Università degli studi
di Modena e Reggio Emilia

Dipartimento di Scienze Fisiche, Informatiche
e Matematiche

REALIZZAZIONE DI UN MOTORE DI RICERCA SEMANTICO BASATO SUL CONTESTO

Tesi di Laurea in Informatica

Relatore:
Ing. Riccardo Martoglia

Laureando:
Marco Valerio Manzini

Anno Accademico 2013/2014

Ringraziamenti

Ringrazio l'Ing. Riccardo Martoglia, per la sua disponibilità, ed il grande aiuto che mi ha fornito grazie alle sue competenze didattiche e scientifiche.
Un grande ringraziamento va alla mia compagna Maria Chiara, che mi ha sostenuto in tutti gli anni di corso sopportandomi ed aiutandomi a superare i momenti più difficili con il suo amore ed il suo sorriso.
Infine ringrazio i miei genitori, per avermi cresciuto nel migliore dei modi, insegnandomi cosa è giusto e cosa è sbagliato; e per avermi aiutato sia moralmente che economicamente ad arrivare fin qui.

Parole chiave

Contesto
Context-Aware
Similarità
Profilo
Ranking

Indice

Introduzione

Negli ultimi vent'anni c'è stata una notevole crescita dell'interesse nell'usare il contesto dell'utente per aumentare le prestazioni del sistema di Information Retrieval, in termini di precisione rispetto User Information Need (UIN) dei documenti restituiti all'utente, e del conseguente ranking dei documenti rilevanti.

Infatti, tutte le attività d'informazione occupano un posto all'interno di un contesto che interessa il modo in cui le persone accedono alle informazioni, interagiscono con il sistema di recupero, valutano e prendono decisioni riguardo i documenti recuperati[10].

Una strategia contestualizzata può permettere a un sistema di Information Retrieval (IR) di: imparare e predire quali informazioni un ricercatore necessita, imparare come e quando le informazioni devono essere visualizzate, presentare i risultati collegandoli a informazioni precedenti e lavori in cui l'utente è coinvolto e inoltre decidere chi altro dovrebbe prendere le nuove informazioni[14].

L'uso del contesto all'interno di applicazioni di IR è l'obiettivo del progetto AMBIT[1] il quale si propone di studiare e sviluppare un'architettura software prototipale per lo sviluppo di applicazioni e sistemi dipendenti dal contesto, cioè strumenti in grado di fornire agli utenti servizi che siano personalizzati proprio in base al contesto nei quali essi si trovano ad operare.

La tesi riguarda lo sviluppo di un piccolo motore di ricerca semantico, influenzato dalla presenza del contesto dell'utente, che corrisponde dalla sua cronologia di navigazione.

Nell'implementazione del motore di ricerca ci poniamo nello scenario di un sito di e-commerce, perciò formato da pagine web, le quali descrivono tutte un prodotto presente sul sito.

[1] Algorithms and Models for Building context-dependent Information delivery Tools

L'obiettivo della tesi è quello di riuscire a trovare un algoritmo che implementi il processo di ricerca delle pagine rilevanti per un utente nel miglior modo possibile; in modo tale che restituisca un ranking delle pagine in linea con il contesto dell'utente, aumentando così i valori di pecision e recall[2] del sistema di IR stesso.

A tale proposito viene usato, oltre al software di AMBIT, anche un web service basato su COGITO[3], per estrapolare dalle pagine web le informazioni rilevanti per classificarle; tali informazioni saranno utilizzate all'interno del motore di ricerca per costruire diversi metodi che classificheranno le pagine web in base al profilo dell'utente.

Infine si combineranno queste classificazioni tra di loro, tramite un algoritmo di ranking fusion, per riuscire a migliorare il ranking delle pagine in termini di precision del sistema.

Tutto ciò sarà poi spiegato nel dettaglio nei capitoli 2 e 3 della tesi di ricerca.

Riassumendo, gli obiettivi che mi prefiggo di raggiungere in questa tesi sono i seguenti:

- **Recupero e manipolazione dei dati** riguardanti le pagine web HTML del sito di e-commerce e del profilo dell'utente estrapolati tramite i metodi di AMBIT.

- **Recupero e manipolazione dei dati** riguardanti le pagine web XML del sito di e-commerce e del profilo dell'utente estrapolati dal web service basato su COGITO.

- **Studio e modifica** dell'algoritmo di similarità usato da AMBIT, per adattarlo alle esigenze dell'applicazione.

- **Studio, progettazione e realizzazione** di un algoritmo che costruisca una misura di similarità basata sulle classi IPTC estratte dai dati di COGITO.

- **Studio, progettazione e realizzazione** di un algoritmo di ranking fusion, per combinare tra di loro diversi ranking delle pagine web.

[2] vedi sottosezione 1.5.3

[3] Tecnologia semantica della ditta Expert System

- **Valutazione** dell'efficacia degli algoritmi implementati tramite l'analisi dei risultati ottenuti attraverso le misure di: precision, recall, F measure e precision ad ogni livello standard di recall.

La tesi sarà strutturata in capitoli, ordinati seguendo l'ordine cronologico del processo di studio e sviluppo del motore di ricerca:

Capitolo 1 - Stato dell'arte. Cos'è il contesto (Sezione 1.1), le classificazioni di contesto(Sezione 1.2), cos'è un sistema context-aware e quali caratteristiche possiede(Sezione 1.3), quali contesti si intendono estrapolare in alcuni sistemi context-aware(Sezione 1.4), cos'è il modello spazio vettoriale e cosa si intende per precision e recall (Sezione 1.5), come il contesto impatta nella fase di ranking(Sezione 1.6), cos'è il progetto AMBIT e quali sono i suoi obiettivi(Sezione 1.7).

Capitolo 2 - Progetto del software. Presentazione generale del progetto realizzato (Sezione 2.1), esposizione della creazione di glossari di inverted index e l'uso dei dati estratti dal web server basato su COGITO (Sezione 2.2), esposizione delle funzioni di similarità usate all'interno del programma e della funzione di ranking fusion (Sezione 2.3).

Capitolo 3 - Implementazione del software. Metodi di estrazioni del testo da pagine HTML e da pagine XML (Sezione 3.1), estrazione dei termini rilevanti dal testo estratto e implementazione dei glossari (Sezione 3.2), implementazione degli Inverted Index (Sezione 3.3), realizzazione delle funzioni di similarità e implementazione dell'algoritmo di ranking fusion (Sezione 3.4).

Capitolo 4 - Prove sperimentali e risultati ottenuti. L'insieme di dati utilizzati per costruire il profilo dell'utente e il sito e-commerce (Sezione 4.1), le prove sperimentali effettuate sul profilo dell'utente 1 (Sezione 4.2), le prove sperimentali effettuate sul profilo dell'utente 3 (Sezione 4.3), la valutazione finale sulle prove sperimentali effettuate (Sezione 4.4).

Capitolo 5 - Conclusioni e sviluppi futuri.

Appendice A - Archivio dei codici. Contenente i codici dei moduli utilizzati.

Capitolo 1

Stato dell'arte

In questo capitolo saranno presentati le basi teoriche che mi hanno aiutato a capire e a realizzare al meglio il programma che ho implementato e che spiegherò nel dettaglio nei capitolo successivi.

In particolare spiegherò:

- Cosa si intende per contesto. (Sezione 1.1)
- Quali sono le classificazioni di contesto in diversi ambiti di utilizzo. (Sezione 1.2)
- Cosa si intende per sistema context-aware e quali caratteristiche un buon sistema contex-aware deve avere. (Sezione 1.3)
- Quali contesti si intendono estrapolare solitamente nei sistemi context-aware. (Sezione 1.4)
- Cos'è il modello spazio vettoriale e quali sono i suoi indici di valutazione. (Sezione 1.5)
- Come l'uso del contesto impatta nella fase di ranking dei risultati finali. (Sezione 1.6)
- Cos'è il progetto AMBIT e quali obiettivi si pone di raggiungere. (Sezione 1.7)

1.1 Il Contesto

Il termine "contesto" è stato definito da molti ricercatori, in particolare Dey et al.[6] hanno valutato e sottolineato le debolezze di queste definizioni, affermando che la definizione data da Schilit e Theimer[19] era basata su esempi e non poteva essere usata per identificare nuovo contesto.

Inoltre Abowd et al.[1] hanno affermato che le definizioni date da Brown[4], Franklin e Flachsbart[7], Rodden et al.[17], Hull et al.[9] e Ward et al.[24] usavano dei sinonimi per riferirsi al contesto, come 'ambiente' o 'situazione' rendendo queste definizioni troppo specifiche e non applicabili al fine di identificare il contesto nel suo senso più ampio[15]; per questo motivo Abowd et al.[1] fornirono la seguente definizione di contesto:

Contesto è qualsiasi informazione che può essere usata per caratterizzare la situazione di un' entità. Un'entità è una persona, un posto, o un oggetto che è considerato rilevante nell'interazione tra un utente e un'applicazione, includendo l'utente e l'applicazione stessi.

Avendo definito cosa s'intende per contesto distinguerò quali tipi di contesti esistono e come vengono classificati.

1.2 Classificazioni di contesto

Molti ricercatori hanno proposto diverse categorizzazioni di contesto a seconda dell'applicazione che stavano considerando. Personalmente ne prenderò in considerazione quattro, in particolare: due molto generali (1.2.1, 1.2.2) e due specifiche per l'Information Retrieval (1.2.3, 1.2.4).

1.2.1 Classificazione di Abowd et al.

Abowd et al.[1] hanno introdotto una categorizzazione generale applicabile a qualsiasi tipo di applicazione; in particolare, loro identificano due categorie di contesto:

- **Primario.** al cui interno sono presenti quattro tipi di contesto; Locations, Identità, Tempo e Attività.

- **Secondario.** contenente il contesto che può essere derivato usando quello primario.

Per esempio se noi sapessimo l'identità di un utente (contesto primario), potremmo ricavarne dei contesti secondari come il numero di telefono o l'indirizzo.

L'unico problema di questa classificazione è che, com'è stato rilevato da Perera et al.[15], sebbene sia corretta, non riesce ad identificare alcuni tipi di contesto che sono presi in considerazione da una eventuale applicazione.

1.2.2 Classificazione di Perera et al.

Perera et al hanno definito uno schema di categorizzazione del contesto che può essere usato per classificare il dato di un contesto in termini di come è stato acquisito, inoltre lo stesso dato può essere considerato primario in uno scenario e secondario in un altro. Loro definiscono due categorie di contesto:

- **Contesto primario.** ogni informazione recuperata direttamente dai sensori; perciò senza usare contesti già esistenti e senza fare alcuna fusione di dati.
- **Contesto secondario.** ogni informazione che può essere ricavata usando il contesto primario, da operazioni di fusioni di dati o da operazioni di data retrival come la chiamata a web services.

Il problema delle categorizzazioni di contesto 1.2.1, 1.2.2, risiede nel fatto che sono troppo generali rispetto lo specifico ambito dell'Information Retrieval.

1.2.3 Classificazione di Ingwersen et al.

Ingwersen et al.[10][11] hanno proposto un modello nidificato di contesto fatto apposta per l'Information Retireval, composto da sei dimensioni:

- **Strutture intra-oggetto.** Si riferiscono al contesto ottenuto da ogni documento dove le immagini sono contestuali ad un testo circostante e i paragrafi fungono da contesto per le loro frasi e le loro parole.

  Per esempio: Termini, frasi, componenti di immagini, paragrafi, sezioni

- **Contesti inter-oggetto.** Riguardano le proprietà del documento, come: i riferimenti, le citazioni, i collegamenti interni ed esterni, che danno e prendono il contesto da altri oggetti.

  Per esempio: links, citazioni

- **Interazioni/contesti di sessione.** Composto dalle interazioni dell'utente con il sistema, questi tipi di interazioni formano una ricca rete di potenziali informazioni riguardanti le preferenze, lo stile, l'esperienza, la conoscenza nonché gli interessi dell'utente.

- **Contesti sociali, sistemici, mediatici, di attività di lavoro, concettuali, emotivi.** Sono collegati ad aspetti socio-organizzativi relativi all'utente quali: informazioni riguardanti le relazioni sociali dell'utente, gli interessi dell'utente, oppure le percezioni emotive dell'utente.

- **Contesti tecnico-fisici, societari, ed economici.** Corrispondono alle infrastrutture sociali prevalenti, come: convenzioni culturali, preferenze organizzative o tradizioni specifiche per ogni dominio.

- **Contesti storici.** È una forma temporale di contesto che comprende tutte le esperienze passate degli utenti.

1.2.4 Classificazione di Tan et al.

Un'altra categorizzazione di contesto può essere trovata dalla valutazione fatta da Tan et al.[22] i quali analizzano: delle guide sul contesto, diverse applicazioni mobili per il turismo, per il settore medico e per gli uffici, così da ricavarne i contesti più usati e costruire il seguente modello:

- **Contesti di posizione (prossimità).** Descrivono il variare dei dati in base alla posizione dell'utente.
  Di cui ne fanno parte:

  - Posizione attuale
  - Posizioni di attrazioni vicine
  - Velocità di andatura

  - Direzione di andatura
  - Mezzo di trasporto attuale

- **Contesti temporali.** Descrivono le caratteristiche temporali dell'applicazione.
  Di cui ne fanno parte:
  - Anno e giorno corrente
  - Eventi accaduti di recente
  - Stagione dell'anno

- **Contesti d'identità.** Si riferisce al profilo dell'utente.
  Di cui ne fanno parte:
  - Profilo – nome, data di nascita, paese d'origine, età, sesso
  - Lingua preferita
  - Preferenze generali dell'utente

- **Contesti ambientali.** Descrive cosa sta intorno all'utente.
  Di cui ne fanno parte:
  - Tempo meteorologico – temperatura, umidità ecc..
  - Condizione del traffico

- **Contesti sociali.** Si riferisce alla condizione dell'utente all'interno della società
  Di cui ne fanno parte:
  - Persone vicine
  - Amici dell'utente
  - Recensioni di amici
  - Video e immagini caricate da altri

- **Contesti della rete.** Descrive le risorse di rete disponibili per le applicazioni.
  Di cui ne fanno parte:
  - Tipo di rete

  - Velocità della rete

- **Contesti delle attività.** Si riferisce ai compiti attualmente eseguiti dagli utenti.

- **Contesti della periferica.** Descrive le capacità e le caratteristiche del dispositivo in uso.
  Di cui ne fanno parte:

  - Tipo di periferica (marca, modello)
  - Memoria disponibile

- **Contesti fisiologici.** Si riferisce alle informazioni sullo stato fisiologico degli utenti.
  Di cui ne fanno parte:

  - Pressione sanguigna
  - Frequenza cardiaca.

- **Contesti cognitivi.** Descrive lo stato mentale degli utenti quando si utilizzano applicazioni context-aware.
  Di cui ne fanno parte:

  - Emozioni degli utenti
  - Stato mentale attuale
  - Livello di stress.

In conclusione come Tan et al.[22] affermano che, come ci si aspettava, si è osservato che le applicazioni context-aware adottano diversi tipi di contesto basati sullo scopo dell'applicazione e dei bisogni dell'utente.

Come dichiarato da Kaasinen[12], è difficile identificare i tipi di contesti necessari per le applicazione context-aware e siccome non esiste alcun insieme standard di tipi di contesto mirati per le applicazioni in ogni campo, la progettazione del contesto può essere migliorata solo considerando un insieme opportunamente completo di contesti corretti.

Ciò è supportato da Christensen et al.[5], i quali hanno spiegato che prendere molte informazioni contestuali non necessariamente aiuta il sistema context-aware a incontrare I bisogni dell'utente, al contrario, ciò può portare una diminuzione delle prestazioni del sistema stesso.

La chiave sta nell'adottare dei tipi di contesti appropriati, per l'applicazione che si sta costruendo.

1.3 I sistemi context-aware

Il contesto così come è definito e modellato in precedenza viene utilizzato per costruire applicazioni basate su di esso, identificate come applicazioni o sistemi context-aware.

Come la definizione di contesto anche, la definizione di sistema context-aware è stata definita da diversi ricercatori; Perera et al.[15] spiegano che questa definizione fu introdotta in origine da Schilit e Theimer[19] nel 1994; più tardi fu ridefinita da Rayan et al.[18] nel 1998; ma in entrambi i casi il focus era su applicazioni per il computer e sistemi specifici.

Come dichiarato da Abowd et al.[1] queste definizioni sono troppo specifiche e non possono essere usate per identificare se un sistema sia context-aware o no. Pertanto, Abowd et al.[1] hanno definito il termine context-awareness come segue:

Un sistema è context-aware se usa il contesto per fornire informazioni rilevanti e/o servizi all'utente, dove la rilevanza dipende dall'obiettivo dell'utente.

1.3.1 Componenti di un sistema context-aware

Abowd et al.[1] inoltre hanno identificato tre componenti che un buon sistema context-aware deve supportare:

- **Presentazione.** Il contesto deve poter essere usato per decidere quale informazione o quali servizi devono essere presentati all'utente.
  Per esempio, usando il contesto dell'utente si può presentargli in un certo ordine di rilevanza una serie di documenti che sta cercando.

- **Esecuzione.** Esecuzione automatica di servizi in base al contesto.
  Per esempio, usando il contesto dell'utente gli si può far apparire in modo automatico, su di una pagina web che sta visitando, dei banner pubblicitari correlati al suo contesto.

- **Marcatura.** Il contesto deve poter essere marcato per poter essere usato per operazioni di information retrieval successive.
  Per esempio tramite il contesto si può costruire un profilo dell'utente per essere usato successivamente.

1.4 Contesti usati in alcune applicazioni context-aware

In questa sezione ho ricercato, per alcune applicazioni context-aware per l'information retrieval e indagini sul contesto, quali contesti si intendono estrapolare al fine di riuscire a capire in che modo questi sistemi costruiscono il profilo dell'utente.

Come base di classificazione uso la categorizzazione dei contesti fatta da Tan et al.[22] siccome è la più completa tra quelle presentate e permette una categorizzazione veloce di quasi tutti i contesti nelle rispettive classi d'appartenenza.

La categorizzazione di Tan et al.[22] però non permette di classificare in modo chiaro il contesto storico dell'utente e le strutture intra-oggetto viste nel modello di Ingwersen et al.[10][11], per questo motivo integro la tabella con i contesti sopracitati ed inoltre ometto i contesti cognitivi, fisiologici e di dispositivo in quanto non sono utili per classificare applicazioni context-aware per l'information retrieval, ed inoltre come è stato mostrato da Tan et al.[22] questi contesti vengono utilizzati quasi esclusivamente in ambito medico.

Autori	*Dominio*	*Posizione*	*Temporale*	*Identità*	*Ambientale*	*Sociale*	*Rete*	*Attività*	*Storico*	*Strutture Intra-oggetto*
TREC 2013 Lugano	Information Retrieval	X		X					X	
Tan et al.	Information Retrieval	X	X	X	X	X				
Abowd et al.	Indagine sul contesto	X	X	X				X		
Lopes	Indagine sul contesto	X	X	X	X	X		X	X	X
Ingwersen et al.	Indagine sul contesto	X	X	X	X	X		X	X	X
Castelli et al.	Information Retrieval	X	X	X				X		
Palacio et al.	Information Retrieval	X		X				X	X	X
Shen et al.	Information Retrieval			X				X	X	X
Godoy et al.	Information Retrieval			X				X	X	X
Sieg et al.	Information Retrieval			X				X	X	X
Vallet et al.	Information Retrieval			X				X	X	

Figura 1.1: Contesti usati in diverse applicazioni cotext-aware.

Da quello che possiamo dedurre dalla tabella, è che le applicazioni context-aware per l'information retrieval utilizzano, per la maggior parte dei casi, contesti riguardanti:

- L'**identità** dell'utente (Profilo dell'utente, interessi)
- Le sue **attività recenti** (La query che ha specificato)
- Il suo **contesto storico** (Cronologia delle pagine visitate)
- Le **strutture intra-oggetto** dei documenti che ha visitato o che sta visitando

Una nota di riguardo va fatta ai documenti di Shen et al.[20], Godoy et al.[8], Sieg et al.[21], Vallet et al.[23], i quali dividono i contesti catturati tra :

- **Contesti a breve termine.** definiti come l'informazione che getta luce sulle attuali necessità di informazione di un utente in una singola sessione, dove una sessione può essere considerata come un periodo costituito da tutte le interazioni per lo stesso bisogno informazioni dell'utente (UIN).
- **Contesti a lungo termine.** si riferisce a informazioni generalmente stabili per un lungo periodo di tempo; ad esempio il profilo dell'utente (data di nascita, sesso, nazionalità, ecc...).

I contesti a breve termine sono più direttamente legate alla corrente esigenza di informazioni dell'utente e quindi dovrebbero essere più utili per migliorare la

ricerca corrente. In generale, il contesto a breve termine è più utile per migliorare la ricerca nella sessione corrente, ma non può essere così utile per le attività di ricerca in una diversa sessione.

I contesti a lungo termine possono essere applicati a tutte le sessioni, ma non è efficace quanto il contesto a breve termine per migliorare la precisione della ricerca per una particolare sessione.

Per questo motivo tutti e quattro i ricercatori cercano di usare al meglio i contesti a breve termine, integrando le lacune lasciate, tramite i contesti a lungo termine, per costruire così dei sistemi context-aware più efficaci nel reperire informazioni utili per l'utente in una determinata sessione di Information Retrieval.

1.5 Il modello spazio vettoriale e i suoi indici di valutazione

Attraverso l'analisi di diversi articoli riguardanti applicazioni context-aware ho notato che quasi tutti utilizzano il modello spazio vettoriale per costruire una misura di similarità tra i documenti, inoltre viene utilizzato anche dal software di AMBIT[1]; per questo motivo in questa sezione spiegherò cos'è il modello spazio vettoriale, e specificherò cosa si intende per precision e recall di un sistema di IR.

1.5.1 Il modello spazio vettoriale

Finora si tratta del modello di Information Retrieval più moderno e comune, grazie alla sua struttura permette una corrispondenza parziale tra la richiesta dell'utente e i documenti ricercati; generalmente la richiesta dell'utente consiste in testo libero, ed invece che predire quali documenti sono rilevanti per la richiesta o no, ordina i documenti in base al grado di similarità con la richiesta dell'utente.

L'idea di base di questo modello sta nel vedere tutti gli oggetti, che si trattino di documenti, richieste o termini, come dei vettori di grandi dimensioni.
In particolare, dopo la fase di preprocessamento del nostro oggetto, restano (t)

[1] Algorithms and Models for Building context-dependent Information delivery Tools

termini unici, ai quali, per ogni termine (i) e per ogni documento (j), gli vengono assegnati dei pesi w_{ij} (variabili tra 0 e 1) che formeranno lo spazio vettoriale di quel particolare oggetto.
Perciò sia il vettore che la richiesta dell'utente saranno rappresentati da un vettore di dimensione (t) :

$$\vec{d_j} = (w_{1j}, w_{2j}, ..., w_{tj}) \qquad w_{ij} \geq 0$$

$$\vec{q} = (w_{1q}, w_{2q}, ..., w_{tq}) \qquad w_{iq} \geq 0$$

La similarità tra la richiesta dell'utente e l'insieme di documenti, viene calcolata attraverso la così detta **similarità coseno**, la quale misura la similarità in modo da essere proporzionale all'angolo α che c'è tra di loro nella rappresentazione sullo spazio vettoriale.
In questo modo si cattura la similarità tra due vettori calcolando il coseno dell'angolo α.

$$\cos(\alpha) = \begin{cases} 0 & \text{se } \alpha = 90^\circ \text{ (nessuna similarità)} \\ 1 & \text{se } \alpha = 0^\circ \text{ (massima similarità)} \end{cases}$$

Ma come si dovrebbe calcolare il la similarità coseno in concreto ?
Attraverso la seguente formula chiamata anche *prodotto interno normalizzato*:

$$sim(d_j, q) = \frac{\vec{d_j} \cdot \vec{q}}{|\vec{d_j}| \times |\vec{q}|} = \frac{\sum_{i=1}^{t} w_{ij} \times w_{iq}}{\sqrt{\sum_{i=1}^{t} w_{ij}^2} \cdot \sqrt{\sum_{i=1}^{t} w_{iq}^2}} = \begin{cases} \simeq 0 & \text{(quasi nessuna similarità)} \\ \simeq 1 & \text{(quasi massima similarità)} \end{cases}$$

Grazie all'uso del calcolo della similarità, tra la richiesta dell'utente ed ogni documento nella collezione, è possibile:

- Ordinare i risultati secondo la rilevanza che è stata calcolata.
- Imporre una soglia di similarità, in questo modo la grandezza dei documenti recuperati può essere controllata.
- Trovare i k-vicini ad un certo valore di similarità, garantendo un recupero dei documenti più veloce.

1.5.2 Term Frequency - Inverse Document Frequency

Term Frequency

Fino ad ora abbiamo considerato i pesi dei termini (w_{ij}) come valori binari, cioè:

$$w_{ij} = \begin{cases} 1 & \text{il termine occorre nel documento} \\ 0 & \text{il termine non occorre nel documento} \end{cases}$$

A causa di questo tutti i termini sono considerati equamente importanti.
Per impedire questa uguaglianza tra termini viene utilizzata la misura di **Term Frequency** (TF), la quale assegna valori più alti a parole che sono più prominenti in determinati documenti. In particolare la Term Frequency misura quanto più un termine descrive meglio i contenuti di un documento; e può essere calcolata in tre diversi modi:

- Frequenza grezza(brute force): calcola la frequenza f_{ij} delle occorrenze dell' i-esimo termine nel j-esimo documento.

- Frequenza normalizzata: calcola la frequenza normalizzata f_{ij} delle occorrenze dell' i-esimo termine nel j-esimo documento.
  Attraverso la formula :
  $$f_{ij} = \frac{freq_{ij}}{max(freq_{ij})}$$
  $$(freq_{ij} = frequenza\ grezza\ del\ termine\ k_j\ nel\ documento\ d_j)$$

- Frequenza logaritmica: calcola il peso della frequenza logaritmica f_{ij} dell' i-esimo termine nel j-esimo documento.
  Attraverso la formula :
  $$f_{ij} = \begin{cases} 1 + log_{10} freq_{ij} & \text{se la } freq_{ij} > 0 \\ 0 & \text{altrimenti} \end{cases}$$

Inverse Document Frequency

I termini che sono più frequenti sono meno informativi rispetto ai termini rari; tramite la misura di **Document frequency** espressa come df_t cattureremo il

numero di documenti nella collezione in cui appare il termine t.
Considerando dei termini di una richiesta di un utente che sono rari nella collezione di documenti, allora un documento contenete quei termini ha più probabilità di essere rilevante per quella richiesta, ma non è un indicatore di rilevanza sicuro. Perciò è stato definito il così detto Inverse Document Frequency (IDF) di t in N documenti come[2]:

$$idf_t = log_{10}(N/df_t)$$

TF-IDF

Grazie al prodotto dei valori di Term Frequency e Inverse Document Frequency è possibile ottenere un peso dei termini che viene influenzato da:

- Il numero di occorrenze del termine dentro il documento.
- Dalla rarità del termine all'interno della collezione di documenti.

$$w_{t,d} = tf_{t,d} \times idf_t$$

Avendo così un ranking più preciso dei documenti della collezione.

1.5.3 Precision e Recall

Precision e Recall sono misure necessarie per valutare l'efficacia di un sistema di Information retrieval.
Data una certa richiesta di informazione definiamo:

- **R**: come l'insieme dei documenti rilevanti.
- **A**: come l'insieme delle risposte (i documenti che sono effettivamente recuperati dal sistema di information retrieval).
- **|Ra|**: come l'insieme di documenti facente parte dell'intersezione tra R e A.

Grazie a queste definizioni possiamo dichiarare cosa si intende per Precision e Recall; in particolare:

[2]Viene usato il logaritmo per smorzare l'effetto di idf

- **Recall** : La frazione dei documenti rilevanti che sono stati ritrovati; calcolata attraverso la formula

$$Recall = \frac{|Ra|}{R}$$

- **Precision** : La frazione dei documenti ritrovati che sono rilevanti; calcolata attraverso la formula

$$Precision = \frac{|Ra|}{A}$$

1.6 L'uso del contesto nella fase di ranking

In tutti i documenti analizzati, trattanti nuovi metodi di Information Retrieval basati sull'uso del contesto (TREC 2013 Lugano[16], Shen et al.[20], Godoy et al.[8], Sieg et al.[21], Vallet et al.[23]), concordano sul fatto che l'uso del contesto corretto porta ad un sostanziale aumento di precisione dei documenti ritrovati; in altri termini, le misure di Precision e Recall[3] tendono ad aumentare, permettendo di avere un sistema più efficiente ed efficace nella soddisfazione dell' User Information Need.

Il problema principale nell'uso del contesto in operazioni di Information Retrieval è di riuscire ad individuare il contesto corretto, in quanto, come affermano Vallet et al.[23], non tutte le preferenze degli utenti sono rilevanti in tutte le situazioni; infatti è risaputo che le preferenze umane sono:

- Complesse.
- Multiple.
- Eterogenee.
- Variabili.
- Contraddittorie tra di loro.

Per questo motivo, generalmente, si tende ad utilizzare principalmente i contesti a corto termine, i quali con maggiore probabilità sono legati ad interessi degli utenti ancora 'vivi', a differenza delle preferenze a lungo termine che possono

[3] vedi sottosezione 1.5.3

portare a perdita di precisione perciò ad un conseguente ranking dei risultati errato.

Ovviamente non tutti i contesti a lungo termine portano ad una perdita di precisione, ma solo quelli che in quella sessione sono irrilevanti o contraddittori con il bisogno d'informazione dell'utente, infatti negli algoritmi di similarità degli scritti analizzati si usano anche i contesti a lungo termine, ma si è riscontrato che sebbene siano utili per aumentare la precisione del sistema, non lo sono tanto quanto i contesti a breve termine che aumentano il livello di precisione in modo molto sostanziale.

Questo tipo di contesto viene utilizzato, negli scritti analizzati, in modo tale da costruire un profilo dell'utente valido per una sessione di ricerca di informazioni.
Il profilo, contenente le preferenze dell'utente rilevanti per quella sessione, viene confrontato con i documenti disponibili tramite una misura di similarità, generalmente viene usato il modello vettoriale[4], integrandolo con la misura di Term Frequency-Inverse Document Frequency(TF-IDF)[5], il quale calcola la similarità del vettore profilo e del vettore documento, tramite la similarità coseno; in seguito viene calcolata anche la similarità della query con i documenti applicando il metodo precedentemente descritto; infine unendo i due risultati e mettendoli in ordine decrescente secondo il punteggio di similarità finale, ne risulta un ranking coerente con i contesti degli utenti considerati utili per quella sessione di Information Retrieval, soddisfacendo efficacemente l'User Information Need iniziale, aumentando così le misure di Precision e Recall[6] del sistema.

1.7 Il progetto AMBIT

Il progetto AMBIT[7] si propone di studiare e sviluppare un'architettura software di supporto ad applicazioni e sistemi dipendenti dal contesto. Lo scopo di tali applicazioni/sistemi è di fornire servizi all'utente che siano personalizzati in base al contesto in cui l'utente si trova ad agire.

Con 'contesto' si intendono un ampio insieme di condizioni, come ad esempio:

[4] vedi sottosezione 1.5.1
[5] vedi sottosezione 1.5.2
[6] vedi sottosezione 1.5.3
[7] Algorithms and Models for Building context-dependent Information delivery Tools

- La locazione geografica
- Il tempo atmosferico
- L'ora del giorno
- Età e il sesso
- Le preferenze dell'utente
- Le attività passate dell'utente
- ecc.

Gli obiettivi che questo progetto si propone dal punto di vita della ricerca, consistono nel contribuire ad un significativo progresso delle conoscenze nel campo dei servizi context-dependent, cioè servizi che si adattano al contesto in cui vengono fruiti, attraverso lo studio e la produzione di risultati innovativi lungo le seguenti direttrici:

- Definizione di **modelli** generali e **tecniche di rappresentazione** di contesti per sistemi e applicazioni in grado di fornire informazioni/servizi dipendenti dal contesto.
- Individuazione di **supporti di memorizzazione** (relational database, graph database, strutture dati ad-hoc,...) e di **algoritmi** atti alla manipolazione efficiente dei contesti (accesso, modifica dinamica, apprendimento,...).
- Disegno di **un'architettura software** generale e di una **piattaforma** prototipale di supporto ad applicazioni client/server dipendenti dal contesto, il cui lato client sia eseguibile anche (e soprattutto) su dispositivi mobili.
- **Verticalizzazione** della piattaforma rispetto ad applicazioni test individuate insieme ai partner operativi.
- **Validazione** dei risultati e loro **diffusione** in ambito scientifico (articoli a congressi, workshop e su riviste specializzate).

Il progetto si propone anche di contribuire ad una maggiore **competitività** dei partner operativi, che è raggiungibile (in special modo in campo informatico) solo

producendo continuamente innovazione. Da questo punto di vista, le ricadute che si attendono dal progetto sono legate sia al miglioramento (in termini di maggiore flessibilità, adattabilità al contesto e personalizzazione) di prodotti che già oggi le aziende partner prevedono 'a catalogo', sia all'individuazione di nuovi prodotti e servizi context-dependent grazie al know-how acquisito.

Il progetto ha interessanti obiettivi di **ricaduta anche a livello territoriale**. Infatti, servizi context-dependent offerti da (per esempio) associazioni di comuni e/o di particolari categorie imprenditoriali (ad esempio, i consorzi per l'Aceto Balsamico Tradizionale di Modena e quello per il Parmigiano Reggiano), possono contribuire a migliorare l'offerta e ad incrementare i volumi di affari, attraverso la segnalazione di punti vendita di interesse, percorsi culturali e gastronomici, fiere, marcati itineranti o periodici, ecc.

Capitolo 2

Progetto del software

In questo capitolo presenterò il progetto che ho realizzato, in particolare spiegherò quali idee e quali scelte ho attuato per realizzare il motore di ricerca, inoltre descriverò quali algoritmi ho sfruttato per far si che il programma funzioni al meglio, senza fornir alcun dettaglio implementativo a livello di codice.

Il contenuto di questo capitolo si divide nelle seguenti sezioni:

- Presentazione generale del progetto (Sezione 2.1).
- Esposizione della creazione dei glossari e degli inverted index, ed inoltre l'uso dei dati estratti dal web server basato su COGITO (Sezione 2.2).
- Esposizione delle funzioni di similarità usate all'interno del programma e della funzione di ranking fusion (Sezione 2.3).

2.1 Presentazione del progetto

Il progetto che ho realizzato in questa tesi di ricerca implementa un motore di ricerca semantico basato sul contesto. Nell'intero progetto ci mettiamo nei panni di un sito di e-commerce, il quale è formato da una moltitudine di pagine web trattanti prodotti specifici, come televisori, libri, cellulari, e videogiochi; tutte le pagine web sono state prese dal sito di e-commerce 'Amazon'[1] e da 'Wikipedia'[2].

[1] www.amazon.com
[2] www.wikipedia.com

Il ruolo di questo sito di e-commerce è quello di consigliare, dei teorici utenti, su quali pagine si avvicinino di più ai loro gusti personali, attraverso un ordinamento delle proprie pagine, anch'esse prese dal sito 'Amazon' e da 'Wikipedia', secondo il grado di similarità con le preferenze di ciascun utente; i gusti di ciascun utente vengono ricavati dal motore di ricerca attraverso la cronologia di navigazione del soggetto che sta navigando il sito di vendita, costruendo un profilo che può essere usato all'interno delle funzioni di similarità e presentando così un ranking coerente con il contesto storico dell'utente in questione.

Il progetto è stato implementato prendendo come base di sviluppo il codice di AMBIT[3], modificato ad-hoc in modo tale da adattarsi agli obiettivi del progetto; inoltre all'interno del progetto vengono usati i dati necessari a caratterizzare una pagina web estratti da un web service basato su COGITO, una tecnologia semantica della ditta Expert System, il quale mi è stato fornito dal Dott. Marcello Pellacani.

Il software mira a paragonare diverse tecniche di ranking, provate al suo interno, per riuscire ad individuare la migliore tra di esse, in particolare si vogliono confrontare i ranking ottenuti per mezzo di diverse tecniche che saranno spiegate nel dettaglio nelle sezioni successive:

- L'uso del modello vettoriale calcolando i pesi dei termini per mezzo della tecnica di **TF-IDF**[4] puro, la quale costituirà la nostra 'baseline'.
- L'uso del modello vettoriale calcolando i pesi dei termini per mezzo della tecnica di **TF-IDF** con anche l'uso dei termini **sinonimi** e dei termini **correlati** di ogni parola all'interno del vocabolario di riferimento.
- La tecnica del **TF-IDF** con l'uso dei soli i termini **sinonimi** di ogni parola all'interno del vocabolario di riferimento.
- Il calcolo della distanza tra le **classi IPTC** estratte dal web service basato su COGITO.

Le prime tre tecniche saranno applicate anche ai dati estratti dal web sevice basato su COGITO, costruendo così ben sette modelli di ranking da confrontare tra di loro e unire per mezzo di una tecnica di ranking fusion implementata all'interno del software.

[3]Vedi sottosezione 1.7
[4]Vedi sottosezioni 1.5.1, 1.5.2

2.2 Glossari, Inverted Index e COGITO

Nella seguente sezione spiegherò quali idee e quali scelte sono state necessarie per costruire i glossari e gli inverted index delle pagine del motore di ricerca e delle pagine del profilo dell'utente, inoltre spiegherò quali dati ho catturato dai dati estratti dal web service basato su COGITO e quale uso ne ho fatto.

2.2.1 Analisi delle pagine web tramite AMBIT

Come prima cosa all'interno del motore di ricerca è stato necessario estrarre i paragrafi dalle pagine web, le quali formano l'insieme di prodotti del sito e-commerce; grazie a questa operazione preliminare è stato possibile, attraverso il software di AMBIT, creare un glossario dei termini presenti in tutte le pagine web, con inoltre l'indicazione del valore di term frequency ed inverse document frequency[5] per esprimere l'importanza di ciascuna parola all'interno delle pagine analizzate.
La medesima operazione, di estrazione del testo e conseguente creazione del glossario, viene fatta anche per le pagine riguardanti il profilo di ogni utente, così da poter usare queste informazioni all'interno di una futura funzione di similarità con i paragrafi delle pagine web.

2.2.2 Costruzione dell'inverted index

Per mezzo dell'uso del glossario, dalle parole estratte da ogni paragrafo delle pagine web, è stato possibile costruire un'inverted index, indicante la parola estrapolata dal glossario ed il rifermento ai paragrafi delle pagine web in cui appare quella parola; inoltre in questa fase vengono estratti anche i termini che sono collegati alle parole all'interno del glossario da legami di sinonimia, usando i 'synset' della libreria 'WordNet', e da legami di correlazione, navigando l'albero degli iponimi e degli iperonimi di 'WordNet'.

La scelta del numero di parole sinonime e correlate da inserire nella collezione è stata una decisione cruciale per il buon raggiungimento del miglior ranking possibile, in quanto una selezione troppo vasta di parole avrebbe portato ad un ranking errato, mentre un numero esiguo di termini avrebbe comportato un

[5] Vedi sottosezione 1.5.2

ranking troppo vicino a quello effettuato senza l'uso di termini sinonimi e/o correlati.
Per questo motivo è stato necessario imporre dei limiti sulla estrazione di sinonimi e termini correlati, in particolare:

- Nel caso di parole sinonime si è deciso di prendere tutti i sinonimi di una determinata parola all'interno del glossario, che abbiano al limite un solo 'synset' fornito da 'WordNet', in questo modo i sinonimi saranno per forza di cose quelli corretti;
- Per i termini correlati si è deciso di prendere le parole, di un determinato termine all'interno del glossario, che sono al più ad una distanza minore o uguale a due sull'albero degli iponimi e degli iperonimi di 'WordNet', dalla parola in questione.

Grazie a questi due tipi di filtri è stato possibile selezionare un numero di termini correlati e sinonimi adeguati, al fine di avere un ranking più preciso rispetto il ranking ottenuto dalla tecnica di TF-IDF puro, la quale non fa uso di sinonimi e termini correlati.

2.2.3 Uso del web service basato su COGITO

Grazie all'utilizzo del web service basato su COGITO, è stato possibile estrarre dalle pagine web gli elementi fondamentali per la categorizzazione di ciascuna pagina web. Andando più nel particolare il web service si occupa di analizzare una pagina web estrapolando da essa una moltitudine di informazioni utili, delle quali nel progetto ne ho utilizzate quattro :

- **Classi IPTC con score.** Si trattano di categorie standard con una struttura ad albero messe a punto dall' International Press Telecommunication Council (IPTC), usato da tutte le principali agenzie di stampa internazionali per la trasmissione e la categorizzazione di contenuti editoriali; le quali COGITO utilizza per assegnare con precisione a ogni tipo di testo una o più categorie con un relativo score di importanza, scegliendo quelle più pertinenti fra le 1200 categorie IPTC; per esempio, «('IPTC/Arte, cultura, intrattenimento/Televisione', '980'), ('IPTC/Economia, affari e finanza/Informatica e Telecomunicazioni/Servizi Telecomunicazione', '116'), ('IPTC/Arte, cultura, intrattenimento/Musica', '60')».

- **Entities.** Sono una serie delle parole rilevanti estratte dal web service che identificano particolari entità, quali: posti, organizzazioni, concetti conosciuti da COGITO, concetti generici, e persone.
- **Domains.** Sono una serie delle parole rilevanti estratte dal web service che identificano dei domini di appartenenza della pagina web.
- **Mainlemmas/Relevants.** Sono una serie delle parole rilevanti estratte dal web service che identificano le parole più ricorrenti ed importanti dentro la pagina web.

Per mezzo di Entities, Domains e Mainlemmas mi è stato possibile costruire un glossario con queste parole nel medesimo modo descritto nella sottosezione 2.2.1 tramite l'uso del codice di AMBIT, è bastato solamente modificare alcuni parametri in quanto il web service, a differenza di AMBIT, non lavora sui paragrafi ma bensì ricava le informazioni da intere pagine web.
La medesima operazione, di estrazione di Entities, Domains e Mainlemmas e conseguente creazione del glossario, viene fatta anche per le pagine riguardanti il profilo di ogni utente, così da poter usare queste informazioni all'interno di una futura funzione di similarità con le pagine web.
In seguito per mezzo dell'uso del glossario creato viene costruito un inverted index nello stesso modo in cui è stato spiegato nella sottosezione 2.2.2, dove però ogni termine non si riferisce più ad un solo paragrafo di una pagine ma alla pagina web intera, in quanto il web sevice basato su COGITO lavora su pagine web intere.

2.3 Funzioni di similarità e ranking fusion

Nella seguente sezione esporrò le diverse funzioni di similarità che ho implementato e usato, descrivendone la logica e le scelte effettuate per costruirle. Inoltre esporrò l'algoritmo di ranking fusion utilizzato e i motivi che mi hanno portato a tale scelta.

2.3.1 Similarità con modello vettoriale esteso

L'algoritmo originale di similarità con modello vettoriale esteso usato da AMBIT confronta una o più parole, affiancate dal relativo peso, con i paragrafi delle

pagine web del sito e-commerce; costruendo un array contenente le parole del profilo estratte dal glossario creato precedentemente[6], ed assegnando a ciascuna parola il peso calcolato attraverso la misura di Term Frequency moltiplicata per la misura di Inverse Document Frequency del termine, contenuti anch'essi all'interno del glossario, è stato possibile sfruttare il modello vettoriale usato da AMBIT, calcolando così la similarità che il profilo dell'utente ha con i paragrafi delle pagine del sito e-commerce.

La similarità con modello vettoriale esteso deriva dall'algoritmo di similarità descritto nell'articolo di Bergamaschi et al.[3], nel quale si definisce la formula di similarità tra documenti $DSim = (D^x, D^y)$ dove, dato un documento sorgente $D^x = (t_1^x, ..., t_n^x)$ ed un documento finale $D^y = (t_1^y, ..., t_n^y)$, si quantifica la similarità del documento sorgente con il documento finale tramite la somma di tutti i termini di somiglianza tra ogni termine in D^x e ogni termine di D^y, ed ogni termine in D^y che massimizza la similarità con il termine in D^x:

$$DSim(D^i, D^j) = \sum_{t_i^x \in D^x} TSim(t_i^x, t_{\bar{j}(i)}^y) \cdot w_i^x \cdot w_{\bar{j}(i)}^y$$

$$t_{\bar{j}(i)}^y = argmax_{t_j^y \in D^y}(TSim(t_i^x, t_j^y)$$

dove $w_i^x = tf_i^x \cdot idf_i$ e $w_{\bar{j}(i)}^y = tf_{\bar{j}(i)}^y \cdot idf_{\bar{j}(i)}$. In questo modo tutti i termini contribuiscono alla similarità finale con un perso differente.
I termini di somiglianza possono essere:

- Termini uguali;
- Termini sinonimi;
- Termini correlati;

definiti tramite la funzione $TSim(t_i, t_j)$:

$$TSim(t_i, t_j) = \begin{cases} 1 & \text{se } t_i = t_j \text{ oppure } t_i \text{ SYN } t_j \\ r & \text{se } t_i \text{ REL } t_j \\ 0 & \text{altrimenti} \end{cases}$$

Perciò se un termine è uguale ad un altro o è all'interno della lista dei termini simili avrà un peso uguale a uno, se si tratta di un termine all'interno della lista

[6]Vedi sottosezione 2.2.1

dei termini correlati prenderà il valore r, altrimenti gli si darà il valore zero.

All'interno del software è a discrezione del programmatore se utilizzare i termini sinonimi e/o i termini correlati, entrambi memorizzati all'interno dell'inverted index[7]; grazie a questa scelta mi è stato possibile sfruttare lo stesso metodo per poter costruire tre funzioni di similarità differenti:

1. Funzione di similarità la quale usa solo i valori di **Term Frequency e Inverse Document Frequency** di ogni termine per calcolare lo score di ogni paragrafo, questa costituirà la **baseline** su cui confrontare le altre funzioni di similarità.

2. Funzione di similarità la quale utilizza oltre ai valori di Term Frequency e Inverse Document Frequency di ogni termine, anche i pesi di eventuali **parole sinonime e correlate**.

3. Funzione di similarità la quale utilizza oltre ai valori di Term Frequency e Inverse Document Frequency di ogni termine anche i pesi di eventuali **parole sinonime**.

Le stesse funzioni di similarità sopra descritte usate da AMBIT, vengono utilizzate per costruire tre varianti che sfruttano, non più i termini estratti dalle pagine web del profilo per mezzo dei metodi di AMBIT, ma bensì, i dati estratti dal web service basati su COGITO, come è stato descritto nella sottosezione 2.2.3, le quali confrontano i termini del profilo dell'utente, composti da Entities, Domains e Mainlemmas, con i termini delle intere pagine web, composti anch'essi da Entities, Domains e Mainlemmas; in modo tale da costruire tre nuovi ranking delle pagine del sito e-commerce basandosi esclusivamente sui dati estratti da COGITO.

2.3.2 Similarità attraverso le classi IPTC

Un'ulteriore funzione di similarità l'ho costruita per mezzo delle classi IPTC e relativi score, estratti tramite il web service basato su COGITO.
Per costruirla ho seguito una serie di passi:

[7] Vedi sottosezione 2.2.2

1. Per prima cosa, è stato necessario estrarre, dai dati forniti da COGITO[8], le classi IPTC con i relativi score delle pagine web che formano il profilo dell'utente, e le classi IPTC con i relativi score delle pagine web del sito e-commerce.

2. In seguito, ho unito le classi IPTC uguali tra tutte le pagine che formano il profilo dell'utente, sommando tra di loro gli score di ciascuna classe IPTC; così da non avere classi duplicate e dare uno score più elevato a classi maggiormente presenti tra quelle estratte dalle pagine del profilo dell'utente.

3. A partire dalla funzione di similarità esposta da Leacock et al.[13]:

$$sim(t_x, t_y) = \begin{cases} -ln\frac{len(t_x,t_y)}{2 \cdot H} & \text{se } \exists \text{ un antenato comune} \\ 0 & \text{altrimenti} \end{cases}$$

dove $len(t_x, t_y)$ è il minimo numero di connessioni colleganti ogni senso dentro $Senses(t_x)$ con ogni senso dentro $Senses(t_y)$ e H è l'altezza della gerarchia degli iperonimi di WordNet; ho infine costruito la funzione di similarità sostituendo a $len(t_x, t_y)$ la distanza $D(P_i, E_{kj})$, calcolata come la differenza di cammino che c'è sull'albero delle classi IPTC tra ogni classe i del profilo P, ricavate al punto 2, con ogni classe j di ciascuna delle k pagine del sito e-commerce E:

$$sim(P_i, E_{kj}) = -log(\frac{D(P_i, E_{kj})}{2 \cdot H})$$

dove H indica l'altezza dell'albero delle classi IPTC (la quale corrisponde a 5).

Nel caso in cui la classe P_i sia uguale alla classe E_{kj} moltiplico la similarità $sim(P_i, E_{kj})$ per lo score s del profilo P_i, aumentando così la similarità nel caso in cui la pagina del sito e-commerce e le pagine del profilo condividano una o più classi uguali, in quanto lo score s in generale è un valore molto maggiore di 1:

$$sim(P_i, E_{kj}) = -log(\frac{D(P_i, E_{kj})}{2 \cdot H}) \cdot s(P_i)$$

[8]Vedi sottosezione 2.2.3

Le funzioni di similarità citate sopra vengono sommate tra di loro per costruire una misura di similarità finale per ogni pagina all'interno del sito e-commerce:

$$sim(P_i, E_k) = \sum_{j=1}^{n} \begin{cases} -log(\frac{D(P_i,E_{kj})}{2 \cdot H}) \cdot s(P_i) & \text{se } P_i = E_{kj} \\ -log(\frac{D(P_i,E_{kj})}{2 \cdot H}) & \text{se } P_i \neq E_{kj} \end{cases}$$

2.3.3 Ranking Fusion

La necessità di un algoritmo di ranking fusion nasce dal bisogno di dover unire i benefici dell'algoritmo di similarità che usa le classi IPTC descritto nella sottosezione 2.3.2, il quale cattura delle informazioni di natura **generale** riguardo la pagine analizzate (per esempio se la pagina tratta di televisori o musica), con gli algoritmi di similarità che usano il modello vettoriale esteso, descritti nella sottosezione 2.3.1, i quali catturano informazioni legate ai singoli termini contenuti nelle pagine analizzate trovando informazioni più **specifiche** rispetto la similarità con le classi IPTC (per esempio marche di televisori come 'Sony' o 'Samsung').

In principio, per il ranking effettuato dall'algoritmo di similarità con modello vettoriale esteso che utilizza i termini estratti dai metodi di AMBIT[9], è stato necessario sommare tra di loro i punteggi di ciascun paragrafo per ogni pagina del sito di e-commerce così da ottenere un ranking delle pagine e non più dei paragrafi, in linea con il ranking effettuato tramite le classi IPTC.

In seguito dato che i punteggi di similarità erano espressi tutti in scale di valori diversi, si è reso necessario normalizzare i valori di ciascun ranking ottenuto, sommando tutti i punteggi all'interno di ciascun ranking e dividendo questo valore per ogni singolo valore all'interno del ranking, così da ottenere tutti dei valori che variano dallo 0 all'1.

Infine basandomi sull'**algoritmo pesato KE** descritto nell'articolo di Akritidis et al.[2] come:

$$W_{Wke} = \frac{\sum_{i=1}^{m}[11 - e(i)]r(i)}{n^m(\frac{k}{10} + 1)^n}$$

dove $\sum_{i=1}^{m}[11 - e(i)]r(i)$ è la somma di tutti i punteggi che l'oggetto ha preso, n è il numero di ranking che includono l'oggetto, m è il numero totale di ranking

[9] Vedi sottosezione 2.3.1

coinvolti nel ranking fusion, k è il numero totale degli oggetti all'interno dei ranking che l'algoritmo usa, e $e(i)$ è il fattore di peso dell'i-esimo ranking che, in questo caso, può ricevere un valore compreso tra l'1 e il 10; ho adattato questo algoritmo nel seguente modo :

$$W_{Wke} = \frac{\sum_{i=1}^{m}[(len(i)+1) - e(i)]r(i)}{n^m(\frac{k}{max(len(r))}+1)^n}$$

dove $len(i)$ indica la lunghezza dell'i-esimo ranking, $e(i)$ indica la posizione dell'oggetto all'interno del ranking, in questo modo si da più peso ad elementi più in alto nel ranking e viceversa, e $max(len(r))$ indica la lunghezza massima tra i ranking che si vogliono fondere.

Capitolo 3

Implementazione del software

In questo capitolo spiegherò come le scelte descritte nel capitolo 2 sono state usate all'interno del codice, fornendo un commento completo sui dettagli implementativi di sezioni significative del codice realizzato.
L'intero progetto è stato realizzato nel linguaggio di programmazione **Python**, facile da comprendere e molto portabile, così da poter facilitare eventuali modifiche future.
La logica di esposizione delle varie sezioni rispecchia il flusso di codice del modulo **ambit.py**[1], il quale gestisce ed esegue tutte le operazioni descritte all'interno di questo capitolo nello stesso ordine in cui sono presentate.
Il contenuto di questo capitolo si divide nelle seguenti sezioni:

- Estrazioni del testo (Sezione 3.1)
- Realizzazione dei glossari di riferimento (Sezione 3.2)
- Realizzazione degli Inverted Index (Sezione 3.3)
- Realizzazione delle funzioni di similarità e implementazione dell'algoritmo di ranking fusion (Sezione 3.4)

3.1 Estrazioni del testo

In questa sezione spiegherò come i dati utili per costruire i glossari sono stati estratti, dividendo le metodologie di estrazione a seconda che i dati siano presi

[1] Vedi appendice A.1

da normali pagine web o dalle pagine XML restituite dal web service basato su COGITO.

3.1.1 Estrazione dei dati da pagine web

L'estrazione del testo dagli URL delle pagine web in HTML, sia del profilo degli utenti che del sito e-commerce, è affidato al modulo **extractText.py**[2], il quale si preoccupa di applicare alla pagina web da cui estrarre i paragrafi, alcune espressioni regolari implementate all'interno del metodo **extractText.extractText** che si occupano di:

- Eliminare gli 'a capo'

```
data = data.replace("\n", " ")
data = data.replace("\r", " ")
```

- Estrarre il body della pagina HTML

```
bodyPat = re.compile(r'<body[^<>]*?>(.*?)</body>', re.I)
result = re.findall(bodyPat, data)
data = result[0]
```

- Eliminare i JavaScript

```
p = re.compile(r'<script[^<>]*?>.*?</script>')
data = p.sub('', data)
```

- Eliminare gli stili CSS

```
p = re.compile(r'<style[^<>]*?>.*?</style>')
data = p.sub('', data)
```

- Eliminare i commenti

```
p = re.compile(r'<!--(.)*?-->')
data = p.sub('', data)
```

- Estrarre i paragrafi

```
parPat = re.compile(r'<(p|div)[^<>]*?>(.*?)</(p|div)>', re.I)
result = re.findall(parPat, data)
```

[2]Vedi appendice A.2

In seguito per ciascun paragrafo raccolto si memorizza in una variabile **pHTMLLen** la lunghezza attuale del paragrafo, poi si eliminano tutti i tag, gli spazi consecutivi e gli unescape, traducendo ciò che ne rimane in 'ASCII'; fatto ciò si calcola il rapporto tra la lunghezza del paragrafo ottenuto **pTextLen** e **pHTMLLen**, ottenendo un valore variabile tra 0 e 1 indicante quanto il paragrafo è cambiato dopo aver eliminato gli elementi sopra citati, perciò se il rapporto ottenuto ha un valore vicino allo 0 significa che il paragrafo molto probabilmente era formato solo da testo non rilevante (come tag, spazi consecutivi, o unescape); infine si utilizza il rapporto precedente per eliminare i paragrafi che hanno un valore inferiore a 0.2 (indicata dalla costante MIN_TEXT_RATIO) ed una lunghezza minore di 150 caratteri(indicata dalla costante MIN_PAR_LEN), così da togliere quei paragrafi formati da pochi caratteri.

```
1      if pTextLen>MIN_PAR_LEN and pTextRatio > MIN_TEXT_RATIO:
2        paragrafi.append((docId+"-"+str(count),par))
3        count = count+1
```

Queste operazioni di estrazione vengono applicate ad ogni URL delle pagine web, contenuti nella lista **docsList**, passata dal metodo **extractText.multiExtractText**, il quale infine memorizza i paragrafi di ciascuna pagina HTML in un file di testo **docsFile** che in seguito sarà utilizzato in altri moduli del programma.

```
1  def multiExtractText(docsList,docsFile):
2    pars = []
3    for doc in docsList:
4      docId = doc[0]
5      url = doc[1]
6      par = extractText(docId, url)
7      pars.extend(par)
8    outF = open(docsFile,'w')
9    for p in pars:
10     if p!=pars[0]:
11       outF.write('\n')
12     outF.write(p[0]+"\t"+p[1])
13   outF.close()
14   print "Done!"
15   return pars
```

3.1.2 Estrazione dei dati dalle pagine del web service

L'estrazione dei dati rilevanti all'interno della pagina XML, restituita dal web service basato su COGITO, fatta per ogni URL, sia del profilo degli utenti che

del sito e-commerce, è affidato al modulo **cogito.py**[3] il quale, per mezzo del metodo analyzeText, passa l'URL della pagina web in questione al web service basato su COGITO

```
def analyzeText(url):
  print "Analysing "+url+"..."
  urlS = "http://217.26.90.209/ESDemo/Analyzer"
  query_args = { 'customerKey':'unimore', 'url':url }
  data = urllib.urlencode(query_args)
  request = urllib2.Request(urlS, data)
  response = urllib2.urlopen(request)
  xml = response.read()
  d = extractDomains(xml)
  e = extractEntities(xml)
  r = extractRelevants(xml)
  return (d,e,r)
```

grazie al quale genera una pagina XML dalla quale estrae le informazioni descritte nella sottosezione 2.2.3:

- Domains e classi IPTC

```
def extractDomains(data):
  ris = []
  pat = re.compile(r'<DOMAIN[^<>]*?NAME="([^"]*?)" SCORE="([^"]*?)"/>', re
    .I)
  result = re.findall(pat, data)
  for r in result:
    ris.append((r[0],r[1]))
  return ris
```

- Entities

```
def extractEntities(data):
  ris = []
  lastType = ""
  pat = re.compile(r'<ENTITY NAME="([^"]*?)">', re.I)
  patType = re.compile(r'<ENTITIES TYPE="([^"]*?)">', re.I)
  for line in data.splitlines():
    result = re.findall(pat, line)
    if len(result)>0:
      ris.append((result[0],lastType))
    else:
      result = re.findall(patType, line)
      if len(result)>0:
        lastType = result[0].lower()
  return ris
```

- Relevants

[3] Vedi appendice A.3

```
def extractRelevants(data):
  ris = []
  lastType = ""
  pat = re.compile(r'<RELEVANT NAME="([^"]*?)"[^<>]*?(RANKING="([^"]*?)")
    ?[^<>]*?SCORE="([^"]*?)"/?>', re.I)
  patType = re.compile(r'<RELEVANTS TYPE="([^"]*?)">', re.I)
  for line in data.splitlines():
    result = re.findall(pat, line)
    if len(result)>0:
      ris.append((result[0][0],result[0][3],lastType))
    else:
      result = re.findall(patType, line)
      if len(result)>0:
        lastType = result[0].lower()
  return ris
```

Queste operazioni di estrazione vengono applicate ad ogni URL delle pagine web, contenuti nella lista **docsList**, passata al metodo **cogito.multiAnalyzeText**, il quale infine memorizza i dati estratti da ciascuna pagina XML in un file 'outFile.dat' creato per mezzo della libreria 'pickle' che in seguito sarà utilizzato in altri moduli del programma.

```
def multiAnalyzeText(docsList,outFile):
  analyses = []
  for doc in docsList:
    docId = doc[0]
    url = doc[1]
    an = analyzeText(url)
    analyses.append((docId,an))
  f_out = open(outFile,"w")
  pickle.dump(analyses,f_out)
  f_out.close()
  print "Pickling OK!"
  return analyses
```

3.2 Glossari

In questa sezione spiegherò come i dati estratti nella sezione 3.1 vengono utilizzati per costruire i glossari che registrano i termini rilevanti di ciascuna pagina o paragrafo, con anche l'indicazione per ciascun termine delle misure di Term Frequency e Inverse Document Frequency, facendo una distinzione tra l'estrazione dei termini rilevanti dal testo delle pagine web HTML[4], e l'estrazione dei

[4]Vedi sottosezione 3.1.1

termini rilevanti ricavati dalla pagina XML restituita dal web service basato su COGITO[5].
I metodi, le variabili e le costanti, citate all'interno di questa sezione, sono tutte contenute all'interno del modulo **glossaryExtractor.py**[6].

3.2.1 Estrazione dei termini rilevanti dai paragrafi delle pagine web HTML

L'estrazione dei termini rilevanti dai paragrafi delle pagine web HTML è gestita dal metodo **glossaryExtractor.extractTermData** il quale, per mezzo della libreria **Topia**, estrapola per ciascuna linea di ogni paragrafo i termini rilevanti, filtrando i caratteri non voluti (come parentesi e simboli), inoltre estrae anche l'indicazione di quante volte ricorrono tali termini all'interno quella linea(variabile term[1]) e l'indicazione del codice riguardante il documento a cui appartiene (variabile **qmCode**).

```
1  for line in f:                          # for each doc: extract term data
2    curDocLen = 0
3    line = line.strip()
4    qmCode = line[:line.find("\t")]        # qm code
5    qmText = line[(line.find("\t")+1):].lower()   # qm text description (lower
     case)
6    #print extractor.tagger(qmText)         # POS-tagged terms
7    for term in sorted(extractor(qmText)):        # extracted terminology
8      if ( "." in term[0] or "(" in term[0] or ")" in term[0] or "+" in term[0]
       # filter out unwanted terms
9                          or "\xa0" in term[0] or "\x80" in term[0] or "/" in
     term[0]
10                         or ":" in term[0] or term[0].isdigit() or len(term
     [0])<3):
11         continue
12     docTerm = (term[0].lower(), qmCode, term[1], term[2])   # term, code, freq
     , strength
13     docTerms.append(docTerm)
14     curDocLen = curDocLen+1
15     #print docTerm
16   docLengths.append((qmCode,curDocLen))
17   docTerms.append(("zzzzz","zzzzz",-1,-1))   # last docTerm (terminator)
```

Infine il metodo restituisce: l'array di tuple contenente tutte le informazione descritte precedentemente dove alla fine delle tuple di ciascun paragrafo viene posta una tupla di fine (**docTerms.append((zzzzz,zzzzz,-1))**), e un array

[5]Vedi sottosezione 3.1.2
[6]Vedi appendice A.4

di tuple formate da il codice del paragrafo (**qmCode**) e il numero di termini rilevanti estratti per quel paragrafo (**curDocLen**).

Una nota di riguardo è va fatta riguardo l'uso del filtro della libreria **Topia**, il quale permette di prendere solo i termini rilevanti che all'interno del testo, o in questo caso nel paragrafo, ricorrono più di n volte (tale valore n è definito dal programmatore); ebbene all'interno del metodo si utilizza il 'permissiveFilter' il quale non da alcuna restrizione sul numero di ricorrenze dei termini estratti, in quanto è stato testato che tale filtro escluderebbe troppi termini rilevanti e la dimensione del glossario finale sarebbe troppo esigua.

```
extractor = extract.TermExtractor()
extractor.filter = extract.permissiveFilter
```

3.2.2 Estrazione dei termini rilevanti ricavati dalla pagina XML

I termini che si vogliono estrarre dai termini rilevanti ricavati dalla pagina XML corrispondono ai Relevants, Domains e Mainlemmas, tale estrazione è operata dal metodo **glossaryExtractor.extractCogitoData** il quale, dopo aver caricato i dati dal file **cogFile** per mezzo della libreria **pickle**

```
print "Reading cogito data..."
docTerms=[]
docLenghts=[]
pickle_file = open(cogFile)
cogito_file = pickle.load(pickle_file)
pickle_file.close()
```

estrae per ciascun documento, i termini Relevants, Domains e Mainlemmas, memorizzando il numero di volte che quel termine occorre all'interno del documento e il codice del documento **docId[0]** all'interno dell'array **docTerms**.

```
for docId in cogito_file:
  i=0
  currDocLen=0
  for ent_rel in docId[1]:
    if (i==0) :
      i+=1
      continue
    for currWord in ent_rel:
      currDocLen+=1
      if (currDocLen==1):
        docTerms.append((currWord[0].lower(),docId[0],1))
```

```
12          else:
13            count=0
14            isInside=False
15            for pastTerm in docTerms:
16              if (pastTerm[1]== docId[0]):
17                if (currWord[0].lower()==pastTerm[0]):
18                  docTerms[count] = list(docTerms[count])
19                  docTerms[count][2]+=1
20                  docTerms[count] = tuple(docTerms[count])
21                  isInside=True
22              count+=1
23            if (isInside == False):
24                docTerms.append((currWord[0].lower(),docId[0],1))
25        i+=1
26      docLenghts.append((docId[0],currDocLen))
27      docTerms.append(("zzzzz","zzzzz",-1))
28    numDocs = len(cogito_file)
```

Infine il metodo restituisce: l'array **docTerms** di tuple contenente tutte le informazione descritte precedentemente dove alla fine delle tuple di ciascun documento viene posta una tupla di fine (**zzzzz,zzzzz,-1**), l'array di tuple docLenghts formate da il codice del documento (**qmCode**) e il numero di termini rilevanti estratti per quel documento (**curDocLen**), e la variabile **numDocs** indicante il numero di pagine da cui sono stati estratti i termini rilevanti.

3.2.3 Generazione del glossario

Le seguenti operazioni di generazione del glossario vengono effettuate sia per le pagine riguardanti il profilo dell'utente che per le pagine del sito di e-commerce. La costruzione del glossario è affidata al metodo **glossaryExtractor.go** il quale, oltre ad estrarre i termini rilevanti, distinguendo il caso in cui si voglia estrarre il testo come descritto nella sottosezione 3.2.1 oppure come descritto nella sottosezione 3.2.2 impostando la variabile booleana **USE_COGITO** rispettivamente falsa o vera,

```
1  def go(inputFile,outFile,USE_COGITO):
2    if (USE_COGITO) :
3      print 'with cogito...'
4      docTerms,docLengths,numDocs=extractCogitoData(inputFile)
5    else:
6      numPars, numDocs = countNumberOfDocuments(inputFile)
7      print "Total number of documents: "+str(numDocs)
8      print "Total number of paragraphs: "+str(numPars)
9      print "Extracting glossary data..."
10     print 'without cogito...'
```

```
11      docTerms, docLengths = extractTermData(inputFile)
```

si occupa di calcolare i valori di Term Frequency e Inverse Document Frequency dei termini estratti nella fase precedente.

Il calcolo dell'Inverse Document frequency è affidato al metodo **glossaryExtractor.generateGlossary** il quale calcola per ogni termine il logaritmo del rapporto tra il numero di documenti **numDocs** e la Document Frequency del termine **curDocList**; infine il metodo restituisce il valore **termCount** il quale indica il numero di termini del glossario e un dizionario **termIDFs** contenente come chiavi i termini del glossario e come argomenti i rispettivi valori di Inverse Document Frequency.

```
1  def generateGlossary(docTerms,numDocs,USE_COGITO):          # glossary
       generation
2    curTerm = ""     # current term
3    termIDFs = {}     # term, idf
4    termCount = 0     # number of terms in glossary
5    for docTerm in sorted(docTerms):
6      if (docTerm[0]!=curTerm):     # new term
7        if curTerm!="":
8          curFreq = len(curDocList)
9          curIDF = math.log(numDocs/curFreq)
10         termIDFs[curTerm] = curIDF
11         termCount +=1
12       curTerm=docTerm[0]
13       #curStren=docTerm[3]
14       curDocList = []
15     if (USE_COGITO):
16       curDoc = docTerm[1]
17     else:
18       curDoc = docTerm[1].split('-')[0]     # extract doc id from paragraph id
19     if curDoc not in curDocList:
20       curDocList.append(curDoc)
21   return termCount, termIDFs
```

Il calcolo del valore di Term Frequency è affidato al metodo **glossaryExtractor.generateReport** il quale calcola il rapporto tra il numero di occorrenze di un determinato termine all'intero di un documento **curFreq**, calcolato nella fase di estrazione dei termini rilevanti, diviso la lunghezza del documento **curDocLen**; restituendo infine un'array di tuple formate da: il codice del documento **curDoc**, il termine **curTerm** e il valore di Term Frequency relativo al termine **curTF**.

```
1  def generateReport(docTerms, docLengths, termIDFs):                  # report
       generation
2    termTFs = []     # doc-code, term, tf
3    for doc in sorted(docLengths):          # for each document
```

```
    curDoc = doc[0]
    curDocLen = doc[1]
    for docTerm in docTerms:
      if (docTerm[1]==curDoc):     # relevant term
        curTerm = docTerm[0]     # term
        curFreq = docTerm[2]     # current term frequency (in current doc)
        if curDocLen==0:
          continue
        curTF = float(curFreq)/curDocLen      # current term TF
        curOutData = (curDoc, curTerm, curTF)
        termTFs.append(curOutData)
  return termTFs
```

Alla fine di tutte queste operazioni il metodo **glossaryExtractor.go** salva sul file **outFile** l'array in cui è contenuto il valore di Term Frequency di ogni termine del glossario e il dizionario in cui è contenuto, per ciascun termine all'interno del glossario, il valore di Inverse Document Frequency.

```
f_out = open(outFile,"w")
glossaryData = (termTFs,termIDFs)     # list:  doc-code,term,tf (array), term:
  idf (dict)
# print termTFs
# print termIDFs
pickle.dump(glossaryData,f_out)
f_out.close()
print "Pickling OK!"
```

3.3 Inverted Index

L' Inverted Index è stato creato sfruttando il glossario generato nella sezione 3.2 (letto tramite il metodo **similarityComp.readGlossary**) il quale è usato dal metodo **similarityComp.genIndexFile** per creare la struttura dati finale, memorizzata su file per mezzo della libreria **pickle**, contenete quattro dizionari:

- Il dizionario delle occorrenze, ottenuto per mezzo del metodo **similarityComp.genInvertedIndex**, il quale contiene per ogni termine nel glossario, la lista dei documenti in cui quel termine occorre.

```
def genInvertedIndex(termTFs):
  invIndex = {}
  for t in termTFs:
    doc = t[0]
    term = t[1]
    if (term in invIndex):
      invIndex[term].append(doc)
```

```
8       else:
9         list = []
10        list.append(doc)
11        invIndex[term]=list
12    print invIndex.keys()
13    print invIndex.values()
14    return invIndex
```

- Il dizionario dei Term Frequency, ottenuto per mezzo del metodo **similarityComp.genDocTerms**, il quale contiene per ciascun documento una lista di tuple, formate dal termine appartenente al documento e il rispettivo valore di Term Frequency.

```
1  def genDocTerms(termTFs):
2    docTerms = {}
3    for t in termTFs:
4      doc = t[0]
5      term = t[1]
6      tf = t[2]
7      if (doc in docTerms):
8        docTerms[doc].append((term,tf))
9      else:
10       list = []
11       list.append((term,tf))
12       docTerms[doc]=list
13   return docTerms
```

- Il dizionario dei termini sinonimi, ottenuto per mezzo del metodo **similarityComp.genAllWnSyns**,

```
1  def genAllWnSyns(termIDFs):
2    synTerms = {}
3    for term1 in sorted(iter(termIDFs)):
4      print ("TERM: "+term1)
5      terms = getWnSyns(term1)
6      print (terms)
7      synTerms[term1]=terms
8    return synTerms
```

il quale contiene per ogni termine la sua lista di termini sinonimi, scelti in base ai criteri spiegati nella sottosezione 2.2.2 i quali sono stati implementati all'interno del metodo **similarityComp.getWnSyns**.

```
1  def getWnSyns(term):
2    syns = []
3    for s in wn.synsets(term):
4      for lemma in s.lemmas:
5        if (len(wn.synsets(lemma.name.replace('_',' ')))==1):
6          syn = lemma.name.replace('_',' ')
```

```
7           syns.append(syn)
8     syns = sorted(list(set(syns)))
9     #print syns
10    return syns
```

- il dizionario dei termini correlati, ottenuto per mezzo del metodo **similarityComp.genAllWnRelated**,

```
def genAllWnRelated(termIDFs):
  relTerms = {}
  for term1 in sorted(iter(termIDFs)):
    print ("TERM: "+term1)
    terms = getWnRel(term1)
    print terms
    relTerms[term1]=terms
  return relTerms
```

il quale contiene per ogni termine la lista di termini correlati, scelti in base ai criteri spiegati nella sottosezione 2.2.2 i quali sono stati implementati all'interno del metodo **similarityComp.getWnRel**.

```
def getWnRel(term):
  rel = []
  for s in wn.synsets(term):
    for hype in s.hypernym_distances():
      he = hype[0].name.split('.')[0].replace('_',' ')
      if ((hype[1]<=2) and (he not in rel)and(he!=term)):
        if (len(wn.synsets(he))==1):
          #print he
          rel.append(he)
    for hypo in s.hyponyms():
      dist = 0
      for ho in hypo.hypernym_distances():
        if s == ho[0]:
          dist = ho[1]
      for ho in hypo.hypernym_distances():
        h = ho[0].name.split('.')[0].replace('_',' ')
        if ((ho[1]<= dist)and((dist-ho[1])<=2)and(h not in rel)and(h!=term
    )):
          if (len(wn.synsets(h))==1):
            #print h
            rel.append(h)
  rel = sorted(list(set(rel)))
  #print rel
  return rel
```

3.4 Funzioni di similarità e ranking fusion

In questa sezione esporrò le due funzioni di similarità implementate nel modulo **similarityComp.py**[7] di cui se ne è già discussa la logica di funzionamento nella sezione 2.3; infine esporrò come queste funzioni sono state combinate tramite una tecnica di ranking fusion, mostrando inoltre i passi preliminari che si sono resi necessari per avere un ranking finale coerente.

3.4.1 Similarità con modello vettoriale esteso

Per prima cosa, come passo preliminare al calcolo della similarità, è stato necessario creare il vettore del profilo, contenente tutte le parole del glossario del profilo e per ciascuna il valore di Term Frequency moltiplicato per Il valore di Inverse Document Frequency per dare a ciascuna parola un determinato peso.

```
prof_tfs1 ,prof_idfs1 = similarityComp.readGlossary(PROF3_GLOSSARY)
Query1 = [] #Costruisco un vettore con le parole contenute dentro il glossario
    del profilo per poterle usare nella funzione similarityComp.executeQuery
for prof_tf in prof_tfs1:
  if prof_tf[1] in prof_idfs1.keys():
    Query1.append((prof_tf[1],prof_tf[2]*prof_idfs1[prof_tf[1]]))
  else:
    Query1.append((prof_tf[1],prof_tf[2]))
```

La similarità con modello vettoriale esteso, descritta nel dettaglio nella sottosezione 2.3.1, è modellata per mezzo del metodo **similarityComp.executeQuery** il quale permette, in fase di chiamata, di impostare a TRUE o FALSE le variabili booleane globali **USE_SYNS** e **USE_REL**, indicanti rispettivamente se usare o meno i termini sinonimi nel calcolo della similarità, e se includere o meno i termini correlati nel calcolo di similarità.

```
def executeQuery(queryTerms,glossaryFile,indexFile,USE_SYNS_NEW,USE_REL_NEW):
  global USE_REL
  USE_REL=USE_REL_NEW
  global USE_SYNS
  USE_SYNS=USE_SYNS_NEW
  termTFs, termIDFs = readGlossary(glossaryFile)
  invIndex, docTerms, synTerms, relTerms = readIndex(indexFile)
  syns = {}
  ranking = ambitSimilarityV2(queryTerms,termTFs,termIDFs,synTerms,relTerms,
    invIndex,docTerms)
  return ranking
```

[7] Vedi appendice A.5

Grazie all'uso di queste variabili booleane è stato possibile riusare lo stesso metodo per costruire tre funzioni di similarità diverse:

- La funzione di **baseline**, la quale usa solo i valori di Term Frequency e Inverse Document Frequency di ogni termine all'interno dei vettori come pesi di ciascuna parola.
- Una funzione che somma, oltre ai valori di Term Frequency e Inverse Document Frequency, anche i valori delle parole **simili** e **correlate**.
- Una funzione che somma, oltre ai valori di Term Frequency e Inverse Document Frequency, anche i valori delle sole parole **simili**.

Queste funzioni vengono usate: sia con il glossario e l'inverted index derivanti dall'estrazione da pagine HTML, che con il glossario e l'inverted index derivanti dall'estrazione delle pagine XML prodotte dal web service basato su COGITO, ciò viene fatto specificando in fase di chiamata al metodo **similarityComp.executeQuery** il **glossaryFile** e l'**indexFile** che si intendono utilizzare.

La funzione di similarità vera e propria, descritta nella sottosezione 2.3.1, è implementata all'interno del metodo **similarityComp.ambitSimilaritySingleDocV2**, il quale è applicato ad ogni documento del sito di e-commerce, e non fa altro che la somma dei punteggi, calcolati con la formula descritta nella sottosezione 2.3.1, di tutti i termini estratti dal documento che sono uguali

```
1       if (term1==term2):     # equal terms
2         bestT1Score=(EQ_SCORE*w1*w2)
3         # print (term1+", "+term2+" equal "+str(EQ_SCORE*w1*w2))
4         break
```

o eventualmente sinonimi

```
1       if (USE_SYNS and isWnSynonym(term1,term2,synTerms)):     # synonym terms
2         if (bestT1Score<SYN_SCORE*w1*w2):
3           bestT1Score=SYN_SCORE*w1*w2
4           #print (term1+", "+term2+" syns "+str(SYN_SCORE*w1*w2))
5         continue
```

o correlati

```
1       if ( USE_REL and isRelated(term1,term2,relTerms) ):     # related terms
2         if (bestT1Score<REL_SCORE*w1*w2):
3           bestT1Score=REL_SCORE*w1*w2
4           #print (term1+", "+term2+" rel "+str(REL_SCORE*w1*w2))
```

con le parole estratte dalle pagine web che costituiscono il profilo dell'utente.

3.4.2 Similarità attraverso le classi IPTC

Prima di tutto, per poter applicare tale similarità, è stato necessario estrarre le classi IPTC delle pagine del sito e-commerce e delle pagine del profilo dai file creati nella sottosezione 3.1.2 attraverso il metodo **similarityComp.extractIPTCs**.

In seguito è stato necessario un'ulteriore passo preliminare, in particolare ho unito le classi IPTC uguali tra tutte le pagine che formano il profilo dell'utente, sommando tra di loro gli score di ciascuna classe IPTC attraverso il metodo **similarityComp.mergeIPTCs**.

Tramite il metodo **similarityComp.similarityIPTC**, ho calcolato la similarità tra le classi IPTC del profilo dell'utente e le classi IPTC di ciascuna pagina web all'interno del sito di e-commerce, ricavando: prima la distanza sull'albero delle classi IPTC che c'è tra le due classi IPTC (indicata dalla variabile **iPath**) e in seguito il punteggio di similarità tra le due classi come descritto nella sottosezione 2.3.2, usando la distanza **iPath**; inoltre si può notare che nel caso in cui la distanza **iPath** tra le due classi sia uguale a 1, perciò le due classi sono identiche, allora il punteggio viene moltiplicato per lo score della classe IPTC che è stato estratto dal web service basato su COGITO, aumentando così il punteggio di similarità finale.

```
1         score = -math.log10(iPath/(2.0*5))
2         if iPath==1:
3           score *= int(profile[1])
```

Infine ho sommando questi i punteggi ottenuti per ogni pagina del sito di e-commerce per definire il punteggio finale di quella pagina.

```
1         if ranking == []:
2           ranking.append([score,document[0]])
3         else:
4           isIn = False
5           for doc in ranking:
6             if doc[1] == document[0]:
7               isIn=True
8               doc[0]=(doc[0]+score)
9           if isIn==False:
10            ranking.append([score,document[0]])
```

3.4.3 Ranking Fusion

Per poter applicare la tecnica di ranking fusion spiegata alla sottosezione 2.3.3 è stato necessario:

- in primo luogo, per i ranking ottenuti dai paragrafi estratti dalle pagine in HTML, è stato necessario sommarli tra loro per ottenere un nuovo ranking basato non più sui paragrafi ma sulle pagine; ciò è stato implementato nel metodo **similarityComp.paragraphFusion** nel modo seguente.

```
def paragraphFusion(paragRank):
  ranking = []
  totScore = 0.0
  for parag in paragRank:
    totScore += parag[0]
    curDoc = parag[1].split('-')[0]
    if ranking == []:
      ranking.append([parag[0],curDoc])
    else:
      isIn=False
      for doc in ranking:
        if doc[1] == curDoc:
          isIn=True
          doc[0] += parag[0]
      if isIn==False:
        ranking.append([parag[0],curDoc])
  sortedRank=sorted(ranking, key = lambda x: float(x[0]),reverse=True)
  return sortedRank
```

- in secondo luogo, per poter fondere due diversi ranking con punteggi di similarità espressi con scale di valori diversi, si è reso necessario normalizzare i valori di ciascun ranking ottenuto, sommando tutti i punteggi all'interno di ciascun ranking e dividendo questo valore per ogni singolo valore all'interno del ranking da trasformare, così da ottenere tutti dei valori che variano dallo 0 all'1; ciò è stato implementato nel metodo **similarityComp.normalizeRank** nel modo seguente.

```
def normalizeRank(ranking):
  normalized=[]
  totScore = 0.0
  for rankItem in ranking:
    totScore += rankItem[0]
  for rankItem in ranking:
    normalized.append([rankItem[0]/totScore,rankItem[1]])
  return normalized
```

Grazie a queste operazioni preliminari è stato possibile far funzionare al meglio la tecnica di ranking fusion implementata nel metodo **similarityComp.rankingFusion**, il quale applica alla lettera l'algoritmo pesato KE descritto nella sottosezione 2.3.3.

```
def rankingFusion(rank1,ranking2): # THE Weighted KE ALGORITHM
  ranking=[]
  m=2
  EWFMAX = max(len(rank1),len(ranking2))
  EWF1 = len(rank1)+1
  EWF2 = len(ranking2)+1
  k=len(rank1)+len(ranking2)
  count1 = 0
  for doc1 in rank1:
    count1+=1
    n=1
    isIn=False
    count2 =0
    for doc2 in ranking2:
      count2+=1
      if doc1[1]==doc2[1]:
        isIn=True
        num = ((doc1[0]*(EWF1-count1))+(doc2[0]*(EWF2-count2)))
        den = ((n+1)**m)*(((k/EWFMAX)+1)**(n+1))
        ranking.append([num/den,doc1[1]])
    if isIn==False:
      num = doc1[0]*(EWF1-count1)
      den = ((n)**m)*(((k/EWFMAX)+1)**(n))
      ranking.append([num/den,doc1[1]])
  sortedRank=sorted(ranking, key = lambda x: float(x[0]),reverse=True)
  return sortedRank
```

Capitolo 4

Prove sperimentali e risultati ottenuti

In questo capitolo presenterò le prove sperimentali che ho effettuato, per due ipotetici utenti coinvolti nella ricerca di prodotti all'interno del sito di e-commerce, per poter valutare le prestazioni degli algoritmi utilizzati e presentati nei capitoli precedenti.
Le analisi delle prestazioni, operata per ciascun ranking risultante dalle fasi precedenti, è stata realizzata per mezzo metodo **similarityComp.evaluateResults** il quale calcola, in base alle pagine fornite dal programmatore come rilevanti, i valori di: Precision, Recall[1], F measure, e mostra la precision a livelli standard di recall del ranking fornito.
Il contenuto di questo capitolo si divide nelle seguenti sezioni:

- L'insieme di dati utilizzati (Sezione 4.1)
- Prove sperimentali sul profilo dell'utente 1 (Sezione 4.2)
- Prove sperimentali sul profilo dell'utente 3 (Sezione 4.3)
- Valutazione finali sulle prove sperimentali (Sezione 4.4)

[1]Vedi sottosezione 1.5.3

4.1 L'insieme di dati utilizzati

Il progetto, come detto in precedenza, si pone in uno scenario di un sito e-commerce, formato da una moltitudine di pagine web riguardanti dei prodotti, in particolare ho deciso di scegliere dei prodotti omogenei tra loro, in modo tale da verificare in fase di ricerca l'effettiva correttezza dell'algoritmo di ranking; in particolare ho deciso di formare l'ipotetico sito di e-commerce da un insieme di 12 pagine web le quali riguardano:

- 3 Televisori
  - Un televisore di marca Samsung con codice TV001
  - Un televisore di marca Sony con codice TV002
  - Un televisore di marca Philips con codice TV003
- 3 Libri
  - Un libro intitolato 'The Lord of the Rings' con codice BOOK1
  - Un libro intitolato 'The Hobbit' con codice BOOK2
  - Un libro intitolato 'Dracula' con codice BOOK3
- 3 Videogiochi
  - Un videogioco intitolato 'The Last of Us' per Playstation 3 con codice GAME001
  - Un videogioco intitolato 'Assassin's Creed IV Black Flag' per Playstation 3 con codice GAME002
  - Un videogioco intitolato 'Watch Dogs' per Playstation 3 con codice GAME003
- 3 Cellulari
  - Un cellulare di marca Apple con codice CELL001
  - Un cellulare di marca Samsung con codice CELL002
  - Un cellulare di marca Motorola con codice CELL003

Anche il profilo di ciascun utente che interagisce con il sito e-commerce, è formato ciascuno da 6 pagine web riguardanti:

- Per il **Profilo dell'utente 1**
  - Una pagina di Wikipedia riguardante le televisioni
  - Un televisore di marca Seiki
  - Un televisore di marca LG
  - Un sistema di Dolby surround marca Bose
  - Un lettore Blu-ray marca Sony
- Per il **Profilo dell'utente 2**
  - Un libro intitolato 'A Shade Of Vampire'
  - Una pagina di Wikipedia riguardante i vampiri
  - Un ebook intitolato 'Dracula Chronicles: Empire of the Crescent Moon'
  - Un disco Blu-ray intitolato 'Interview With the Vampire'
  - Un gioco per computer intitolato 'Vampire: The Masquerade - Bloodlines'
  - Una T-shirt del telefilm 'Buffy the vampire slayer'
- Per il **Profilo dell'utente 3**
  - Un cellulare di marca Sony
  - Una fotocamera di marca Sony
  - Una console Playstation Portable di marca Sony
  - Un notebook della Sony
  - Un controller per Playstation 3 di marca Sony
  - Una console Playstation 3

Andrò a presentare le prove sperimentali effettuate sul profilo dell'utente 1 e sul profilo dell'utente 3 in quanto bastano per valutare la correttezza e l'effettiva efficienza degli algoritmi presentati in precedenza.

4.2 Prove sperimentali sul profilo dell'utente 1

L'utente 1 è inquadrato, grazie all'uso delle sue pagine visitate, come un soggetto alla ricerca di un televisore; per questo motivo indico al programma, tramite la variabile **QUERY_SOL**, come pagine rilevanti le tre pagine riguardanti i televisori, le quali hanno i codici: TV001, TV002 e TV003.

4.2.1 Uso del modello vettoriale esteso

Attraverso l'uso del modello vettoriale esteso, nel caso in cui si utilizzino i termini estratti dalle pagine HTML per mezzo della libreria **Topia**, indifferentemente che si usino i termini sinonimi e/o i termini correlati; ne risulta un ranking fatto in questo modo:

Pagine web	**Similarità**
TV002	0,4068
TV001	0,1413
GAME003	0,0983
BOOK001	0,0650
GAME001	0,0581
CELL001	0,0522
TV003	0,0510
BOOK003	0,0450
CELL003	0,0390
BOOK002	0,0276
GAME002	0,0140
CELL002	0,0018

Tabella 4.1: Ranking delle pagine usando i termini estratti dalle pagine HTML. (Utente 1)

nel quale si trovano ai primi posti soltanto due dei tre televisori attesi (TV002 e TV001), inoltre come si può notare il televisore mancante (TV003) ha un punteggio di ranking otto volte minore rispetto al primo televisore nel ranking (TV002), rendendo tale classifica poco efficiente.

Ciò non succede nel ranking che utilizza i dati presi dalla pagina XML pro-

dotta dal web service basato su COGITO, nel quale , indipendentemente che si usi o meno i termini sinonimi e/o correlati, ne risulta un ranking fatto in questo modo:

Pagine web	Similarità
TV002	0,1985
TV003	0,1530
TV001	0,1255
GAME003	0,1116
CELL001	0,0782
CELL003	0,0572
GAME001	0,0557
BOOK002	0,0459
CELL002	0,0456
BOOK001	0,0441
BOOK003	0,0435
GAME002	0,0414

Tabella 4.2: Ranking delle pagine usando i termini estratti dalle pagine XML (Relevants, Domains, Mainlemmas). (Utente 1)

nel quale si può notare che le pagine che erano state indicate come rilevanti nella variabile **QUERY_SOL** sono tutte alle prime posizione, restituendo così un ranking ottimale dove il valore di precision ad ogni livello standard di recall è sempre al massimo.

4.2.2 Uso della similarità con classi IPTC e ranking fusion

Grazie all'uso delle classi IPTC estratte dalle pagine XML restituite dal web service basato su COGITO, ne è risultato il seguente ranking delle pagine :

Pagine web	Similarità
TV001	0,2567
TV003	0,2499
TV002	0,2168

BOOK003	0,0659
CELL002	0,0473
CELL003	0,0473
BOOK001	0,0401
BOOK002	0,0401
GAME002	0,0119
GAME003	0,0119
CELL001	0,0075
GAME001	0,0046

Tabella 4.3: Ranking delle pagine usando le classi IPTC. (Utente 1)

Si può notare che anche in questo ranking le pagine, che erano state indicate come rilevanti nella variabile **QUERY_SOL**, sono tutte alle prime posizione rendendo anch'esso un ranking ottimale, ma ciò che lo distingue dalle classificazioni precedenti sta nel fatto che i valori delle tre pagine rilevanti sono molto vicini tra di loro, inoltre questi tre valori si discostano tre volte tanto dai punteggi delle pagine al di sotto, rendendo questo tipo di ranking **il migliore**, in termini di efficienza, fra quelli visti fin'ora.

Infine vediamo ciò che risulta dal ranking fusion tra il ranking delle pagine che usa i termini estratti dalle pagine HTML e il ranking delle pagine che usa le classi IPTC:

Pagine web	**Similarità**
TV002	0,3620
TV001	0,2380
TV003	0,1568
GAME003	0,0523
BOOK001	0,0424
BOOK003	0,0420
CELL003	0,0250
GAME001	0,0241
CELL001	0,0195
CELL002	0,0195

BOOK002	0,0145
GAME002	0,0039

Tabella 4.4: Ranking fusion tra il ranking che usa le classi IPTC e il ranking delle pagine che prende le informazioni dalle pagine HTML. (Utente 1)

L'unione dei due ranking permette di restituirne uno che si avvicina al ranking ottimale precedente, senza perdere precision e perdendo poca distanza tra il punteggio dei tre documenti rilevanti e tutti gli altri.

4.3 Prove sperimentali sul profilo dell'utente 3

L'utente 3 è inquadrato, grazie all'uso delle sue pagine visitate, come un soggetto alla ricerca di uno o più prodotti della marca Sony; per questo motivo indico al programma, tramite la variabile **QUERY_SOL**, come unica pagina rilevante la sola che riguarda un prodotto di marca Sony con codice TV002.

4.3.1 Uso del modello vettoriale esteso

Attraverso l'uso del modello vettoriale esteso, nel caso in cui si utilizzino i termini estratti dalle pagine HTML per mezzo della libreria **Topia**, indifferentemente che si usino i termini sinonimi e/o i termini correlati; ne risulta un ranking fatto in questo modo:

Pagine web	**Similarità**
TV002	0,2551
CELL001	0,1868
GAME003	0,1012
GAME001	0,0976
TV001	0,0855
CELL003	0,0816
CELL002	0,0744
GAME002	0,0304

BOOK002	0,0276
BOOK001	0,0214
TV003	0,0199
BOOK003	0,0184

Tabella 4.5: Ranking delle pagine usando i termini estratti dalle pagine HTML. (Utente 3)

Ciò che si può notare è l'effettiva correttezza del ranking con il televisore rilevante (TV002) alla prima posizione con un punteggio di similarità abbastanza superiore a tutti gli altri presenti nel ranking.

Lo stesso tipo di classificazione ma con alcune differenze in termini di punteggio, viene prodotto dal ranking che utilizza i dati presi dalla pagina XML prodotta dal web service basato su COGITO :

Pagine web	**Similarità**
TV002	0,1352
TV003	0,1202
CELL001	0,1091
CELL003	0,0967
TV001	0,0962
GAME003	0,0952
CELL002	0,0893
BOOK002	0,0721
GAME001	0,0598
GAME002	0,0443
BOOK001	0,0428
BOOK003	0,0393

Tabella 4.6: Ranking delle pagine usando i termini estratti dalle pagine XML (Relevants, Domains, Mainlemmas). (Utente 3)

Come si può notare i punteggi di similarità sono molto vicini tra loro, rendendo questo ranking non affidabile e non ottimale, a differenza di quello precedente.

4.3.2 Uso della similarità con classi IPTC e ranking fusion

Grazie all'uso delle classi IPTC estratte dalle pagine XML restituite dal web service basato su COGITO, ne è risultato il seguente ranking delle pagine :

Pagine web	Similarità
GAME002	0,1353
GAME003	0,1353
GAME001	0,1166
BOOK003	0,0885
CELL002	0,0875
CELL003	0,0875
TV001	0,0866
BOOK002	0,0720
BOOK001	0,0656
TV003	0,0624
CELL001	0,0428
TV002	0,0198

Tabella 4.7: Ranking delle pagine usando le classi IPTC.
(Utente 3)

Si può notare che il ranking restituito è pessimo, in quanto pone la pagina rilevante (TV002) all'ultima posizione della classifica, con un punteggio di similarità molto basso, mettendo al suo posto in cima al ranking i videogiochi GAME002, GAME003 e GAME001.

Per mezzo dell'uso della tecnica di ranking fusion tra il ranking delle pagine che usa i termini estratti dalle pagine HTML e il ranking delle pagine che usa le classi IPTC otteniamo il seguente ranking:

Pagine web	Similarità
TV002	0,1821
GAME003	0,1478
CELL001	0,1265
GAME001	0,1209
GAME002	0,1050
CELL003	0,0766

CELL002	0,0751
TV001	0,0571
BOOK003	0,0482
BOOK002	0,0278
BOOK001	0,0193
TV003	0,0134

Tabella 4.8: Ranking fusion tra il ranking che usa le classi IPTC e il ranking delle pagine che prende le informazioni dalle pagine HTML. (Utente 3)

L'unione dei due ranking permette di restituirne uno che si avvicina al ranking ottimale ottenuto alla sottosezione 4.3.1, senza perdere precision ma perdendo distanza tra il punteggio del documento rilevante (TV002) e tutti gli altri.

4.4 Valutazioni finali sulle prove sperimentali

Dalle prove sperimentali effettuate nelle sottosezioni 4.2 e 4.3 si può dedurre che:

- L'utente 1 essendo alla ricerca di una categoria di prodotti, quale le televisioni, riesce ottenere un ranking ottimale e affidabile solo nel caso in cui si utilizzino le classi IPTC estratte dal web service basato su COGITO, mentre nel caso in cui si usi il modello vettoriale esteso ciò che ne risulta è un ranking dalle scarse prestazioni; la ragione a cui ricondurre tale comportamento sta nella natura stessa delle classi IPTC, le quali catturano argomenti di tipo generale (per esempio 'IPTC/Arte, cultura, intrattenimento/Televisione', '1000'), e per mezzo del web service basato su COGITO, che permette una rapida classificazione di ciascuna pagina tramite le classi IPTC, si è in grado di catturare gli argomenti generale trattati dalle pagine analizzate e restituire così un ranking delle pagine basato su quest'ultimi; per questo motivo se un'utente è interessato, come in questo caso, a trovare prodotti generali il ranking ottimale è quello che utilizza le classi IPTC;

- Al contrario l'utente 3 che è interessato alla ricerca di prodotti specifici, cioè qualsiasi prodotto che sia di marchio Sony, riesce a ottenere un ranking ottimale e affidabile solo nel caso in cui si utilizzi il modello vettoriale esteso, mentre nel caso in cui si utilizzino le classi IPTC ne risulta un ranking pessimo; la ragione per cui ciò succede è da ricondurre al fatto che, dato che il modello vettoriale esteso calcola la similarità usando i termini del testo estratto da ciascuna pagina web, a differenza della similarità tramite le classi IPTC, riesce ad estrarre i termini specifici che sono più rilevanti per l'utente (in questo caso il termine 'Sony') sia dal profilo dell'utente che dalle pagine del profilo, riuscendo a soddisfare efficacemente l'User Information Need (UIN) di un utente in cerca di prodotti specifici.

Un'ultima nota di riguardo va fatta sull'uso della tecnica di ranking fusion, la quale applicandola tra il ranking migliore e quello più scarso, unendo i difetti di uno ai pregi dell'altro, permette di ottenere sempre un ranking ottimale e abbastanza affidabile delle pagine.

Capitolo 5

Conclusioni e Sviluppi futuri

In questo capitolo finale voglio esporre gli obiettivi che sono stati raggiunti rispetto quelli prefissati nella parte introduttiva di questa tesi di ricerca.

Tutti gli obiettivi che mi sono prefissato sono stati raggiunti nel miglior modo in base alle mie capacità e i limiti di tempo che ho dovuto rispettare, in particolare sono stati raggiunti gli obiettivi di:

- **Recupero e manipolazione dei dati** riguardanti le pagine web HTML del sito di e-commerce e del profilo dell'utente tramite l'uso del modulo **extractText.py**, il cui funzionamento è stato descritto nel dettaglio nella sottosezione 3.1.1.

- **Recupero e manipolazione dei dati** riguardanti le pagine web XML del sito di e-commerce e del profilo dell'utente estrapolati dal web service basato su COGITO tramite l'uso del modulo **cogito.py**, il cui funzionamento è stato descritto nel dettaglio nella sottosezione 3.1.2.

- **Studio e modifica** dell'algoritmo di similarità usato da AMBIT, per adattarlo alle esigenze dell'applicazione, progettato nella sottosezione 2.3.1 e implementato all'interno del metodo **similarityComp.ambitSimilaritySingleDocV2**, il quale è stato descritto nella sottosezione 3.4.1.

- **Studio, progettazione e realizzazione** di un algoritmo che costruisca una misura di similarità basata sulle classi IPTC estratte dai dati di COGITO, progettato nella sottosezione 2.3.2 e implementato all'interno

del metodo **similarityComp.similarityIPTC**, il quale è stato descritto nella sottosezione 3.4.2.

- **Studio, progettazione e realizzazione** di un algoritmo di ranking fusion, per combinare tra di loro diversi ranking delle pagine web, progettato nella sottosezione 2.3.3 e implementato all'interno del metodo **similarityComp.rankingFusion**, il quale è stato descritto nella sottosezione 3.4.3.

- **Valutazione** dell'efficacia degli algoritmi implementati tramite l'analisi dei risultati ottenuti attraverso le misure di: precision, recall, F measure e precision ad ogni livello standard di recall; implementata all'interno del progetto attraverso il metodo **similarityComp.evaluateResults**, di cui sono stati spiegati i risultati ottenuti nel capitolo 4.

Nonostante il raggiungimento di tutti gli obbiettivi che mi sono prefissato restano dei punti in cui il programma prodotto è migliorabile, quali:

- Sviluppo di una interfaccia grafica;

- Parallelizzazione dei processi di creazione dei glossari e degli inverted index;

- Uso di altri tipi di contesti oltre la cronologia dell'utente, per raffinare la ricerca;

- Introduzione di un metodo di Word Sense Disambiguation (WSD), che vada a disambiguare le parole estratte dalle pagine HTML, assegnando ad ogni termine il senso appropriato rispetto al suo contesto all'interno della frase in cui sta, permettendo al programma di selezionare i termini correlati e sinonimi corretti del termine disambiguato.

Appendice A

Achivio dei codici

A.1 ambit.py

Codice sorgente di ambit.py

```
# -*- coding: utf-8 -*-
import extractText
import cogito
import pickle
import glossaryExtractor
import similarityComp
import numpy as np
import numpy.linalg as LA

#============================================================================
# Per le pagine web da analizzare ho scelto:
#   - 3 pagine web riguardanti dei televisori
#   - 3 pagine web riguardanti dei libri
# - 3 pagine web riguardanti dei videogames
# - 3 pagine web riguardanti dei cellulari
# le pagine sono tutte prese da 'amazon' in quanto hanno più testo da poter
#analizzare.
#============================================================================

DOCS_LIST = [('TV001','http://www.amazon.com/Samsung-UN60H6203-60-Inch-1080p-120
    Hz/dp/B00K4UJ2S2/ref=sr_1_1?ie=UTF8&qid=1407848697&sr=8-1&keywords=samsung+
    television+60+inch'),
      ('TV002','http://www.amazon.com/Sony-KDL40W600B-40-Inch-1080p-Smart/dp/
    B00HPMCO46/ref=sr_1_1?ie=UTF8&qid=1407849219&sr=8-1&keywords=SONY+TELEVISION
    '),
      ('TV003','http://www.amazon.com/58PFL4909-58-1080p-LED-LCD-TV/dp/
    B00JP8ZTAE/ref=sr_1_11?ie=UTF8&qid=1407849252&sr=8-11&keywords=philips+
    television'),
      ('BOOK001','http://www.amazon.com/Lord-Rings-50th-Anniversary-Vol/dp
    /0618640150/ref=sr_1_1?s=books&ie=UTF8&qid=1407849512&sr=1-1&keywords=the+
    lord+of+the+rings'),
```

```
24          ('BOOK002','http://www.amazon.com/Hobbit-There-Back-Again/dp/054792822X/
        ref=sr_1_1?s=books&ie=UTF8&qid=1407849529&sr=1-1&keywords=THE+HOBBIT'),
25          ('BOOK003','http://www.amazon.com/Dracula-Dover-Thrift-Editions-Stoker/dp
        /0486411095/ref=sr_1_1?s=books&ie=UTF8&qid=1407849571&sr=1-1&keywords=
        DRACULA'),
26          ('GAME001','http://www.amazon.com/gp/product/B007CM0K86/ref=
        s9_hps_bw_g63_i5?pf_rd_m=ATVPDKIKX0DER&pf_rd_s=merchandised-search-7&pf_rd_r
        =0EQ455M5A27TXPTPJJ3K&pf_rd_t=101&pf_rd_p=1531308782&pf_rd_i=14210751'),
27          ('GAME002','http://www.amazon.com/gp/product/B00BMFIXKQ/ref=
        s9_hps_bw_g63_i10?pf_rd_m=ATVPDKIKX0DER&pf_rd_s=merchandised-search-7&
        pf_rd_r=1PNB7VH5Y9VESD11QQ6J&pf_rd_t=101&pf_rd_p=1531308782&pf_rd_i=14210751
        '),
28          ('GAME003','http://www.amazon.com/gp/product/B00BGHUS58/ref=
        s9_hps_bw_g63_i7?pf_rd_m=ATVPDKIKX0DER&pf_rd_s=merchandised-search-7&pf_rd_r
        =1PNB7VH5Y9VESD11QQ6J&pf_rd_t=101&pf_rd_p=1531308782&pf_rd_i=14210751'),
29          ('CELL001','http://www.amazon.com/Apple-iPhone-5s-Gold-Unlocked/dp/
        B00F3J4E5U/ref=sr_1_1?s=wireless&ie=UTF8&qid=1408721078&sr=1-1&keywords=
        iphone+5s'),
30          ('CELL002','http://www.amazon.com/Samsung-SM-G900H-Factory-Unlocked-
        International/dp/B00J4TK4B8/ref=sr_1_1?s=wireless&ie=UTF8&qid=1408721096&sr
        =1-1&keywords=samsung+s5'),
31          ('CELL003','http://www.amazon.com/Motorola-Moto-Global-Unlocked-Black/dp/
        B00GWR373M/ref=sr_1_5?s=wireless&ie=UTF8&qid=undefined&sr=1-5&keywords=
        motorola')]
32
33  #=============================================================================
34  # Per le pagine del primo profilo ho preso
35  # - 2 pagine riguardanti dei televisori
36  # - 1 pagina di wikipedia riguardante i televisori
37  # - 1 pagina che riguarda un'impianto di dolby surround
38  # - 1 pagina che riguarda una PS4
39  # - 1 pagina che riguarda un lettore blu-ray
40  # così da avere alla fine un ranking più alto delle pagine riguardanti i
        televisori
41  # tra quelle dei documenti.
42  #=============================================================================
43
44  PROFILE1_LIST = [('P001','http://en.wikipedia.org/wiki/Television'),
45          ('P002','http://www.amazon.com/Seiki-SE55UY04-55-Inch-120Hz-Black/dp/
        B00IMNTH10/ref=sr_1_17?s=electronics&ie=UTF8&qid=1407849694&sr=1-17'),
46          ('P003','http://www.amazon.com/LG-Electronics-49UB8500-49-Inch-Ultra/dp/
        B00II6VY2G/ref=sr_1_18?s=electronics&ie=UTF8&qid=1407849694&sr=1-18'),
47          ('P004','http://www.amazon.com/Bose%C2%AE-Solo-TV-Sound-System/dp/
        B008EWNVI4/ref=lp_281056_1_2?s=tv&ie=UTF8&qid=1408723885&sr=1-2'),
48          ('P005','http://www.amazon.com/PlayStation-4-Console/dp/B00BGA9WK2/ref=
        sr_1_1?s=tv&ie=UTF8&qid=1408723908&sr=1-1'),
49          ('P006','http://www.amazon.com/Sony-BDPS3200-Blu-ray-Player-Wi-Fi/dp/
        B00HPMCO4Q/ref=sr_1_4?s=tv&ie=UTF8&qid=1408723937&sr=1-4')]
50  #=============================================================================
51  # Per le pagine del secondo profilo ho preso
52  # - 2 pagine riguardanti libri sui vampiri
53  # - 1 pagina di wikipedia riguardante i vampiri
54  # - 1 pagina che riguarda 'intervista con il vampiro'
```

```
55 # - 1 pagina che riguarda un gioco per pc sui vampiri
56 # - 1 pagina che riguarda una maglietta di 'buffy l'ammazzavampiri'
57 #=============================================================================
58
59 PROFILE2_LIST = [('P001','http://www.amazon.com/Shade-Vampire-Book-One/dp
       /1481280767/ref=sr_1_1?s=books&ie=UTF8&qid=1409758634&sr=1-1&keywords=
       vampire'),
60           ('P002','http://en.wikipedia.org/wiki/Vampire'),
61           ('P003','http://www.amazon.com/Dracula-Chronicles-Empire-Crescent-Moon-
       ebook/dp/B00IZ3XEGQ/ref=sr_1_31?s=books&ie=UTF8&qid=1409758871&sr=1-31&
       keywords=vampire'),
62           ('P004','http://www.amazon.com/Interview-Vampire-20th-Anniversary-Blu-
       ray/dp/B00LXPR0EA/ref=sr_1_3?s=movies-tv&ie=UTF8&qid=1409758964&sr=1-3&
       keywords=vampire'),
63           ('P005','http://www.amazon.com/Vampire-Masquerade-Bloodlines-Pc/dp/
       B0001NJHH8/ref=sr_1_2?s=videogames&ie=UTF8&qid=1409759065&sr=1-2&keywords=
       the+masquerade'),
64           ('P006','http://www.amazon.com/Buffy-Vampire-Slayer-Sunnydale-
       Razorbacks/dp/B00DR4IZZS/ref=sr_1_4?ie=UTF8&qid=undefined&sr=8-4&keywords=
       buffy')]
65 #=============================================================================
66 # Per le pagine del TERZO profilo ho preso
67 # - 1 pagina riguardante un cellulare della sony
68 # - 1 pagina riguardante una fotocamera della sony
69 # - 1 pagina che riguarda una ps3
70 # - 1 pagina che riguarda una psp
71 # - 1 pagina che riguarda un controller per ps3
72 # - 1 pagina che riguarda un notebook vaio della sony
73 #=============================================================================
74 PROFILE3_LIST = [('P001','http://www.amazon.com/Sony-Unlocked-Android-Phone---U-S
       -Warranty/dp/B00BNR5AWM/ref=sr_1_1?ie=UTF8&qid=1409760661&sr=8-1&keywords=
       SONY'),
75           ('P002','http://www.amazon.com/Sony-Interchangeable-Digital-Camera
       -18-55mm/dp/B00EH5UGR6/ref=sr_1_16?ie=UTF8&qid=1409760661&sr=8-16&keywords=
       SONY'),
76           ('P003','http://www.amazon.com/PlayStation-Portable-3000-System-Sony-
       PSP/dp/B001KMRN0M/ref=sr_1_74?ie=UTF8&qid=1409760752&sr=8-74&keywords=SONY')
       ,
77           ('P004','http://www.amazon.com/Sony-SVP1321ACXS-13-Inch-Touchscreen-
       Ultrabook/dp/B00D3YVD6A/ref=sr_1_146?ie=UTF8&qid=undefined&sr=8-146&keywords
       =SONY'),
78           ('P005','http://www.amazon.com/PlayStation-Dualshock-Wireless-
       Controller-Blue-3/dp/B002GJRQRS/ref=sr_1_154?ie=UTF8&qid=1409760809&sr
       =8-154&keywords=SONY'),
79           ('P006','http://www.amazon.com/Sony-Computer-Entertainment-Playstation-
       System-3/dp/B00E369SDM/ref=sr_1_179?ie=UTF8&qid=undefined&sr=8-179&keywords=
       SONY')]
80
81 DOCS_FILE = 'allDocs.txt'
82 COG_FILE = 'allAnalyses.dat'      # list: docId, (domains, entities, relevants)
83 GLOSSARY_FILE = 'gloss.dat'       # termTFs:  doc-code,term,tf (array), termIDFs
       : term:idf (dict)
84 COG_GLOSSARY_FILE = 'cogGloss.dat'       # termTFs:  doc-code,term,tf (array),
```

```
    termIDFs: term:idf (dict)
85 INDEX_FILE = 'index.dat'          # term:termlist (dict), doc:docTerms (dict),
    term:relTermsList (dict)
86 COG_INDEX_FILE = 'cogIndex.dat'   # term:termlist (dict), doc:docTerms (dict),
    term:relTermsList (dict)
87
88 PROFILE1_FILE = 'allProfile1.txt'
89 COG_FILE_PROF1 = 'prof1Analyses.dat'
90 PROF1_GLOSSARY = 'prof1Gloss.dat'
91 PROF1_COG_GLOSSARY = 'prof1CogGloss.dat'
92
93 PROFILE2_FILE = 'allProfile2.txt'
94 COG_FILE_PROF2 = 'prof2Analyses.dat'
95 PROF2_GLOSSARY = 'prof2Gloss.dat'
96 PROF2_COG_GLOSSARY = 'prof2CogGloss.dat'
97
98 PROFILE3_FILE = 'allProfile3.txt'
99 COG_FILE_PROF3 = 'prof3Analyses.dat'
100 PROF3_GLOSSARY = 'prof3Gloss.dat'
101 PROF3_COG_GLOSSARY = 'prof3CogGloss.dat'
102
103 QUERY_SOL = ["TV002","GAME001","GAME002","GAME003"]
104
105 # 1a. EXTRACT TEXT WITH TOPIA
106 para = extractText.multiExtractText(DOCS_LIST, DOCS_FILE)   # list: docId-numPar
    , text
107 for p in para:
108   print p
109
110 # 1b. EXTRACT TEXT WITH COGITO
111 resp = cogito.multiAnalyzeText(DOCS_LIST, COG_FILE)   # list: docId, (domains,
    entities, relevants)
112 for r in resp:
113   print r
114
115 #2a. EXTRACT PROFILE WITH TOPIA
116 para = extractText.multiExtractText(PROFILE3_LIST, PROFILE3_FILE)   # list:
    docId-numPar, text
117 for p in para:
118   print p
119
120 # 2b. EXTRACT TEXT PROFILE WITH COGITO
121 resp = cogito.multiAnalyzeText(PROFILE3_LIST, COG_FILE_PROF3)   # list: profId,
    (domains, entities, relevants)
122 for r in resp:
123   print r
124
125 # 3a. GENERATE GLOSSARY AND INDEX WITH COGITO FILE
126 glossaryExtractor.go(COG_FILE, COG_GLOSSARY_FILE, True)
127 similarityComp.genIndexFile(COG_GLOSSARY_FILE,COG_INDEX_FILE)
128
129 # 3b. GENERATE GLOSSARY AND INDEX WITHOUT COGITO FILE
130 glossaryExtractor.go(DOCS_FILE, GLOSSARY_FILE, False)
```

```
131 similarityComp.genIndexFile(GLOSSARY_FILE,INDEX_FILE)
132
133 # 4a. GENERATE PROFILE GLOSSARY AND INDEX WITH COGITO FILE
134 glossaryExtractor.go(COG_FILE_PROF3, PROF3_COG_GLOSSARY,True)
135
136 # 4b. GENERATE PROFILE GLOSSARY AND INDEX WITHOUT COGITO FILE
137 glossaryExtractor.go(PROFILE3_FILE, PROF3_GLOSSARY,False)
138
139 print '1a) RANKING TRA IL PROFILO ESTRAPOLATO SENZA COGITO E IL GLOSSARIO DEI
        DOCUMENTI ESTRAPOLATO SENZA COGITO SENZA SYNS E SENZA RELATED'
140 prof_tfs1 ,prof_idfs1 = similarityComp.readGlossary(PROF3_GLOSSARY)
141 Query1 = [] #Costruisco un vettore con le parole contenute dentro il glossario
        del profilo per poterle usare nella funzione similarityComp.executeQuery
142 for prof_tf in prof_tfs1:
143   if prof_tf[1] in prof_idfs1.keys():
144     Query1.append((prof_tf[1],prof_tf[2]*prof_idfs1[prof_tf[1]]))
145   else:
146     Query1.append((prof_tf[1],prof_tf[2]))
147 ranking1a = similarityComp.executeQuery(Query1,GLOSSARY_FILE,INDEX_FILE,False,
        False)
148 similarityComp.printRanking(ranking1a)
149 rank1a=similarityComp.paragraphFusion(ranking1a)
150 similarityComp.printRanking(rank1a)
151 rank1a = similarityComp.normalizeRank(rank1a)
152 similarityComp.printRanking(rank1a)
153 similarityComp.evaluateResults(QUERY_SOL,rank1a)
154
155
156 print '1b) RANKING TRA IL PROFILO ESTRAPOLATO SENZA COGITO E IL GLOSSARIO DEI
        DOCUMENTI ESTRAPOLATO SENZA COGITO CON SYNS E RELATED'
157 ranking1b = similarityComp.executeQuery(Query1,GLOSSARY_FILE,INDEX_FILE,True,
        True)
158 similarityComp.printRanking(ranking1b)
159 rank1b=similarityComp.paragraphFusion(ranking1b)
160 similarityComp.printRanking(rank1b)
161 rank1b = similarityComp.normalizeRank(rank1b)
162 similarityComp.printRanking(rank1b)
163 similarityComp.evaluateResults(QUERY_SOL,rank1b)
164
165 print '1c) RANKING TRA IL PROFILO ESTRAPOLATO SENZA COGITO E IL GLOSSARIO DEI
        DOCUMENTI ESTRAPOLATO SENZA COGITO CON SYNS SENZA RELATED'
166 ranking1c = similarityComp.executeQuery(Query1,GLOSSARY_FILE,INDEX_FILE,True,
        False)
167 similarityComp.printRanking(ranking1c)
168 rank1c=similarityComp.paragraphFusion(ranking1c)
169 similarityComp.printRanking(rank1c)
170 rank1c = similarityComp.normalizeRank(rank1c)
171 similarityComp.printRanking(rank1c)
172 similarityComp.evaluateResults(QUERY_SOL,rank1c)
173
174
175 print '2a) RANKING TRA IL PROFILO ESTRAPOLATO CON COGITO E IL GLOSSARIO DEI
        DOCUMENTI ESTRAPOLATO CON COGITO SENZA SYNS E SENZA RELATED'
```

```
176 prof_tfs2 ,prof_idfs2 = similarityComp.readGlossary(PROF3_COG_GLOSSARY)
177 Query2 = []
178 for prof_tf in prof_tfs2:
179   if prof_tf[1] in prof_idfs2.keys():
180     Query2.append((prof_tf[1],prof_tf[2]*prof_idfs2[prof_tf[1]]))
181   else:
182     Query2.append((prof_tf[1],prof_tf[2]))
183 rank2a = similarityComp.executeQuery(Query2,COG_GLOSSARY_FILE,COG_INDEX_FILE,
      False,False)
184 similarityComp.printRanking(rank2a)
185 rank2a = similarityComp.normalizeRank(rank2a)
186 similarityComp.printRanking(rank2a)
187 similarityComp.evaluateResults(QUERY_SOL,rank2a)
188
189
190 print '2b) RANKING TRA IL PROFILO ESTRAPOLATO CON COGITO E IL GLOSSARIO DEI
      DOCUMENTI ESTRAPOLATO CON COGITO CON SYNS E RELATED'
191 rank2b = similarityComp.executeQuery(Query2,COG_GLOSSARY_FILE,COG_INDEX_FILE,
      True,True)
192 similarityComp.printRanking(rank2b)
193 rank2b = similarityComp.normalizeRank(rank2b)
194 similarityComp.printRanking(rank2b)
195 similarityComp.evaluateResults(QUERY_SOL,rank2b)
196
197 print '2c) RANKING TRA IL PROFILO ESTRAPOLATO CON COGITO E IL GLOSSARIO DEI
      DOCUMENTI ESTRAPOLATO CON COGITO CON SYNS E SENZA RELATED'
198 rank2c = similarityComp.executeQuery(Query2,COG_GLOSSARY_FILE,COG_INDEX_FILE,
      True,False)
199 similarityComp.printRanking(rank2c)
200 rank2c = similarityComp.normalizeRank(rank2c)
201 similarityComp.printRanking(rank2c)
202 similarityComp.evaluateResults(QUERY_SOL,rank2c)
203
204 print "3a) RANKING CON LE CLASSI IPTC DI COGITO"
205 docIPTCs,profIPTCs = similarityComp.extractIPTCs(COG_FILE,COG_FILE_PROF3)
206 mergedIPTCs = similarityComp.mergeIPTCs(profIPTCs)
207 rank3a = similarityComp.similarityIPTC(docIPTCs,mergedIPTCs)
208 similarityComp.printRanking(rank3a)
209 rank3a = similarityComp.normalizeRank(rank3a)
210 similarityComp.printRanking(rank3a)
211 similarityComp.evaluateResults(QUERY_SOL,rank3a)
212
213
214 print "4a) RANKING FUSION TRA IL RANKING 1a e 3"
215 if len(rank1a)>=len(rank3a):
216   rankFus1a = similarityComp.rankingFusion(rank1a, rank3a)
217 else:
218   rankFus1a = similarityComp.rankingFusion(rank3a, rank1a)
219 similarityComp.printRanking(rankFus1a)
220 similarityComp.evaluateResults(QUERY_SOL,rankFus1a)
221 print 'ESTRAGGO I PARAGRAFI DEI DOCUMENTI NEL RANKING FUSION 1 DAL RANKING 1a'
222 parag4a = similarityComp.returnParagraph(rankFus1a, ranking1a)
223 similarityComp.printRanking(parag4a)
```

```
224
225
226 print "4b) RANKING FUSION TRA IL RANKING 1b e 3"
227 if len(rank1b)>=len(rank3a):
228   rankFus1b = similarityComp.rankingFusion(rank1b, rank3a)
229 else:
230   rankFus1b = similarityComp.rankingFusion(rank3a, rank1b)
231 similarityComp.printRanking(rankFus1b)
232 similarityComp.evaluateResults(QUERY_SOL,rankFus1b)
233 print 'ESTRAGGO I PARAGRAFI DEI DOCUMENTI NEL RANKING FUSION 2 DAL RANKING 1b'
234 parag4b = similarityComp.returnParagraph(rankFus1b, ranking1b)
235 similarityComp.printRanking(parag4b)
236
237 print "4c) RANKING FUSION TRA IL RANKING 1c e 3"
238 if len(rank1c)>=len(rank3a):
239   rankFus1c = similarityComp.rankingFusion(rank1c, rank3a)
240 else:
241   rankFus1c = similarityComp.rankingFusion(rank3a, rank1c)
242 similarityComp.printRanking(rankFus1c)
243 similarityComp.evaluateResults(QUERY_SOL,rankFus1c)
244 print 'ESTRAGGO I PARAGRAFI DEI DOCUMENTI NEL RANKING FUSION 2 DAL RANKING 1b'
245 parag4c = similarityComp.returnParagraph(rankFus1c, ranking1b)
246 similarityComp.printRanking(parag4c)
247
248
249 print "5a) RANKING FUSION TRA IL RANKING 2a e 3"
250 if len(rank2a)>=len(rank3a):
251   rankFus2a = similarityComp.rankingFusion(rank2a, rank3a)
252 else:
253   rankFus2a = similarityComp.rankingFusion(rank3a, rank2a)
254 similarityComp.printRanking(rankFus2a)
255 similarityComp.evaluateResults(QUERY_SOL,rankFus2a)
256 print 'ESTRAGGO I PARAGRAFI DEI DOCUMENTI NEL RANKING FUSION 2 DAL RANKING 1a'
257 parag5a = similarityComp.returnParagraph(rankFus2a, ranking1a)
258 similarityComp.printRanking(parag5a)
259
260
261 print "5b) RANKING FUSION TRA IL RANKING 2b e 3"
262 if len(rank2b)>=len(rank3a):
263   rankFus2b = similarityComp.rankingFusion(rank2b, rank3a)
264 else:
265   rankFus2b = similarityComp.rankingFusion(rank3a, rank2b)
266 similarityComp.printRanking(rankFus2b)
267 similarityComp.evaluateResults(QUERY_SOL,rankFus2b)
268 print 'ESTRAGGO I PARAGRAFI DEI DOCUMENTI NEL RANKING FUSION 2 DAL RANKING 1a'
269 parag5b = similarityComp.returnParagraph(rankFus2b, ranking1a)
270 similarityComp.printRanking(parag5b)
271
272 print "5c) RANKING FUSION TRA IL RANKING 2c e 3"
273 if len(rank2c)>=len(rank3a):
274   rankFus2c = similarityComp.rankingFusion(rank2c, rank3a)
275 else:
276   rankFus2c = similarityComp.rankingFusion(rank3a, rank2c)
```

```
similarityComp.printRanking(rankFus2c)
similarityComp.evaluateResults(QUERY_SOL,rankFus2c)
print 'ESTRAGGO I PARAGRAFI DEI DOCUMENTI NEL RANKING FUSION 2 DAL RANKING 1a'
parag5c = similarityComp.returnParagraph(rankFus2c, ranking1a)
similarityComp.printRanking(parag5c)
```

A.2 extractText.py

Codice sorgente di extractText.py

```
import re
import HTMLParser
import urllib

def extractText(docId, url):
  print "Extracting "+url+"..."

  paragrafi = []
  count = 1
  h = HTMLParser.HTMLParser()
  MIN_PAR_LEN = 150
  MIN_TEXT_RATIO = 0.2

  # apre pagina web
  data = urllib.urlopen(url).read()
  data = data.decode("ISO-8859-1")

  # elimina a capo
  data = data.replace("\n", " ")
  data = data.replace("\r", " ")

  # estrae il body
  bodyPat = re.compile(r'<body[^<>]*?>(.*?)</body>', re.I)
  result = re.findall(bodyPat, data)
  data = result[0]

  # elimina java script
  p = re.compile(r'<script[^<>]*?>.*?</script>')
  data = p.sub('', data)

  # elimina css
  p = re.compile(r'<style[^<>]*?>.*?</style>')
  data = p.sub('', data)

  # elimina commenti
  p = re.compile(r'<!--(.)*?-->')
  data = p.sub('', data)

  # estrae i paragrafi
  parPat = re.compile(r'<(p|div)[^<>]*?>(.*?)</(p|div)>', re.I)
```

```
41    result = re.findall(parPat, data)
42    for par in result:
43      par = par[1]
44      pHTMLLen = len(par)
45
46      # elimina tutti i tag
47      p = re.compile(r'<[^<]*?>')
48      par = p.sub(' ', par)
49      par = p.sub(' ', par)
50
51      # rimuove spazi consecutivi
52      par = " ".join(par.split())
53
54      # unescape
55      par = h.unescape(par)
56
57      # unicode to ascii
58      par = par.encode('ascii', 'ignore')
59
60      # calcolo rapporto testo/codice
61      pTextLen = len(par)
62      if pTextLen == 0:
63        continue
64      pTextRatio = float(pTextLen)/pHTMLLen
65      # print str(pTextLen)+" "+str(pTextRatio)+" -> "+par
66
67      # filtra i paragrafi per lunghezza e per rapporto testo/codice
68      if pTextLen>MIN_PAR_LEN and pTextRatio > MIN_TEXT_RATIO:
69        paragrafi.append((docId+"-"+str(count),par))
70        count = count+1
71    return paragrafi
72
73  def multiExtractText(docsList,docsFile):
74    pars = []
75    for doc in docsList:
76      docId = doc[0]
77      url = doc[1]
78      par = extractText(docId, url)
79      pars.extend(par)
80    outF = open(docsFile,'w')
81    for p in pars:
82      if p!=pars[0]:
83        outF.write('\n')
84      outF.write(p[0]+"\t"+p[1])
85    outF.close()
86    print "Done!"
87    return pars
```

A.3 cogito.py

Codice sorgente di cogito.py

```
import re
import urllib
import urllib2
import pickle

def analyzeText(url):
  print "Analysing "+url+"..."
  urlS = "http://217.26.90.209/ESDemo/Analyzer"
  query_args = { 'customerKey':'unimore', 'url':url }
  data = urllib.urlencode(query_args)
  request = urllib2.Request(urlS, data)
  response = urllib2.urlopen(request)
  xml = response.read()
  d = extractDomains(xml)
  e = extractEntities(xml)
  r = extractRelevants(xml)
  return (d,e,r)

def multiAnalyzeText(docsList,outFile):
  analyses = []
  for doc in docsList:
    docId = doc[0]
    url = doc[1]
    an = analyzeText(url)
    analyses.append((docId,an))
  f_out = open(outFile,"w")
  pickle.dump(analyses,f_out)
  f_out.close()
  print "Pickling OK!"
  return analyses

def extractDomains(data):
  ris = []
  pat = re.compile(r'<DOMAIN[^<>]*?NAME="([^"]*?)" SCORE="([^"]*?)"/>', re.I)
  result = re.findall(pat, data)
  for r in result:
    ris.append((r[0],r[1]))
  return ris

def extractEntities(data):
  ris = []
  lastType = ""
  pat = re.compile(r'<ENTITY NAME="([^"]*?)">', re.I)
  patType = re.compile(r'<ENTITIES TYPE="([^"]*?)">', re.I)
  for line in data.splitlines():
    result = re.findall(pat, line)
    if len(result)>0:
      ris.append((result[0],lastType))
    else:
```

```
50        result = re.findall(patType, line)
51        if len(result)>0:
52          lastType = result[0].lower()
53    return ris
54
55  def extractRelevants(data):
56    ris = []
57    lastType = ""
58    pat = re.compile(r'<RELEVANT NAME="([^"]*?)"[^<>]*?(RANKING="([^"]*?)")
        ?[^<>]*?SCORE="([^"]*?)"/?>', re.I)
59    patType = re.compile(r'<RELEVANTS TYPE="([^"]*?)">', re.I)
60    for line in data.splitlines():
61      result = re.findall(pat, line)
62      if len(result)>0:
63        ris.append((result[0][0],result[0][3],lastType))
64      else:
65        result = re.findall(patType, line)
66        if len(result)>0:
67          lastType = result[0].lower()
68    return ris
```

A.4 glossaryExtractor.py

Codice sorgente di glossaryExtractor.py

```
1  # -*- coding: utf-8 -*-
2  import math
3  import pickle
4  from topia.termextract import tag
5  from topia.termextract import extract
6  from nltk.corpus import wordnet as wn
7
8  def countNumberOfDocuments(inputFile):              # count number of paragraphs
       and documents
9    f = open(inputFile,'rU')
10   numDocs = 0
11   numPars = 0
12   lastDoc = ""
13   for line in f:
14     line = line.strip()
15     docId = line[:line.find("\t")]        # doc code
16     if sameDoc(lastDoc,docId) == False:
17       numDocs +=1
18       lastDoc = docId
19     numPars += 1
20   f.close()
21   return numPars, numDocs
22
23 def sameDoc(id1,id2):      # check if two paragraphs are from same doc
24   if id1.split('-')[0] == id2.split('-')[0]:
```

```
25      return True
26    else:
27      return False
28
29  def extractTermData(inputFile):                    # extract term data with
      Topia package
30    MIN_OCCURS = 2     # minimum number of occurrences for term extraction (if not
      using DefaultFilter)
31    docTerms = []     # term, code, freq, strength (temp data)
32    docLengths = []   # code, len
33    extractor = extract.TermExtractor()
34    extractor.filter = extract.permissiveFilter
35    # extractor.filter = extract.DefaultFilter(singleStrengthMinOccur=MIN_OCCURS)
36    f = open(inputFile,'rU')
37    for line in f:                          # for each doc: extract term data
38      curDocLen = 0
39      line = line.strip()
40      qmCode = line[:line.find("\t")]        # qm code
41      qmText = line[(line.find("\t")+1):].lower()   # qm text description (lower
      case)
42      #print extractor.tagger(qmText)         # POS-tagged terms
43      for term in sorted(extractor(qmText)):        # extracted terminology
44        if ( "." in term[0] or "(" in term[0] or ")" in term[0] or "+" in term[0]
        # filter out unwanted terms
45                            or "\xa0" in term[0] or "\x80" in term[0] or "/" in
      term[0]
46                            or ":" in term[0] or term[0].isdigit() or len(term
      [0])<3):
47            continue
48        docTerm = (term[0].lower(), qmCode, term[1], term[2])   # term, code, freq
      , strength
49        docTerms.append(docTerm)
50        curDocLen = curDocLen+1
51        #print docTerm
52      docLengths.append((qmCode,curDocLen))
53      docTerms.append(("zzzzz","zzzzz",-1,-1))   # last docTerm (terminator)
54    f.close()
55    print docTerms
56    print docLengths
57    return docTerms, docLengths
58
59
60
61  #la funzione estrae dal file di cogito i relevants i mainlemmas e i domains
62  #così da utilizzarli in seguito per costruire il glossario e l'inverted index
63  def extractCogitoData(cogFile):
64    print "Reading cogito data..."
65    docTerms=[]
66    docLenghts=[]
67    pickle_file = open(cogFile)
68    cogito_file = pickle.load(pickle_file)
69    pickle_file.close()
70    for docId in cogito_file:
```

```
71      i=0
72      currDocLen=0
73      for ent_rel in docId[1]:
74        if (i==0) :
75          i+=1
76          continue
77        for currWord in ent_rel:
78          currDocLen+=1
79          if (currDocLen==1):
80            docTerms.append((currWord[0].lower(),docId[0],1))
81          else:
82            count=0
83            isInside=False
84            for pastTerm in docTerms:
85              if (pastTerm[1]== docId[0]):
86                if (currWord[0].lower()==pastTerm[0]):
87                  docTerms[count] = list(docTerms[count])
88                  docTerms[count][2]+=1
89                  docTerms[count] = tuple(docTerms[count])
90                  isInside=True
91              count+=1
92            if (isInside == False):
93                docTerms.append((currWord[0].lower(),docId[0],1))
94        i+=1
95      docLenghts.append((docId[0],currDocLen))
96      docTerms.append(("zzzzz","zzzzz",-1))
97    numDocs = len(cogito_file)
98    print docTerms
99    print docLenghts
100   return docTerms,docLenghts,numDocs
101
102 def generateGlossary(docTerms,numDocs,USE_COGITO):            # glossary
      generation
103   curTerm = ""     # current term
104   termIDFs = {}     # term, idf
105   termCount = 0     # number of terms in glossary
106   for docTerm in sorted(docTerms):
107     if (docTerm[0]!=curTerm):     # new term
108       if curTerm!="":
109         curFreq = len(curDocList)
110         curIDF = math.log(numDocs/curFreq)
111         termIDFs[curTerm] = curIDF
112         termCount +=1
113       curTerm=docTerm[0]
114       #curStren=docTerm[3]
115       curDocList = []
116     if (USE_COGITO):
117       curDoc = docTerm[1]
118     else:
119       curDoc = docTerm[1].split('-')[0]     # extract doc id from paragraph id
120     if curDoc not in curDocList:
121       curDocList.append(curDoc)
122   return termCount, termIDFs
```

```
123
124 def generateReport(docTerms, docLengths, termIDFs):                    # report
      generation
125   termTFs = []    # doc-code, term, tf
126   for doc in sorted(docLengths):        # for each document
127     curDoc = doc[0]
128     curDocLen = doc[1]
129     for docTerm in docTerms:
130       if (docTerm[1]==curDoc):    # relevant term
131         curTerm = docTerm[0]    # term
132         curFreq = docTerm[2]    # current term frequency (in current doc)
133         if curDocLen==0:
134           continue
135         curTF = float(curFreq)/curDocLen     # current term TF
136         curOutData = (curDoc, curTerm, curTF)
137         termTFs.append(curOutData)
138   return termTFs
139
140 def go(inputFile,outFile,USE_COGITO):
141   if (USE_COGITO) :
142     print 'with cogito...'
143     docTerms,docLengths,numDocs=extractCogitoData(inputFile)
144   else:
145     numPars, numDocs = countNumberOfDocuments(inputFile)
146     print "Total number of documents: "+str(numDocs)
147     print "Total number of paragraphs: "+str(numPars)
148     print "Extracting glossary data..."
149     print 'without cogito...'
150     docTerms, docLengths = extractTermData(inputFile)
151     # print sorted(docTerms)
152     # print docLengths
153   termCount, termIDFs = generateGlossary(docTerms,numDocs,USE_COGITO)
154   termTFs = generateReport(docTerms, docLengths, termIDFs)
155   print "...done! ("+str(termCount)+" terms, "+str(len(termTFs))+" occs
      extracted)"
156   f_out = open(outFile,"w")
157   glossaryData = (termTFs,termIDFs)    # list:  doc-code,term,tf (array), term:
      idf (dict)
158   # print termTFs
159   # print termIDFs
160   pickle.dump(glossaryData,f_out)
161   f_out.close()
162   print "Pickling OK!"
```

A.5 similarityComp.py

Codice sorgente di similarityComp.py

```
1 # -*- coding: utf-8 -*-
2 import pickle
```

```
import locale
import math
from nltk.corpus import wordnet as wn


def disambiguateTerms(terms,verbose=False):
  for t in terms:    # t is target term
    selSense = None
    selScore = 0.0
    for s_ti in wn.synsets(t, wn.NOUN):
      score_i = 0.0
      for w_j in terms:    # w_j word in t's context windows
        if (t==w_j):
          continue
        bestScore = 0.0
        for s_jk in wn.synsets(w_j, wn.NOUN):
          tempScore = s_ti.path_similarity(s_jk)
          #tempScore = s_ti.lch_similarity(s_jk)
          #tempScore = s_ti.wup_similarity(s_jk)
          if (tempScore>bestScore):
            bestScore=tempScore
        score_i = score_i + bestScore
      if (verbose):
        print t+" "+str(score_i) + " " + str(s_ti)+", "+s_ti.definition
      if (score_i>selScore):
        selScore = score_i
        selSense = s_ti
    if (not verbose):
      if (selSense is not None):
        print t+": "+str(selSense)+", "+selSense.definition
      else:
        print t+": ---"

def wnSimilarity(term1, term2):     # similarity between 0 and 1
  bestScore = 0.0
  for s_t1 in wn.synsets(term1, wn.NOUN):
    for s_t2 in wn.synsets(term2, wn.NOUN):
      tempScore = s_t1.path_similarity(s_t2)
      #tempScore = s_t1.lch_similarity(s_t2)
      #tempScore = s_t1.wup_similarity(s_t2)
      if (tempScore>bestScore):
        bestScore=tempScore
  return bestScore

def glossSimilarity(term1, term2, ieeeDefs):
  bestScore = 0
  if (term1 in ieeeDefs) and (term2 in ieeeDefs):
    stemmedDefs1 = ieeeDefs[term1][1]
    stemmedDefs2 = ieeeDefs[term2][1]
    for def1 in stemmedDefs1:
      for def2 in stemmedDefs2:
        tempScore = extGlossOverlap(def1,def2)
        if (tempScore>bestScore):
```

```
56            bestScore=tempScore
57    return bestScore
58
59  def extGlossOverlap(def1, def2):
60    #print def1
61    #print def2
62    score = 0
63    d1 = def1.split()
64    d2 = def2.split()
65    i=0
66    while(i<len(d1)):   # cycle on def1 terms
67      term1=d1[i]
68      #print ("Term1: "+term1)
69      bestScore = 0
70      j=0
71      while(j<len(d2)):    # cycle on def2 terms
72        term2=d2[j]
73        #print ("Term2: "+term2)
74        if term1==term2:
75          length = getMatchLength(d1,i,d2,j)
76          if bestScore<length*length:
77            bestScore=length*length
78          #print ("Match "+str(length))
79          i=i+length-1
80          j=j+length-1
81        j=j+1
82      score = score+bestScore
83      i=i+1
84    return score
85
86  def getMatchLength(def1, i, def2, j):
87    l=1
88    #print (str(def1) + " " + str(i) + " " + str(def2) + " " + str(j))
89    while ((i+l < len(def1)) and (j+l < len(def2))):
90      term1=def1[i+l]
91      term2=def2[j+l]
92      if term1==term2:
93        l=l+1
94      else:
95        break
96    return l
97
98  def isWnRelated(term1, term2):
99    if term1 == term2:
100     return True
101   if wnSimilarity(term1, term2) >= WN_THR:
102     return True
103   return False
104
105 def isRelated(term1, term2, relTerms):
106   if term1 == term2:
107     return True
108   if term1 in relTerms:
```

```
109     if term2 in relTerms[term1]:
110       return True
111   return False
112 
113 def isWnSynonym(term1, term2, synTerms):
114   if term1 == term2:
115     return True
116   if ((term1 in synTerms) and (term2 in synTerms)):  #AGGIUNTO
117     syns1 = synTerms[term1]
118     syns2 = synTerms[term2]
119   #print (syns2)
120     if (term1 in syns2) or (term2 in syns1):
121       return True
122   return False
123 
124 def genAllWnRelated(termIDFs):
125   relTerms = {}
126   for term1 in sorted(iter(termIDFs)):
127     print ("TERM: "+term1)
128     terms = getWnRel(term1)
129     print terms
130     relTerms[term1]=terms
131   return relTerms
132 
133 def genAllWnSyns(termIDFs):
134   synTerms = {}
135   for term1 in sorted(iter(termIDFs)):
136     print ("TERM: "+term1)
137     terms = getWnSyns(term1)
138     print (terms)
139     synTerms[term1]=terms
140   return synTerms
141 
142 def getWnRel(term):
143   rel = []
144   for s in wn.synsets(term):
145     for hype in s.hypernym_distances():
146       he = hype[0].name.split('.')[0].replace('_',' ')
147       if ((hype[1]<=2) and (he not in rel)and(he!=term)):
148         if (len(wn.synsets(he))==1):
149           #print he
150           rel.append(he)
151     for hypo in s.hyponyms():
152       dist = 0
153       for ho in hypo.hypernym_distances():
154         if s == ho[0]:
155           dist = ho[1]
156       for ho in hypo.hypernym_distances():
157         h = ho[0].name.split('.')[0].replace('_',' ')
158         if ((ho[1]<= dist)and((dist-ho[1])<=2)and(h not in rel)and(h!=term)):
159           if (len(wn.synsets(h))==1):
160             #print h
161             rel.append(h)
```

```
162   rel = sorted(list(set(rel)))
163   #print rel
164   return rel
165 
166 def getWnSyns(term):
167   syns = []
168   for s in wn.synsets(term):
169     for lemma in s.lemmas:
170       if (len(wn.synsets(lemma.name.replace('_',' ')))==1):
171         syn = lemma.name.replace('_',' ')
172         syns.append(syn)
173   syns = sorted(list(set(syns)))
174   #print syns
175   return syns
176 
177 def genInvertedIndex(termTFs):
178   invIndex = {}
179   for t in termTFs:
180     doc = t[0]
181     term = t[1]
182     if (term in invIndex):
183       invIndex[term].append(doc)
184     else:
185       list = []
186       list.append(doc)
187       invIndex[term]=list
188   print invIndex.keys()
189   print invIndex.values()
190   return invIndex
191 
192 def genDocTerms(termTFs):
193   docTerms = {}
194   for t in termTFs:
195     doc = t[0]
196     term = t[1]
197     tf = t[2]
198     if (doc in docTerms):
199       docTerms[doc].append((term,tf))
200     else:
201       list = []
202       list.append((term,tf))
203       docTerms[doc]=list
204   return docTerms
205 
206 def genIndexFile(glossaryFile,indexFile):
207   termTFs, termIDFs = readGlossary(glossaryFile)
208   invIndex = genInvertedIndex(termTFs)
209   docTerms = genDocTerms(termTFs)
210   synTerms = genAllWnSyns(termIDFs)
211   relTerms = genAllWnRelated(termIDFs)
212   f_out = open(indexFile,"w")
213   indexData = (invIndex, docTerms, synTerms, relTerms)    # term:termlist (dict)
    , doc:docTerms (dict), term:synTermsList (dict), term:relTermsList (dict)
```

```
214   pickle.dump(indexData,f_out)
215   print "Done!"
216   f_out.close()
217
218 def ambitSimilaritySingleDocV2(terms1,termsTF2,termIDFs,synTerms,relTerms):
219   score = 0.0
220   #print terms1
221   #print termsTF2
222   for termTf1 in terms1:
223     term1 = termTf1[0]
224     tf1 = termTf1[1]
225     w1=1
226     if (USE_WEIGHTS):
227       if (term1 in termIDFs):
228         idf1=termIDFs[term1]
229         w1=tf1*idf1
230       else:
231         w1=NO_WEIGHT
232     bestT1Score = 0
233     for docTermTf2 in termsTF2:
234       term2 = docTermTf2[0]
235       tf2 = docTermTf2[1]
236       if term2 not in termIDFs:
237         continue
238       idf2=termIDFs[term2]
239       w2=1
240       if (USE_WEIGHTS):
241         w2=tf2*idf2
242       if (term1==term2):    # equal terms
243         bestT1Score=(EQ_SCORE*w1*w2)
244         # print (term1+", "+term2+" equal "+str(EQ_SCORE*w1*w2))
245         break
246       if (USE_SYNS and isWnSynonym(term1,term2,synTerms)):    # synonym terms
247         if (bestT1Score<SYN_SCORE*w1*w2):
248           bestT1Score=SYN_SCORE*w1*w2
249           #print (term1+", "+term2+" syns "+str(SYN_SCORE*w1*w2))
250         continue
251       if ( USE_REL and isRelated(term1,term2,relTerms) ):    # related terms
252         if (bestT1Score<REL_SCORE*w1*w2):
253           bestT1Score=REL_SCORE*w1*w2
254           #print (term1+", "+term2+" rel "+str(REL_SCORE*w1*w2))
255     score = score+bestT1Score
256   return score
257
258 def ambitSimilarityV2(terms1,termTFs,termIDFs,synTerms,relTerms,invIndex,
      docTerms):
259   ranking = []
260   unionDocSet = set([])
261   i=0
262   for termTf1 in terms1:
263     term1 = termTf1[0]
264     if (term1 in invIndex):
265       docSet = set(invIndex[term1])
```

```
266         unionDocSet = unionDocSet.union(docSet)
267       if (USE_SYNS):
268         if (term1 in synTerms):
269           syns = synTerms[term1]
270           for syn in syns:
271             if (syn in invIndex):
272               docSet = set(invIndex[syn])
273               unionDocSet = unionDocSet.union(docSet)
274       if (USE_REL):
275         if (term1 in relTerms):
276           rels = relTerms[term1]
277           for rel in rels:
278             if (rel in invIndex):
279               docSet = set(invIndex[rel])
280               unionDocSet = unionDocSet.union(docSet)
281     docs = sorted(unionDocSet)
282     #print docs
283     for doc in docs:
284       termsTF2=docTerms[doc]
285       score = ambitSimilaritySingleDocV2(terms1,termsTF2,termIDFs,synTerms,
        relTerms)
286       # print ".. DOC "+doc
287       if (score>SCORE_THR):
288         ranking.append((score,doc))
289     ranking = sorted(ranking,reverse=True)
290     return ranking
291
292 def printRanking(ranking):
293     for res in ranking:
294       scoreFormatted = locale.format("%0.4f", res[0])
295       print (res[1]+"\t"+scoreFormatted)
296
297 def evaluateResults(querySol,ranking):
298     queryRetr = []
299     precision = 0.0
300     recall = 0.0
301     print ranking
302     for tuple in ranking:
303       #doc = tuple
304       doc = tuple[1]
305       queryRetr.append(doc)
306     if (ONLY_FIRST and len(queryRetr)>len(querySol)):      # only first results are
        considered
307       queryRetr=queryRetr[0:len(querySol)]
308     retrieved = set(queryRetr)      # precision and recall computation
309     relevant = set(querySol)
310     relRetr = retrieved.intersection(relevant)
311     if (len(retrieved) and len(relevant)>0):
312       precision = (0.0 + len(relRetr))/len(retrieved)
313       recall = (0.0 + len(relRetr))/len(relevant)
314     f = -1
315     if (recall+precision>0):
316       f = 2*recall*precision/(recall+precision)
```

```
317   print ("")
318   print (str(len(retrieved))+" retrieved docs")
319   print (str(len(relevant))+" relevant docs")
320   print (str(len(relRetr))+" relevant retrieved docs")
321   print (str(precision)+" precision")
322   print (str(recall)+" recall")
323   print (str(f)+" f measure")
324   print ("")
325   numrel = 0.0    # precision at standard recall levels computation
326   numret = 0.0
327   precisions = []
328   for retr in queryRetr:
329     numret=numret+1
330     if retr in relevant:
331       numrel=numrel+1
332       print ("P("+str(numrel/len(relevant))+")="+str(numrel/numret))
333       precisions.append((numrel/len(relevant),numrel/numret))
334   i=1.0
335   precMax=0.0
336   print ("")
337   while (i>=0.0):
338     for tuple in precisions:
339       r=tuple[0]
340       p=tuple[1]
341       if (r>=(i-0.001) and precMax<p):
342         precMax=p
343     print ("P("+str(i)+")="+str(precMax))
344     print (locale.format("%0.4f", precMax))
345     i=i-0.1
346     if (i>0 and i<0.01):
347       i=0.0
348   # ranking distance computation
349   dist=0.0
350   print("")
351   for retr in queryRetr:
352     a=queryRetr.index(retr)
353     b=len(querySol)
354     if (retr in querySol):
355       b=querySol.index(retr)
356     dist=dist+math.fabs(a-b)
357     #print (str(a)+" "+str(dist))
358     print (locale.format("%0.4f", dist))
359
360 def readGlossary(glossaryFile):
361   print "Reading semantic glossary data..."
362   pickle_file = open(glossaryFile)
363   glossaryData = pickle.load(pickle_file)    # list:  doc-code,term,tf (array),
      term:idf (dict)
364   pickle_file.close()
365   termTFs = glossaryData[0]       # array: doc-code, term, tf
366   termIDFs = glossaryData[1]       # dictionary: term: idf
367   print "...done! ("+str(len(termIDFs))+" terms, "+str(len(termTFs))+" occs)"
368   return termTFs, termIDFs
```

```
369
370 def readIndex(indexFile):
371   print "Reading index data..."
372   pickle_file = open(indexFile)
373   indexData = pickle.load(pickle_file)
374   pickle_file.close()
375   invIndex = indexData[0]       # term:termlist (dict)
376   docTerms = indexData[1]       # doc:docTerms (dict)
377   synTerms = indexData[2]       # term:synTermsList (dict)
378   relTerms = indexData[3]       # term:relTermsList (dict)
379   print "...done! ("+str(len(invIndex))+" invIndex terms, "+str(len(synTerms))+"
      synTerms, "+str(len(relTerms))+" relTerms)"
380   return invIndex, docTerms, synTerms, relTerms
381
382 def executeQuery(queryTerms,glossaryFile,indexFile,USE_SYNS_NEW,USE_REL_NEW):
383   global USE_REL
384   USE_REL=USE_REL_NEW
385   global USE_SYNS
386   USE_SYNS=USE_SYNS_NEW
387   termTFs, termIDFs = readGlossary(glossaryFile)
388   invIndex, docTerms, synTerms, relTerms = readIndex(indexFile)
389   syns = {}
390   ranking = ambitSimilarityV2(queryTerms,termTFs,termIDFs,synTerms,relTerms,
      invIndex,docTerms)
391   return ranking
392
393 #estraggo dai file di cogito riguardanti i documenti e il profilo,
394 # le rispettive classi iptc per ogni documento.
395 def extractIPTCs(documents,profile): # ectract (list) docID,[IPTCs] and (list)
      profID,[IPTCs]
396   pickle_file = open(documents)
397   docFile = pickle.load(pickle_file)
398   pickle_file.close()
399   pickle_file = open(profile)
400   profFile = pickle.load(pickle_file)
401   pickle_file.close()
402   docIPTCs = []
403   for docId in docFile:
404     i=0
405     for IPTC in docId[1]:
406       if i==0 :
407         docIPTCs.append((docId[0],IPTC))
408       i+=1
409   profIPTCs = []
410   for profId in profFile:
411     i=0
412     for IPTC in profId[1]:
413       if i==0 :
414         profIPTCs.append(IPTC)
415       i+=1
416   #print docIPTCs
417   #print profIPTCs
418   return docIPTCs,profIPTCs
```

```
419
420 #funzione usata per unire tra di loro le classi iptc uguali, sommando i loro
        score
421 def mergeIPTCs(IPTCs):
422   mergedIPTCs = []
423   for profIPTC in IPTCs:
424     #print profIPTC
425     for IPTC in profIPTC:
426       if (mergedIPTCs == []):
427           mergedIPTCs.append(list(IPTC))
428       else:
429         isIn=False
430         for merged in mergedIPTCs:
431           if (merged[0] == IPTC[0]):
432             isIn = True
433             score = int(merged[1]) + int(IPTC[1])
434             merged[1] = str(score)
435         if (isIn==False):
436           mergedIPTCs.append(list(IPTC))
437   sortedIPTCs=sorted(mergedIPTCs, key = lambda x: int(x[1]),reverse=True)
438   #print sortedIPTCs
439   return sortedIPTCs
440
441 #Calcola gli score facendo la differenza [-log10(numero di passi tra le due
        classi/2*h)],
442 #moltiplicando poi questo score  nel caso in cui
443 #la differenza di cammino sia uguale a 1(non c'è differenza tra le due classi
        IPTC)
444 # per lo score della classe IPTC del
445 #profilo estratto da cogito.
446 def similarityIPTC(documents,profiles):
447   ranking = []
448   iProf = 1
449   for profile in profiles:
450     for document in documents:
451       iDoc=0
452       for IPTC in document[1]:
453         curProf = profile[0].split('/')
454         curDoc = IPTC[0].split('/')
455         intersection = set(curProf).intersection(curDoc)
456         iPath = 1
457         score = 0.0
458         for profPath in curProf:
459           if profPath not in intersection:
460             iPath+=1
461         for docPath in curDoc:
462           if docPath not in intersection:
463             iPath+=1
464         score = -math.log10(iPath/(2.0*5))
465         if iPath==1:
466           score *= int(profile[1])
467         if ranking == []:
468           ranking.append([score,document[0]])
```

```
469         else:
470           isIn = False
471           for doc in ranking:
472             if doc[1] == document[0]:
473               isIn=True
474               doc[0]=(doc[0]+score)
475           if isIn==False:
476             ranking.append([score,document[0]])
477 
478         iDoc+=1
479     iProf+=1
480   sortedRank=sorted(ranking, key = lambda x: float(x[0]),reverse=True)
481   return sortedRank
482 
483 #similarità tra le classi IPTC che considera solo le classi uguali tra di loro
484 def similarityIPTC_raw(documents,profiles):
485   ranking = []
486   iProf = 1
487   for profile in profiles:
488     #molt = (len(profiles)+1)-iProf
489     for document in documents:
490       i=0
491       for IPTC in document[1]:
492         if ((IPTC[0]==profile[0])):
493           if ranking == []:
494             ranking.append([int(IPTC[1]),document[0]])
495           else:
496             isIn = False
497             for doc in ranking:
498               if doc[1] == document[0]:
499                 isIn=True
500                 score = (int(doc[0])+int(IPTC[1]))
501                 doc[0]=score
502             if isIn==False:
503               ranking.append([int(IPTC[1]),document[0]])
504         i+=1
505     iProf+=1
506   sortedRank=sorted(ranking, key = lambda x: float(x[0]),reverse=True)
507   return sortedRank
508 
509 #somma gli score dei paragrafi di ogni documento
510 def paragraphFusion(paragRank):
511   ranking = []
512   totScore = 0.0
513   for parag in paragRank:
514     totScore += parag[0]
515     curDoc = parag[1].split('-')[0]
516     if ranking == []:
517       ranking.append([parag[0],curDoc])
518     else:
519       isIn=False
520       for doc in ranking:
521         if doc[1] == curDoc:
```

```
522             isIn=True
523             doc[0] += parag[0]
524         if isIn==False:
525           ranking.append([parag[0],curDoc])
526     sortedRank=sorted(ranking, key = lambda x: float(x[0]),reverse=True)
527     return sortedRank
528
529 #Funzione di ranking fusion (THE Weighted KE ALGORITHM)
530 def rankingFusion(rank1,ranking2): # THE Weighted KE ALGORITHM
531     ranking=[]
532     m=2
533     EWFMAX = max(len(rank1),len(ranking2))
534     EWF1 = len(rank1)+1
535     EWF2 = len(ranking2)+1
536     k=len(rank1)+len(ranking2)
537     count1 = 0
538     for doc1 in rank1:
539       count1+=1
540       n=1
541       isIn=False
542       count2 =0
543       for doc2 in ranking2:
544         count2+=1
545         if doc1[1]==doc2[1]:
546           isIn=True
547           num = ((doc1[0]*(EWF1-count1))+(doc2[0]*(EWF2-count2)))
548           den = ((n+1)**m)*(((k/EWFMAX)+1)**(n+1))
549           ranking.append([num/den,doc1[1]])
550       if isIn==False:
551         num = doc1[0]*(EWF1-count1)
552         den = ((n)**m)*(((k/EWFMAX)+1)**(n))
553         ranking.append([num/den,doc1[1]])
554     sortedRank=sorted(ranking, key = lambda x: float(x[0]),reverse=True)
555     return sortedRank
556     #print sortedRank
557
558 # THE Weighted KE ALGORITHM considerante solo le posizioni nei ranking degli
      oggrtti
559 def rankingFusion_positions(rank1,ranking2):
560     ranking=[]
561     m=2
562     EWFMAX = max(len(rank1),len(ranking2))
563     EWF1 = len(rank1)+1
564     EWF2 = len(ranking2)+1
565     k=len(rank1)+len(ranking2)
566     count1 = 0
567     for doc1 in rank1:
568       count1+=1
569       n=1
570       isIn=False
571       count2 =0
572       for doc2 in ranking2:
573         count2+=1
```

```
574         if doc1[1]==doc2[1]:
575           isIn=True
576           num = (EWF1-count1)+(EWF2-count2)
577           den = ((n+1)**m)*(((k/EWFMAX)+1)**(n+1))
578           ranking.append([num/den,doc1[1]])
579       if isIn==False:
580         num = EWF1-count1
581         den = ((n)**m)*(((k/EWFMAX)+1)**(n))
582         ranking.append([num/den,doc1[1]])
583     sortedRank=sorted(ranking, key = lambda x: float(x[0]),reverse=True)
584     return sortedRank
585     #print sortedRank
586 
587 #ritorna in ordine di ranking i paragrafi dei documenti all'interno di un
        ranking dei paragrafi
588 def returnParagraph(ranking,rankParag):
589     rankedParag = []
590     for doc in ranking:
591       for parag in rankParag:
592         curDoc = parag[1].split('-')[0]
593         if curDoc==doc[1]:
594           rankedParag.append(parag)
595     return rankedParag
596 
597 def normalizeRank(ranking):
598     normalized=[]
599     totScore = 0.0
600     for rankItem in ranking:
601       totScore += rankItem[0]
602     for rankItem in ranking:
603       normalized.append([rankItem[0]/totScore,rankItem[1]])
604     return normalized
605 
606 
607 USE_SYNS = True
608 USE_REL = True
609 USE_WEIGHTS = True
610 NO_WEIGHT = 0.5
611 EQ_SCORE = 1.0
612 SYN_SCORE = 1.0
613 REL_SCORE = 0.7
614 WN_THR = 0.3
615 SCORE_THR = 0
616 ONLY_FIRST = False
617 
618 
619 ##getWnSyns('')
620 #getWnRel('dog')
621 
622 #print wnSimilarity("mouse","shiner")
623 #print glossSimilarity("activity","authority",ieeeDefs)
624 #print extGlossOverlap("X A C X C A B C X X","Y A B C")
625 #print extGlossOverlap("Y A B C","X A C X C A B C X X")
```

```
626 #print isWnRelated("procedure","method")
627 #print isIeeeRelated("document","documentation",ieeeDefs)
628 #print isIeeeSynonym("acquirer","buyer",ieeeSyns)
629 #genAllWnRelated(termIDFs)
630
631
632 # locale.setlocale(locale.LC_ALL, 'it_IT')
633 # termTFs, termIDFs = readGlossary(GLOSSARY_FILE)
634 # ieeeDefs, ieeeSyns = readIEEEVocabulary(IEEE_FILE)
```

Bibliografia

[1] Gregory D Abowd, Anind K Dey, Peter J Brown, Nigel Davies, Mark Smith, and Pete Steggles. Towards a better understanding of context and context-awareness. In *Handheld and ubiquitous computing*, pages 304–307. Springer, 1999.

[2] Leonidas Akritidis, Dimitrios Katsaros, and Panayiotis Bozanis. Effective ranking fusion methods for personalized metasearch engines. In *Informatics, 2008. PCI'08. Panhellenic Conference on*, pages 39–43. IEEE, 2008.

[3] Sonia Bergamaschi, Riccardo Martoglia, and Serena Sorrentino. A semantic method for searching knowledge in a software development context. In *SEBD*, pages 115–122, 2012.

[4] Peter J Brown. The stick-e document: a framework for creating context-aware applications. *ELECTRONIC PUBLISHING-CHICHESTER-*, 8:259–272, 1995.

[5] Jim Christensen, Jeremy Sussman, Stephen Levy, William E Bennett, Tracee Vetting Wolf, and Wendy A Kellogg. Too much information. *Queue*, 4(6):50–57, 2006.

[6] Anind K Dey, Gregory D Abowd, and Daniel Salber. A conceptual framework and a toolkit for supporting the rapid prototyping of context-aware applications. *Human-computer interaction*, 16(2):97–166, 2001.

[7] David Franklin and Joshua Flaschbart. All gadget and no representation makes jack a dull environment. In *Proceedings of the AAAI 1998 Spring Symposium on Intelligent Environments*, pages 155–160, 1998.

[8] Daniela Godoy and Analía Amandi. Learning browsing patterns for context-aware recommendation. In *Artificial Intelligence in Theory and Practice*, pages 61–70. Springer, 2006.

[9] Richard Hull, Philip Neaves, and James Bedford-Roberts. Towards situated computing. In *Wearable Computers, 1997. Digest of Papers., First International Symposium on*, pages 146–153. IEEE, 1997.

[10] Peter Ingwersen and Kalervo Järvelin. *The turn: Integration of information seeking and retrieval in context*, volume 18. Springer, 2006.

[11] Peter Ingwersen and Kalervo Järvelin. Information retrieval in context-irix: workshop at sigir 2004-sheffield. In *ACM SIGIR Forum*, volume 38, pages 6–9. ACM, 2004.

[12] Eija Kaasinen. User needs for location-aware mobile services. *Personal and ubiquitous computing*, 7(1):70–79, 2003.

[13] Claudia Leacock and Martin Chodorow. Combining local context and wordnet similarity for word sense identification. *WordNet: An electronic lexical database*, 49(2):265–283, 1998.

[14] Carla Teixeira Lopes. Context features and their use in information retrieval. In *Third BCS-IRSG symposium on future directions in information access*, 2009.

[15] Charith Perera, Arkady Zaslavsky, Peter Christen, and Dimitrios Georgakopoulos. Context aware computing for the internet of things: A survey. *Communications Surveys & Tutorials, IEEE*, 16(1):414–454, 2014.

[16] Andrei Rikitianskii, Morgan Harvey, and Fabio Crestani. University of lugano at the trec 2013 contextual suggestion track.

[17] Tom Rodden, Keith Cheverst, K Davies, and Alan Dix. Exploiting context in hci design for mobile systems. In *Workshop on human computer interaction with mobile devices*, pages 21–22. Citeseer, 1998.

[18] Nick S Ryan, Jason Pascoe, and David R Morse. Enhanced reality fieldwork: the context-aware archaeological assistant. In *Computer applications in archaeology*. Tempus Reparatum, 1998.

[19] Bill N Schilit and Marvin M Theimer. Disseminating active map information to mobile hosts. *Network, IEEE*, 8(5):22–32, 1994.

[20] Xuehua Shen, Bin Tan, and ChengXiang Zhai. Context-sensitive information retrieval using implicit feedback. In *Proceedings of the 28th annual international ACM SIGIR conference on Research and development in information retrieval*, pages 43–50. ACM, 2005.

[21] Ahu Sieg, Bamshad Mobasher, and Robin Burke. Web search personalization with ontological user profiles. In *Proceedings of the sixteenth ACM conference on Conference on information and knowledge management*, pages 525–534. ACM, 2007.

[22] Esther Meng-Yoke Tan, Schubert Foo, Dion Hoe-Lian Goh, and Yin-Leng Theng. Tiles: classifying contextual information for mobile tourism applications. In *Aslib Proceedings*, volume 61, pages 565–586. Emerald Group Publishing Limited, 2009.

[23] David Vallet, Pablo Castells, Miriam Fernández, Phivos Mylonas, and Yannis Avrithis. Personalized content retrieval in context using ontological knowledge. *Circuits and Systems for Video Technology, IEEE Transactions on*, 17(3):336–346, 2007.

[24] Andy Ward, Alan Jones, and Andy Hopper. A new location technique for the active office. *Personal Communications, IEEE*, 4(5):42–47, 1997.