menu a cascata che, attraverso degli sliders, dà la possibilità all'utente di pesare ciascuno score di similarità per la ricerca.

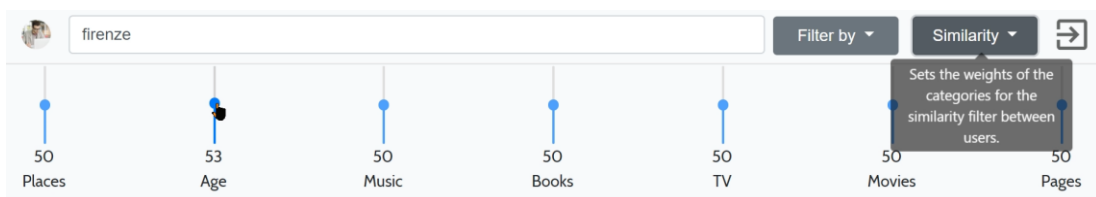


Figura 3.4 Sliders

Cambiando il peso cambieranno gli score e di conseguenza i risultati della ricerca saranno diversi.

3.2 Griglia

La griglia dei contenuti è stata realizzata con Masonry [7]. Una libreria JavaScript, che permette di implementare un layout a griglia nel proprio sito Web in cui gli elementi vengono disposti in automatico per ottimizzare lo spazio occupato in verticale all'interno della pagina, allineandoli in modo del tutto simile al processo di costruzione di un muro di mattoni, da cui il nome della libreria.

Ciascun contenuto nella griglia può assumere dimensioni diverse: verticale, orizzontale o quadrata. La griglia è composta da 4 colonne in generale. Inizialmente si renderizza ciascun contenuto in modo orizzontale o verticale in maniera casuale. Ma se questo crea dei 'buchi' vuoti allora la pagina visualizza alcuni contenuti come quadrati per rendere tutto simmetrico.

Mostriamo ora i contenuti che vengono visualizzati nella griglia.

Le foto mostrano le informazioni dell'utente, la data ed eventualmente il luogo:

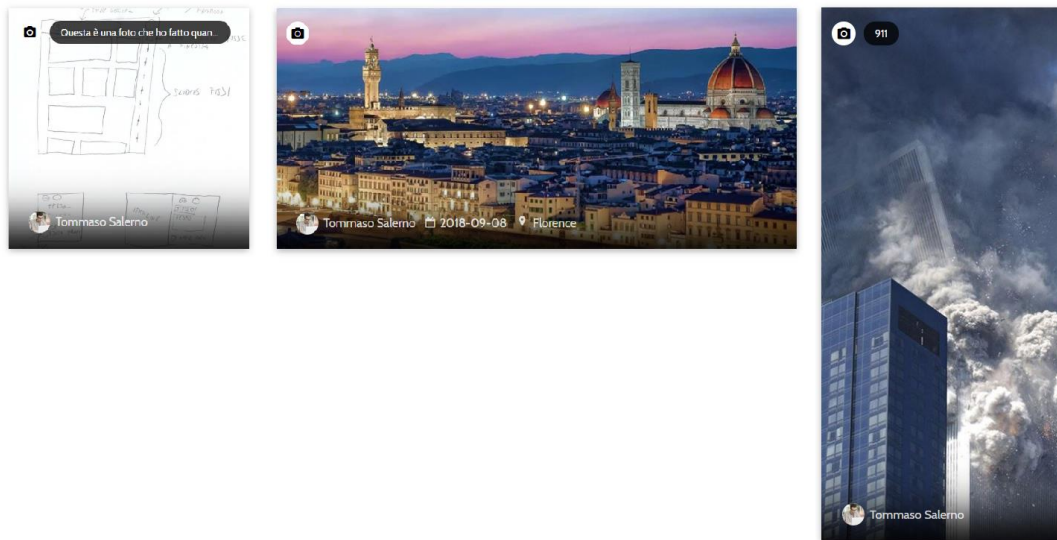


Figura 3.5 Foto

In alto viene mostrata un'icona che rappresenta la tipologia di contenuto seguita eventualmente dal testo inserito dall'utente nel momento della condivisione su Facebook.

I video registrati dall'utente sono rappresentati esclusivamente come quadrati:

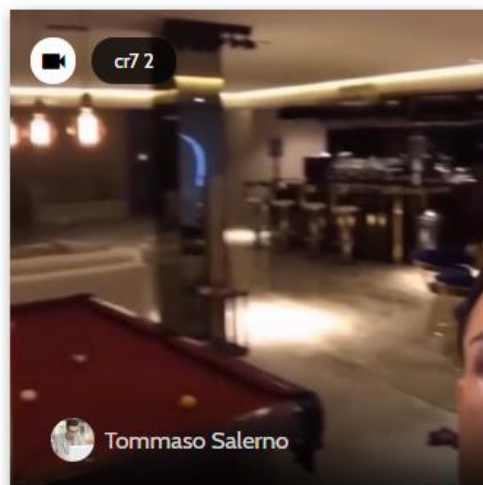


Figura 3.6 Video

Abbiamo poi i video condivisi dall'utente:

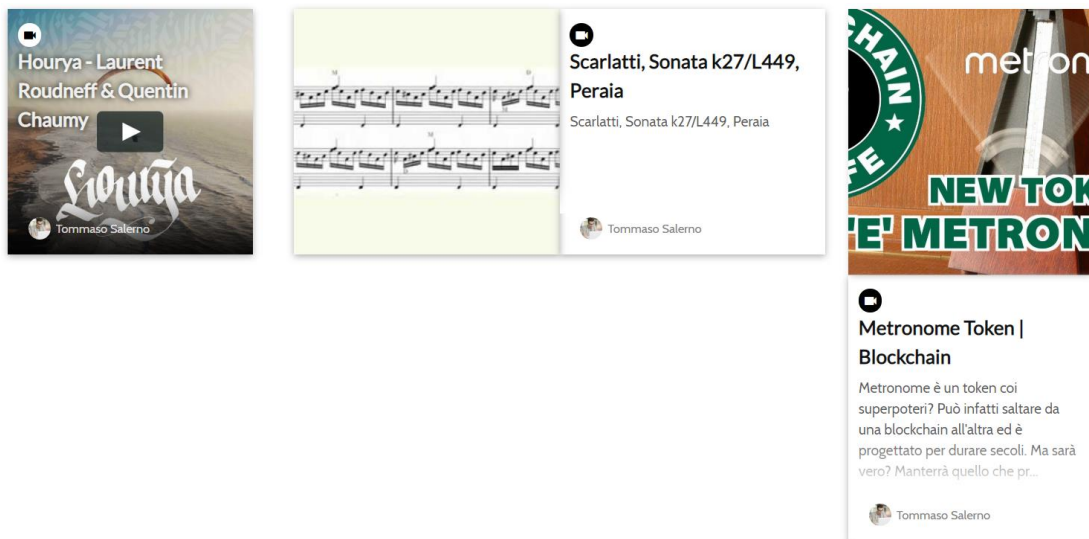


Figura 3.7 Video condivisi

I link postati:

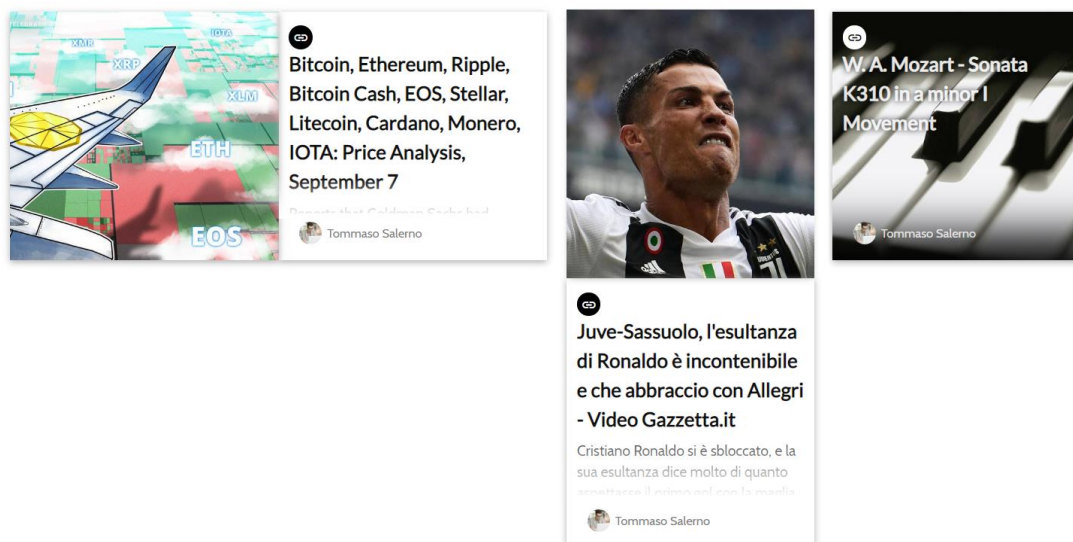


Figura 3.8 Link

E infine l'insieme di pagine Facebook seguite dagli utenti:

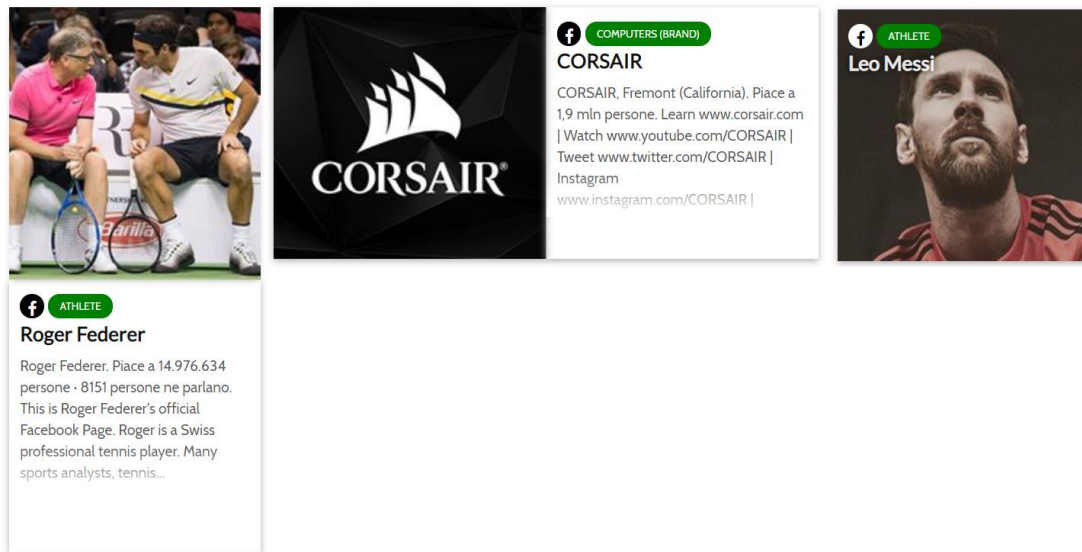


Figura 3.8 Pagine Facebook

Si noti, come già detto per le foto, che la tipologia di ciascun contenuto è indicata dall'icona in alto a sinistra. Cliccando su tale icona si attiva il filtro specifico per quella tipologia di contenuto.

Per quanto riguarda i file multimediali è visualizzato il testo del post mentre le pagine Facebook mostrano la categoria o il genere della pagina stessa.

3.3 Un semplice esempio

Mostriamo con un esempio come funziona l'ordinamento dei contenuti.

La figura 3.9 mostra l'utente corrente e gli score con altri due utenti per le categorie di musica e luoghi.




	 Tommaso Salerno	 Margaret Lisen	 Emiliano Braccini
<i>Places</i>		<i>0.3908</i>	<i>0.1493</i>
<i>Music</i>		<i>0.0</i>	<i>0.4564</i>

Figura 3.9 Score

Dunque Tommaso ha con Margaret uno score di similarità dei luoghi importante, mentre con Emiliano la similarità è molto alta nella musica. Se durante una ricerca l'utente aumenta il peso di similarità di musica, i risultati che vedrà per prima saranno quelli associati all'utente Emiliano Braccini (figura 3.10).

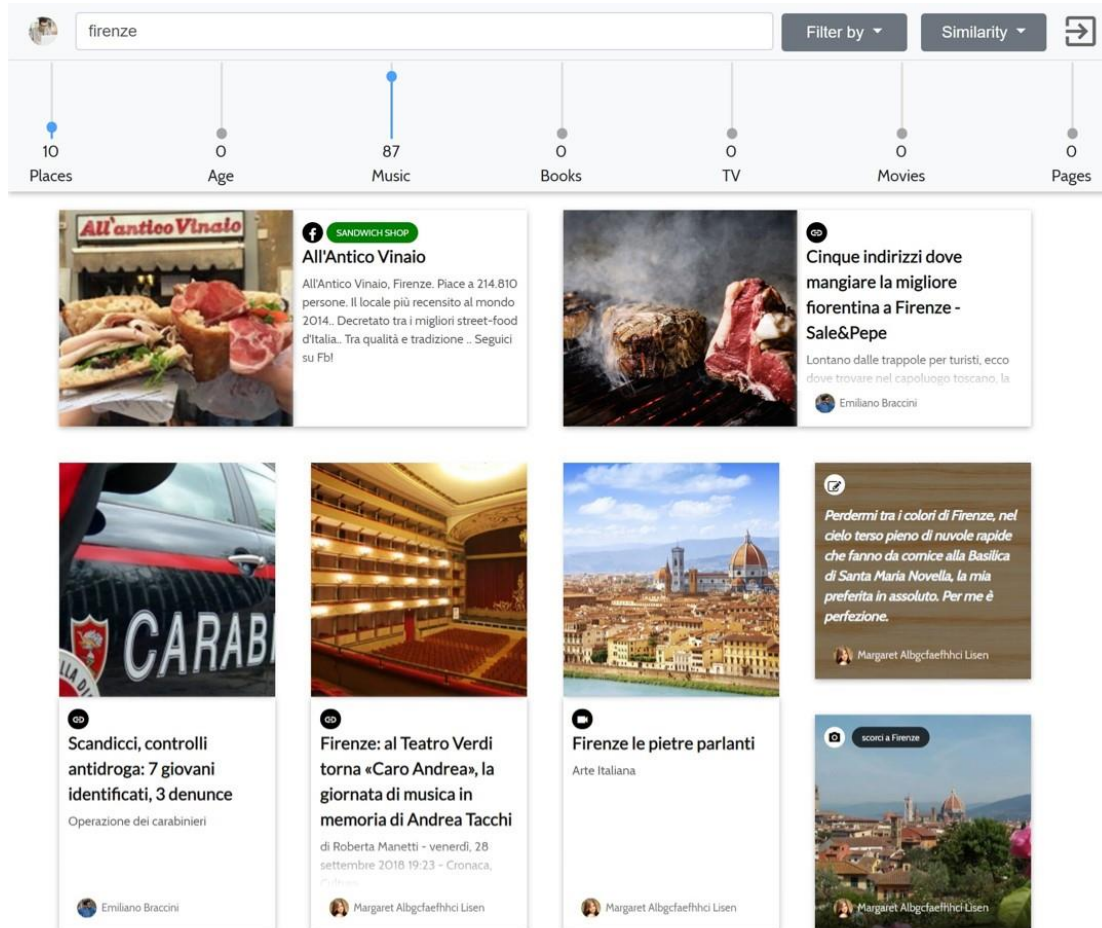


Figura 3.10 Ricerca

Si nota come tutti gli altri score sono stati azzerati. Questo fa sì che non ci sia nessuna preferenza per le altre tipologie di score. Infatti azzerando tutti gli score la ricerca è puramente full text, senza che il Social Analysis faccia un ordinamento dei contenuti.

Analogamente incrementando lo score dei luoghi i contenuti con priorità maggiore sono quelli associati a Margaret, come si vede in figura 3.11.

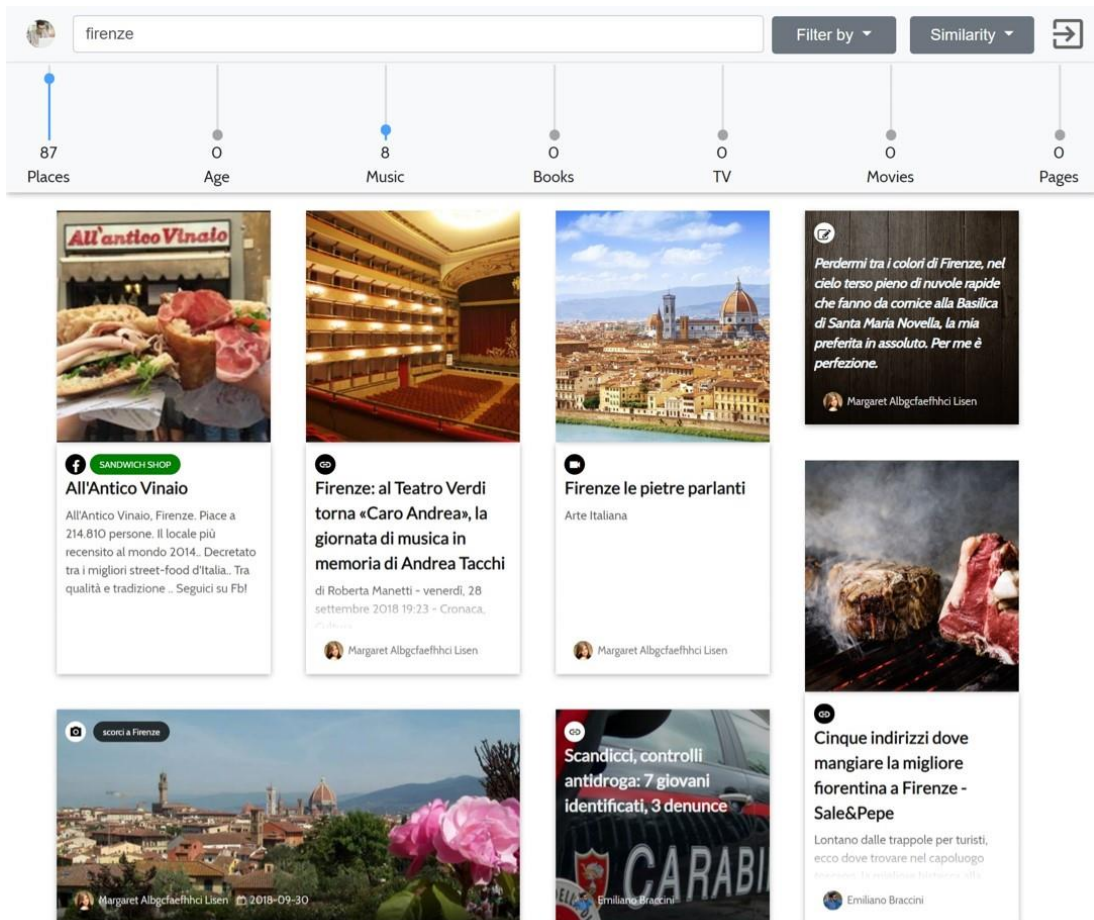


Figura 3.10 Ricerca

CONCLUSIONI

In questa tesi abbiamo descritto un sistema per un possibile ordinamento di contenuti richiesti dall'utente in un motore di ricerca, cercando inizialmente di far comprendere quanto sia importante per un motore di ricerca andare in contro a quelli che sono gli interessi dell'utente. Abbiamo introdotto l'idea progettuale descrivendone l'architettura e gli strumenti utilizzati per implementare l'idea della tesi.

Sicuramente il progetto potrà offrire spunti per nuove tecnologie che possano rendere le ricerche, su la vastissima massa di dati contenuti sul Web, sempre più efficienti.

Tra le idee di sviluppi futuri, ad esempio, ci potrebbe essere quella di estendere il supporto ad altri Social Network, l'utilizzo di Wordnet per implementare un sistema di espansione di query, introdurre un sistema di suggerimenti di ricerche attraverso una cronologia oppure migliorare e affinare gli algoritmi di score sfruttando la stessa cronologia.

BIBLIOGRAFIA

- [1] “Does twitter matter for news sites?”,
<https://www.parse.ly/resources/data-studies/authority-report-10/>.

- [2] “The Big Picture – How Django is structured”,
<https://djangobook.com/tutorials/django-overview/>

- [3] “OAuth 2.0: The Complete Guide”,
<https://auth0.com/blog/oauth2-the-complete-guide/>

- [4] “Apache Solr - Querying Data”,
https://www.tutorialspoint.com/apache_solr/apache_solr_querying_data.htm

- [5] “Facebook SDK for Python”,
<https://facebook-sdk.readthedocs.io/en/latest/api.html>

- [6] “Jaccard index”,
https://en.wikipedia.org/wiki/Jaccard_index

- [7] “Understanding Masonry Layout”,
<https://www.sitepoint.com/understanding-masonry-layout/>

INTRODUZIONE.....	2
 CAPITOLO 1: TECNOLOGIE	3
1.1 Django.....	3
1.2 OAuth2.....	5
1.2.1 Autenticazione three-legged.....	5
1.3 Facebook Graph API	8
1.4 Apache Solr.....	9
1.4.1 Principi basilari	9
1.4.2 Processo di analisi	9
1.4.3 Operazioni di indicizzazione.....	10
1.4.4 Comandi di ricerca	10
1.2.5 Solr schema	11
 CAPITOLO 2: L'APPLICAZIONE	13
2.1 Architettura dell'applicazione.....	14
2.2 Token Manager	15
2.3 Index Manager	15
2.3.1 Formattazione dei dati.....	16
2.3.2 Score Manager	17
2.3.3 Algoritmo di score.....	18
2.4 Social Analysis	21
2.4.1 La ricerca.....	21
2.4.2 Analisi per similitudine	22
 CAPITOLO 3: L'INTERFACCIA	22
3.1 Navbar.....	22
3.2 Griglia	23
3.3 Un semplice esempio	29
 CONCLUSIONI.....	35