Università degli studi
di Modena e Reggio Emilia

Dipartimento di Scienze Fisiche,
Informatiche e Matematiche
Corso di Laurea in Informatica

Progetto AMBIT: Ottimizzazione e Valutazione Sperimentale del Motore di Ricerca Semantico Basato sul Contesto

Martina Pucella

Tesi di Laurea

Relatore:
Ing. Riccardo Martoglia

Anno Accademico 2013/2014

Ringraziamenti

Desidero ringraziare l'Ing. Riccardo Martoglia per la grande disponibilità e la costante assistenza che mi ha fornito. Ringrazio i miei genitori, per avermi sempre sostenuto, appoggiato ed aiutato a raggiungere questo traguardo. Grazie infine al mio compagno e ai miei amici, che tanto mi hanno dato in questi anni, per avermi supportato e soprattutto sopportato.

Parole chiave

Contesto
Information Retrieval
Semantica
Testo
Classificazione

Indice

Introduzione

Nel corso dell'ultimo decennio, l'ottimizzazione incessante dell'efficacia del recupero delle informazioni ha guidato lo sviluppo dei motori di ricerca verso nuovi livelli di qualità. La ricerca sul web per esempio è diventata uno standard e spesso è la fonte preferita per la ricerca di informazioni[16].

Spesso però il problema del recupero delle informazioni è trattato utilizzando una singola query e un insieme di documenti, da cui si ha solo un indizio molto limitato circa il bisogno di informazioni dell'utente (User Information Need). Un sistema di recupero ottimale dovrebbe quindi cercare di sfruttare tutte le possibili informazioni aggiuntive relative al contesto dell'utente per migliorare la precisione del recupero[26].

Una gestione dei dati consapevole del contesto degli utenti è di fondamentale importanza, in quanto, per creare un vero ambiente intelligente bisogna conoscere lo stato in cui si trovano gli utenti, così da poter comprendere i loro desideri e le loro richieste.

L'obiettivo del progetto AMBIT[1], presentato in questa Tesi, è proprio l'utilizzo del contesto all'interno di applicazioni di Information Retrieval. Questo progetto 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.

Per verificare che questi sistemi di recupero siano effettivamente efficaci, bisogna poter effettuare prove sperimentali e valutazioni per misurare le effettive performance del sistema.

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

Misurare la performance relativa ai sistemi di Information Retrieval (IR), come i motori di ricerca Web, è essenziale per la ricerca, lo sviluppo e per la qualità della ricerca di monitoraggio in ambienti dinamici. Tuttavia, a causa delle dimensioni e della natura dinamica delle collezioni di documenti e degli utenti, valutare o confrontare le prestazioni di recupero dei vari motori di ricerca è molto difficile. La valutazione automatica dei sistemi di recupero è la soluzione definitiva a questo problema. Valutare l'efficacia dei sistemi di IR normalmente richiede un insieme di test, una serie di domande e le informazioni rilevanti su ciascun documento rispetto ad ogni query. Tuttavia, per i database di grandi dimensioni è un compito difficile e che richiede molto tempo, dal momento che tutti i documenti devono essere giudicati per rilevanza su ogni query[18].

La Tesi riguarda l'ottimizzazione e la valutazione sperimentale di un motore di ricerca semantico basato sul contesto. Lo scenario utilizzato è quello di un sistema di help-desk intelligente, che una società può utilizzare per comunicare con i propri clienti in modo efficiente.

L'obiettivo di questa Tesi è quello di cercare di ottimizzare alcune delle funzionalità applicative del progetto per aumentarne l'efficienza nel recupero dei dati, come la classificazione dei documenti e dei profili utente, la creazione dei ranking in base al calcolo della similarità e le tecniche di ranking fusion.

In dettaglio, gli obiettivi che si cercherà di raggiungere sono:

- **Ottimizzazione** del ranking estratto tramite l'utilizzo del modello Vettoriale[2];
- **Ottimizzazione** del ranking estratto tramite il software COGITO[3], che utilizza le classi IPTC per categorizzare i documenti;
- **Studio e valutazione** delle funzionalità standard del software AMBIT, come l'utilizzo dei termini sinonimi e/o correlati;
- **Studio e valutazione** di differenti tecniche di ranking fusion;
- **Studio e valutazione** di una tecnica che in modo efficiente possa attribuire ai ranking un valore di importanza, da utilizzare nella fase di ranking fusion;

[2]vedi sezione 1.6

[3]Tecnologia semantica della ditta Expert System di Modena

Sulla base di questi obiettivi che ci si è prefissati, la Tesi è stata strutturata nel seguente modo:

- **Parte 1 - Stato dell'arte**
  - **Capitolo 1 - Inquadramento Teorico.** Cos'è e cosa si intende per Contesto (*Sezione 1.1*); Cosa sono le applicazioni context-aware e quali sono le loro caratteristiche (*Sezione 1.2*); Cos'è l'Information Retrieval, quali obiettivi si pone e qual è il suo processo (*Sezione 1.3*); Perchè è utile utilizzare il contesto nel processo di Information Retrieval (*Sezione 1.4*); Piccola panoramica sui modelli di Information Retrieval (*Sezione 1.5*); Cos'è il modello Vettoriale, quali sono le sue proprietà e i suoi indici di valutazione (*Sezione 1.6*); Come viene effettuata la fase di ranking dei risultati (*Sezione 1.7*).
  - **Capitolo 2 - AMBIT.** Cos'è il progetto AMBIT, quali obiettivi si pone e quali sono le applicazioni previste dal progetto (*Sezione 2.1*); Quali sono e come sono sviluppate le varie funzionalità applicative: profili utente, collezione di documenti, glossari e inverted index, utilizzo del web service basato su COGITO, funzioni di similarità, ranking e algoritmo di ranking fusion (*Sezione 2.2*).
- **Parte 2 - Tecniche di Ottimizzazione e Valutazione Sperimentale**
  - **Capitolo 3 - Ottimizzazione del progetto.** Presentazione generale del progetto e dello scenario utilizzato (*Sezione 3.1*); Descrizione di tutte le tecniche di ottimizzazione effettuate, dettagliando cosa è stato fatto a livello di: collezione di documenti e profili utente, ranking con il modello Vettoriale e con le classi IPTC e tecniche di ranking fusion (*Sezione 3.2*).
  - **Capitolo 4 - Valutazioni Sperimentali.** Panoramica generale sulla collezione di documenti e sui profili utente utilizzati (*Sezione 4.1*); Prove sperimentali e valutazioni sull'utilizzo dei sinonimi e/o dei termini correlati nel calcolo della similarità con il modello Vettoriale (*Sezione 4.2*); Prove sperimentali e valutazioni sull'utilizzo del valore di ICF nel calcolo della similarità per le classi IPTC (*Sezione 4.4*); Confronto e valutazione degli algoritmi di ranking fusion, basati sul rank e sullo score, utilizzati per fondere i due ranking derivanti dai

calcoli di similarità (*Sezione 4.5*); Prove sperimentali e valutazioni sull'utilizzo dei valori di soglia nella fase di ranking fusion (*Sezione 4.6*)

– **Capitolo 5 - Implementazione del software.** Come avviene l'estrazione dei dati (*Sezione 5.1*); Come viene creato il Glossario dei termini rilevanti (*Sezione 5.2*); Come viene creato l'Inverted Index (*Sezione 5.3*); Come avviene l'estrazione dei termini importanti dai Profili Utente (*Sezione 5.4*); Come vengono realizzate le funzioni di similarità (*Sezione 5.5*); Come vengono realizzati gli algoritmi di Ranking Fusion (*Sezione 5.6*).

Parte I

Stato dell'arte

Capitolo 1

Inquadramento Teorico

In questo capitolo verranno presentati tutti i concetti teorici che mi hanno permesso di comprendere a fondo il progetto AMBIT[1] alla base di questo scritto, progetto che sarà esposto più dettagliatamente nei capitoli successivi. In particolare, nelle prossime sezioni verrà descritto:

- Cos'è e cosa si intende per Contesto (*Sezione 1.1*).
- Cosa sono le applicazioni context-aware e quali sono le loro caratteristiche (*Sezione 1.2*).
- Cos'è l'Information Retrieval, quali obiettivi si pone e qual è il suo processo (*Sezione 1.3*).
- Perchè è utile utilizzare il contesto nel processo di Information Retrieval (*Sezione 1.4*).
- Piccola panoramica sui modelli di Information Retrieval (*Sezione 1.5*).
- Cos'è il modello Vettoriale, quali sono le sue proprietà e i suoi indici di valutazione (*Sezione 1.6*).
- Come viene effettuata la fase di ranking dei risultati (*Sezione 1.7*).

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

1.1 Contesto

Molti ricercatori hanno proposto varie classificazioni del termine "contesto"; da quando il termine fu introdotto da Schilit et al. nel 1994 [24], un gran numero di definizioni del termine "contesto" sono state proposte nel settore dell'informatica.

Schilit et al. affermano che i tre principali aspetti del contesto sono: "where you are" (dove ti trovi), "who you are with" (con chi sei) e "what resurces are nearby" (quali risorse ci sono nelle vicinanze).

Altre definizioni simili di contesto furono proposte da Ryan et al. [21] e Brown et al. [4]. Ryan et al. affermano che il contesto può essere definito tramite una serie di caratteristiche ambientali (environment), come ad esempio il luogo, la temperatura e l'identità dell'utente preso in considerazione. Brown et al. descrivono le diverse forme di contesto che possono essere utilizzate dai computer come: il luogo, la presenza di oggetti e persone, la temperatura, la pressione sanguigna o più in generale qualsiasi fattore ambientale che possa influenzare le attività su un computer, a condizione che possa essere percepito e misurato.

Nel 1999 Dey e Abowd [5] presentano punti di vista alternativi sul contesto, dandone una definizione più astratta e generale rispetto alle precedenti:

"Context is any information that can be used to characterize the situation of an entity. An entity is a person, place, or object that is considered relevant to the interaction between a user and an application, including the user and applications themselves."

In questa definizione non esiste più nessuna dipendenza da una serie di caratteristiche enumerate, ma si considera contesto ogni informazione che può essere utile per caratterizzare e descrivere uno scenario in cui si trova una particolare entità, dove per entità si intende un soggetto (persone, oggetti, luoghi) considerato rilevante per l'interazione tra l'utente che sottopone la richiesta e l'applicazione, dove le entità possono essere l'utente e/o l'applicazione stessa.

1.2 Context-Awareness

Il termine context-awareness significa letteralmente "consapevolezza del contesto", quindi quando si parla di applicazioni context-aware, ci si riferisce ad ap-

plicazioni che hanno consapevolezza del contesto in cui l'utente si trova e grazie a questo possono adattare il loro comportamento.

I primi a proporre una definizione di context-aware furono Schilit et al. nel 1994 [25], affermando che context-aware computing è l'abilità di una applicazione di rilevare e reagire ai cambiamenti dell'ambiente in cui è situata.

Più tardi, Hull et al. [11] e Pascoe et al. [19] definiscono il concetto di "context-awareness" più esplicitamente come la capacità dei dispositivi informatici di rilevare e dare un senso, interpretare e rispondere agli aspetti dell'environment locale di un utente e dei dispositivi stessi.

Nel 1999 Dey e Abowd [5] coniano una definizione più generale di context-aware, rispetto alla precedente, non dipendente da particolari tipi di contesto:

"A system is context-aware if it uses context to provide relevant information and/or services to the user, where relevancy depends on the user's task."

Essi affermano che un sistema si può definire context-aware quando sfrutta il contesto per fornire importati informazioni e/o servizi all'utente, dove l'importanza dipende dalla richiesta dell'utente.

Uno degli obiettivi dei sistemi context-aware è quindi quello di acquisire e utilizzare le informazioni sul contesto di un dispositivo, al fine di fornire servizi adeguati a particolari persone, luoghi, tempi, eventi, ecc...

1.2.1 Caratteristiche dei sistemi Context-Aware

Le caratteristiche delle applicazioni context-aware sono generalmente classificate in una delle seguenti categorie: presentazione di informazioni e servizi a un utente (Presentation), l'esecuzione automatica di un servizio (Auto Execution), e il tagging del contesto sulle informazioni necessarie per il successivo recupero.

- *Presentation.* Riferito ad applicazioni che semplicemente visualizzano informazioni contestuali o servizi agli utenti, ad esempio la presentazione all'utente di documenti che stava ricercando in base al contesto. Tuttavia, Dey et al. [5] sottolineano che è spesso difficile distinguere tra presentazione delle informazioni e presentazione di servizi.

- *Auto Execution.* Comprende le fasi di Context-Triggered Actions [24] e Contextual Adaption [19] ed è la capacità di eseguire o modificare automaticamente un servizio in base al contesto attuale.

- *Tagging.* Contrassegnare le informazioni contestuali per il successivo recupero (o Contextual Augmentation [19]), è la capacità di associare dati digitali al contesto dell'utente.

Alcune delle principali sfide in materia di context-aware computing sono[5]:

- lo sviluppo di una tassonomia e la rappresentazione uniforme dei tipi di contesto;
- le infrastrutture per promuovere la progettazione, la realizzazione e l'evoluzione di applicazioni context-aware;
- la scoperta di interessanti applicazioni context-aware che aiutino la nostra interazione quotidiana con i servizi computazionali onnipresenti;

1.3 Information Retrieval

L'Information Retrieval (IR), che significa letteralmente "reperimento delle informazioni", è l'insieme delle tecniche utilizzate per gestire la rappresentazione, la memorizzazione, l'organizzazione delle informazioni e l'accesso ad oggetti contenenti informazioni quali documenti, pagine web, cataloghi online e oggetti multimediali, al fine di rendere agevole all'utente il soddisfacimento dei propri bisogni informativi. I problemi centrali dell'Information Retrieval sono:

- capire quali documenti sono impotanti e quali non lo sono per l'utente; non è semplice caratterizzare esattamente i bisogni informativi dell'utente.
- le decisioni di solito dipendono da un algoritmo di ranking basato sul concetto di rilevanza del documento; questo concetto è determinato dal modello di Information Retrieval adottato.

Il termine Information Retrieval fu coniato da Calvin Mooers nel 1951 [17], che lo definì in questo modo:

"Information retrieval is the name for the process or method whereby a prospective user of information is able to convert his need for information into an actual list of citations to documents in storage containing information useful to him. It is the finding or discovery process with respect to stored information. It is another, more general, name for the production of a demand bibliography.

Information retrieval embraces the intellectual aspects of the description of information and its specification for search, and also whatever systems, technique, or machines that are employed to carry out the operation. Information retrieval is crucial to documentation and organization of knowledge"

In questa definizione Mooers spiega come l'Information Retrieval sia un processo con cui un utente può convertire il suo bisogno di informazioni in un elenco di documenti in memoria contenenti informazioni utili per lui. E' il processo di ritrovamento o di scoperta delle informazioni memorizzate. Information Retrieval abbraccia gli aspetti intellettuali della descrizione delle informazioni e le sue specifiche per la ricerca, e anche qualunque sistema, tecnica, o macchina che venga impiegata per effettuare questa operazione. Infine, Information Retrieval è fondamentale per la documentazione e l'organizzazione della conoscenza.

Dopo mezzo secolo di evoluzione, i sistemi e i processi di Information Retrieval sono diventati altamente sofisticati [22]. Tra i molti cambiamenti e miglioramenti, sicuramente il più significativo è quello che i sistemi di Information Retrieval ora prevedono un alto grado di interazione, con tutte le implicazioni e i problemi di interazione uomo-computer che lo accompagnano. Tuttavia, i problemi identificati dai Mooers sono ancora alla base dei sistemi di IR: *Come fornire ad un potenziale utente informazioni utili?* o in termini contemporanei: *come fornire agli utenti un efficace accesso e l'interazione con le informazioni, per consentire loro di utilizzarle efficacemente?*

Negli anni successivi, il termine Information Retrieval fu ripreso da più ricercatori. Per esempio Garfield nel 1957 [8], secondo quando detto da Mooers, descrisse il significato di Information Retireval, come trovare informazioni la cui ubicazione o esistenza è sconosciuta a priori.

Invece Manning nel 2008 [16] diede questa definizione: Information Retrieval (IR) è trovare materiale (di solito documenti) di natura non strutturata (di solito di testo) che soddisfa un bisogno di informazioni all'interno di grandi collezioni (solitamente memorizzati sui computer).

Infine Hersh nel 2009 [10]: Un campo all'intersezione tra scienza dell'informazione e informatica, IR si occupa dell'indicizzazione e del recupero delle informazioni da risorse informative eterogenee e soprattutto testuali.

1.3.1 Scopo e processo dell'Information Retrieval

Lo scopo di questa importante branca dell'informatica è quello di soddisfare il cosiddetto User Information Need, cioè il "bisogno informativo dell'utente", ovvero garantire a quest'ultimo, in seguito ad una sua ricerca, tutti quelli che sono i documenti e le informazioni rilevanti per quella che è stata la richiesta da egli effettuata.

Due concetti sono di fondamentale importanza per analizzare un sistema di Information Retrieval, query ed oggetto:

- Le **query**("interrogazioni") sono stringhe di parole-chiave che rappresentano l'informazione richiesta. Vengono digitate dall'utente in un sistema IR, come ad esempio un motore di ricerca, e rappresentano la concretizzazione di quello che è il reale bisogno informativo dell'utente.

- Un **oggetto** è un'entità che possiede informazioni che potrebbero essere la risposta all'interrogazione dell'utente. Un documento di testo ad esempio, è un oggetto di dati.

Un processo di Information Retrieval inizia quando un utente inserisce una query nel sistema; una query non identifica univocamente un singolo oggetto della collezione, invece alcuni oggetti potrebbero corrispondere alla query, con diversi gradi di rilevanza.

Le query dell'utente vengono poi confrontate con le informazioni del database.

La maggior parte dei sistemi di Information Retrieval calcolano un punteggio numerico per ciascun oggetto nel database corrispondente alla query, e classificano gli oggetti in base a questo valore; gli oggetti top ranking vengono infine mostrati all'utente.

Il processo può quindi essere ripetuto se l'utente desidera rifinire la query [7].

1.4 Utilità del Contesto nel processo di Information Retrieval

Tipicamente, i sistemi di Information Retrieval supportano le loro decisioni unicamente sulla collezione di query e documenti. Diversi fattori impliciti relativi all'utente e al contesto di ricerca, come tempo, luogo, attività, competenze, in-

terazione, vengono ignorati ma potrebbero essere considerati per ottimizzare le prestazioni nel recupero delle informazioni.

In realtà, tutte le attività di informazione hanno luogo in un contesto che riguarda il modo di accedere alle informazioni, interagire con un sistema di recupero, valutare e prendere decisioni circa i documenti recuperati [13] [9]. Una strategia contestualizzata potrebbe consentire ai sistemi di IR di: capire e prevedere di quali informazioni ha bisogno l'utente, capire come e quando queste debbano essere visualizzate, presentare i risultati collegandoli ad informazioni precedenti e compiti in cui l'utente è coinvolto ed inoltre decidere chi altro dovrebbe ottenerne di nuove[15].

Nel campo dell'Information Retrieval, vi è un crescente interesse per migliorare il processo di ricerca utilizzando le esigenze degli utenti (User Information Need) e il contesto [3]. Un primo modello che ha avvicinato IR al livello di contesto è quello descritto da Belkin nel 1980 [1]. Successivamente, altri autori hanno sviluppato modelli, come Ingwersen [12] e Saracevic [23] nei quali il contesto è posto al centro del processo di Information Retrieval.

1.5 Modelli di Information Retrieval

I modelli classici di Information Retrieval sono tre: il modello Booleano, il modello Vettoriale e il modello Probabilistico [6].

- *Modello Booleano.* Il modello Booleano è un semplice modello di recupero basato sulla teoria degli insiemi e dell'algebra di Boole; in questo modello, i termini indice sono presenti oppure assenti in un documento. Come risultato, i pesi dei termini indice vengono considerati in modo binario, cioè possono assumere solamente i valori 0 o 1. La query q è composta da termini indice collegati da operatori booleani come "and", "or", "not". I principali vantaggi del modello Booleano sono la semplicità e il formalismo pulito che sta dietro a questo modello; mentre i principali svantaggi sono che la corrispondenza esatta (cioè un documento può essere pertinente oppure non pertinente, senza corrispondenze parziali) può portare al recupero di troppi o troppo pochi documenti e che i documenti recuperati non siano classificati in base alla rilevanza della query.

- *Modello Vettoriale.* Il modello Vettoriale riconosce che l'uso di pesi binari è troppo limitante e propone un quadro in cui la corrispondenza parziale è possibile e auspicabile. Ciò si ottiene assegnando pesi non binari ai termini indice nelle query e nei documenti. Questi pesi sono infine utilizzati per calcolare il grado di somiglianza tra ogni documento memorizzato nel sistema e la query dell'utente. Il modello Vettoriale ordina i documenti recuperati in ordine di similarità decrescente.

- *Modello Probabilistico.* L'idea fondamentale su cui si basa il modello Probabilistico è che, data una query utente, vi è un insieme di documenti che contiene esattamente i documenti rilevanti e nient'altro. Data la descrizione di questo insieme ideale, non avremmo problemi a recuperare i suoi documenti, quindi possiamo pensare al processo di interrogazione come ad un processo che specifichi le proprietà di questo insieme ideale. Data una query utente e un documento della collezione, il modello Probabilistico tenta di stimare la probabilità che l'utente troverà questo documento interessante (cioè rilevante).

1.6 Il modello Vettoriale, proprietà e indici di valutazione

Il modello Vettoriale è spesso utilizzato come modello di riferimento per tutte quelle applicazioni che lavorano sul contesto dell'utente (context-aware) ed è anche il modello utilizzato all'interno del progetto AMBIT. In questa sezione verrà quindi dettagliato cos'è il modello Vettoriale, come si misura la rilevanza di un documento rispetto alla query tramite il calcolo della similarità, come si misura l'importanza di un termine rispetto ad un documento o ad una collezione di documenti tramite il calcolo di Term Frequency e Inverse Document Frequency e come si valuta l'efficacia di un sistema di Information Retrieval tramite le misure di Recall e Precision.

1.6.1 Il modello Vettoriale

Il modello Vettoriale è di gran lunga il modello di Information Retrieval più comune, anche se piuttosto recente (1980).

Lo svantaggio dell'assegnamento di pesi binari nel modello Booleano è sorpassato nel modello Vettoriale, nel quale la corrispondenza parziale è possibile, e in cui vengono utilizzati pesi non binari per i termini indice delle query e dei documenti, e di conseguenza per il calcolo del grado di similarità.

Generalmente un utente inserisce la sua richiesta sotto forma di testo libero, a cui viene data risposta con una serie di documenti recuperati dal sistema di Information Retrieval in ordine decrescente rispetto al loro grado di similarità.

L'idea che sta alla base di questo modello è che i vari elementi di interesse sono modellati come elementi di uno spazio vettoriale; più specificamente termini, documenti, query, concetti, e così via, sono tutti rappresentati come vettori di grandi dimensioni in uno spazio vettoriale.

In particolare, la prima fase effettuata da un sistema di Information Retrieval è quella del pre-processamento testuale degli oggetti, procedura che trasforma un documento in un insieme di termini indice. Questo processo effettua le seguenti operazioni:

- *Analisi lessicale del testo.* Trattamento di cifre, trattini, segni di punteggiatura, maiuscole e minuscole.
- *Eliminazione di parole non significative.* Filtra le parole inutili ai fini del recupero dei documenti importanti.
- *Stemming e Lemmatization.* Trattamento delle variazioni sintattiche dei termini (rimozione di suffissi, plurali, ecc. . . , utilizzo della forma base delle parole).
- *Selezione dei termini indice.* Definizione dei termini da utilizzare come termini indice.
- *Costruzione di strutture per la categorizzazione dei termini.* Permettere l'espansione della query originale con termini correlati.

Dopo la fase di pre-processamento degli oggetti, si avrà un insieme di termini (t), nel quale per ogni termine (i) in ogni oggetto (j) sarà associato un peso w_{ij}, che varia tra 0 e 1. Tutti i termini associati ad un particolare oggetto, formano lo spazio vettoriale relativo a quell'oggetto. Quindi, il vettore della query $\vec{q}$ e il vettore del documento $\vec{d_j}$ sono definiti in questo modo:

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

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

Dato $T = \{t_1, ..., t_t\}$ l'insieme di tutti i termini indice, una collezione di N documenti può essere rappresentata con il modello Vettoriale da una matrice termine-documento.

Una entry della matrice corrisponde al peso (*weight*) di un termine nel documento. Una entry con peso zero (0) significa che il termine non ha significato nel documento o semplicemente non esiste in quel documento.

Il modello Vettoriale propone di valutare il grado di similarità del documento d_j riguardo alla query q, come la correlazione tra i vettori $\vec{d_j}$ e $\vec{q}$. Tale correlazione, chiamata similarità coseno, può essere misurata tramite il coseno dell'angolo α che c'è tra questi due vettori nella loro rappresentazione sullo spazio vettoriale:

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

La formula della similarità coseno, detta anche *prodotto interno normalizzato*, è la seguente:

$$sim(d_j, q) = \frac{\vec{d_j} \cdot \vec{q}}{|\vec{d_j}| \times |\vec{q}|} = \frac{\sum_{i=1}^{t} w_{ij} \times w_{iq}}{\sqrt{\sum_{i=1}^{t} w_{ij}^2} \cdot \sqrt{\sum_{i=1}^{t} w_{iq}^2}}$$

Utilizzando la misura di similarità tra la query e ogni documento, è possibile:

- Ordinare i documenti recuperati dal sistema in base alla presunta rilevanza calcolata;
- Applicare una certa soglia di similatirà in modo che la dimensione dell'insieme di documenti recuperati possa essere controllato;
- Trovare i k documenti vicini ad un certo valore di similarità.

1.6.2 Term Frequency, Inverse Document Frequency e TF-IDF

Finora, grazie al modello Booleano, abbiamo considerato i pesi dei termini in modo binario:

$$w_{ij} = \begin{cases} 0 & \textit{Il termine non è presente nel documento} \\ 1 & \textit{Il termine è presente nel documento} \end{cases}$$

In questo modo però, tutti i termini presenti nel documento vengono considerati ugualmente importanti.

Il peso associato a ciascun termine nel modello Vettoriale, è utilizzato per distinguere termini che sono probabilmente importanti per la rappresentazione logica di un particolare documento, da altri termini che probabilmente sono meno importanti. Pesi alti vengono assegnati a parole di rilievo in un particolare documento.

Per associare un peso ad ogni termine, si utilizzano i concetti di Term Frequency e Inverse Document Frequency descritti di seguito.

Term Frequency

Term Frequency (TF), è la frequenza di occorrenza dei termini nei testi in linguaggio naturale ed è legata all'importanza di questi termini ai fini della rappresentazione dei contenuti; misura cioè quanto bene questo termine descrive il contenuto del documento. Ci sono tre possibili alternative per calcolare la frequenza f_{ij}:

- *Frequenza grezza.* Calcola il numero di occorrenze del termine i nel documento j:

$$f_{ij} = freq_{ij}$$

- *Frequenza normalizzata.* Calcola la frequenza normalizzata delle occorrenze del termine i nel documento j, attraverso questa formula:

$$f_{ij} = \frac{freq_{ij}}{max_l freq_{lj}}$$

- *Frequenza logaritmica.* Calcola il peso della frequenza logaritmica del termine i nel documento j, attraverso questa formula:

$$f_{ij} = \begin{cases} 1 + log_{10} freq_{ij}, & se freq_{ij} > 0 \\ 0, & altrimenti \end{cases}$$

Inverse Document Frequency

In una collezione di documenti, i termini che occorrono più raramente sono molto più informativi rispetto ai termini più frequenti, che occorrono cioè in molti documenti della collezione. Utilizziamo il concetto di **Document Frequency** (df) per catturare il numero di documenti della collezione in cui compare il termine t. Considerando i termini di una query utente rari nella collezione, un documento contenente tali termini è più probabile che sia rilevante rispetto ad un documento che non li contiene, ma non è un indicatore di rilevanza sicuro. E' stato quindi definito **Inverse Document Frequency** (IDF) del termine t in N documenti:

$$idf_t = log_{10} \left(\frac{N}{df_t} \right)$$

In questo modo l'IDF di un termine raro è alto, mentre l'IDF di un termine frequente è probabilmente basso. Nella formula viene utilizzato il logaritmo per smorzare l'effetto di IDF. IDF non ha effetto sul ranking di query con un solo termine, ma ha effetto sul ranking solo con query di almeno due termini.

TF - IDF

Combinando Term Frequency e Inverse Document Frequency è possibile ottenere il peso $w_{t,d}$ di ogni termine t nel documento d della collezione.

Questa combinazione non è altro che la moltiplicazione di TF e IDF:

$$w_{t,d} = tf_{t,d} \times log_{10} \left(\frac{N}{df_t} \right)$$

Si tratta del miglior schema di ponderazione conosciuto nell'ambito dell'Information Retrieval, e grazie alle definizioni di TF e IDF, $w_{t,d}$ avrà le seguenti caratteristiche:

- Aumenterà con il numero di occorrenze del termine nel documento
- Aumenterà in base alla rarità del termine nella collezione di documenti

In questo modo, termini molto comuni presenti in un numero elevato di documenti hanno un peso inferiore e daranno un minor contributo alla similarità finale, dal momento che sono probabilmente meno significativi.

1.6.3 Recall e Precision

Per misurare l'efficacia del sistema di Information Retrieval in modo standard, abbiamo bisogno di una collezione di test costituita da:

- Una collezione di documenti;
- Una suite di test di "User Information Need", esprimibili come query;
- Una valutazione, di solito binaria, di rilevanza o non-rilevanza, per ogni query e ogni documento;

L'approccio standard per la valutazione di un sistema di Information Retrieval, ruota intorno al concetto di documenti pertinenti e non pertinenti.

La rilevanza viene valutata rispetto allo User Information Need, non rispetto alla query; un documento è rilevante se si risponde alla necessità di informazione dichiarata dall'utente, non perché contiene tutte le parole della query.

Le due misure più frequenti e di base per misurare l'efficacia di un sistema di Information Retrieval sono **Recall** e **Precision**. Precision può essere vista come una misura di esattezza o fedeltà, mentre Recall è una misura di completezza.

Data una richiesta di informazioni, dobbiamo conoscere:

- **R**: il set di documenti rilevanti
- **A**: il set di risposte, cioè i documenti che sono stati effettivamente recuperati dal sistema di IR
- $|Ra|$: il numero dei documenti nell'intersezione tra A ed R

Possiamo quindi definire Recall e Precision in questo modo:

- **Recall**. Capacità di trovare tutti i documenti rilevanti nella collezione. In termini tecnici si tratta della frazione dei documenti rilevanti (l'insieme R) che sono stati recuperati, calcolata attraverso questa formula:

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

- **Precision**. Capacità di trovare nei primi posti della lista ordinata di documenti, quelli altamente rilevanti. In termini tecnici si tratta della frazione dei documenti recuperati (l'insieme A) che sono rilevanti, calcolata attraverso questa formula:

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

Recall e Precision sono due grandezze inversamente proporzionali. In un buon sistema, Precision decresce se aumenta Recall e il numero di documenti recuperati. Possiamo avere una alta Recall, ma un basso valore di Precision, se recuperiamo tutti i documenti per tutte le query.

Nel ranking finale quindi, più si scende, più aumenta il valore di Recall e diminuisce il valore di Precision.

1.7 Ranking

Nella maggior parte dei modelli di Information Retrieval esistenti, il problema del recupero è trattato utilizzando una singola query e un insieme di documenti. Da una singola query, tuttavia, il sistema di recupero può avere solo un indizio molto limitato circa il bisogno di informazioni dell'utente. Un sistema di recupero ottimale dovrebbe quindi cercare di sfruttare tutte le possibili informazioni aggiuntive relative al contesto dell'utente per migliorare la precisione del recupero. Infatti, il recupero sensibile al contesto è stato identificato come una delle principali sfide per la ricerca in IR.

Ci sono due tipi di informazioni sul contesto che possiamo utilizzare. Uno è il contesto a breve termine, che è l'informazione immediatamente circostante, e mette in luce le informazioni di cui un utente ha bisogno in una singola sessione (una sessione può essere considerata come un periodo costituito da tutte le interazioni per la stessa richiesta di informazioni). La categoria delle informazioni necessarie all'utente (ad esempio, bambini o sport), le query precedenti, e i documenti visualizzati di recente, sono tutti esempi di **contesto a breve termine**. Tali informazioni sono direttamente collegate all'attuale necessità di informazioni dell'utente e dovrebbero quindi essere utili per migliorare la ricerca corrente.

L'altro tipo di contesto è il **contesto a lungo termine**, che si riferisce a informazioni come il livello di istruzione di un utente e i suoi interessi generali,

la cronologia delle query dell'utente accumulate nel tempo e le informazioni passate dei clic effettuati; tali informazioni sono generalmente stabili per un lungo periodo di tempo. Il contesto a lungo termine può essere applicabile a tutte le sessioni, ma non può essere efficace quanto il contesto a breve termine per migliorare la precisione della ricerca di una particolare sessione [26].

Nella fase di ranking dei risultati, possono essere quindi utilizzate informazioni sul contesto dell'utente per migliorare i risultati della ricerca e che portano ad utilizzare un sistema più efficiente ed efficace nella soddisfazione dell'User Information Need.

Si crea quindi un profilo utilizzando anche il contesto dell'utente stesso; il profilo viene confrontato con la collezione dei documenti calcolando la misura di similarità tra i documenti e la query utente (che in questo modo si riferisce a quanto richiesto dall'utente più le informazioni riguardanti il suo contesto), con le tecniche descritte precedentemente. Infine, il risultato viene ordinato in ordine descrescente rispetto al grado di similarità e mostrato all'utente; ne risulta un ranking che soddisfa l'User Information Need per quella specifica sessione utente.

Capitolo 2

AMBIT

In questo capitolo verrà presentato il progetto AMBIT su cui si basa questo scritto e che ho sviluppato estendendo alcuni moduli (questi moduli verranno presentati nei capitoli successivi), cos'è, quali sono i suoi scopi, e una panoramica sulle sue funzionalità applicative.

In particolare, nelle prossime sezioni verrà descritto:

- Cos'è il progetto AMBIT, quali obiettivi si pone e quali sono le applicazioni previste dal progetto (*Sezione 2.1*).
- Quali sono e come sono sviluppate le varie funzionalità applicative: profili utente, collezione di documenti, glossari e inverted index, utilizzo del web service basato su COGITO, funzioni di similarità, ranking e algoritmo di ranking fusion (*Sezione 2.2*).

2.1 Il progetto AMBIT

Il progetto **AMBIT**, ovvero "Algorithms and Models for Building context-dependent Information delivery Tools", è un progetto di ricerca che mira a fornire un modello generale di contesto così come una piattaforma che può essere sfruttata per creare e distribuire diversi tipi di applicazioni e sistemi dipendenti dal contesto.

Questo progetto punta a superare le limitazioni degli approcci esistenti, che sono dovuti principalmente alla limitata nozione di contesto che propongono (sempre che la nozione di contesto sia presente). In particolare, viene sottolineato il fatto che le tecnologie attuali non considerano con precisione il concetto di

contesto e profilo utente, principale fonte di dati inutili che sovraccaricano i sistemi e spesso le menti degli utenti.

Gli obiettivi del progetto mirano a contribuire ad un significativo avanzamento delle conoscenze nel campo dei servizi dipendenti dal contesto (context-dependent), attraverso lo studio e la realizzazione di risultati innovativi lungo le seguenti direttrici:

- Definizione di **modelli generici** e **tecniche per la rappresentazione** dei contesti; questi verranno poi sfruttati da sistemi e applicazioni in grado di fornire informazioni e/o servizi dipendenti dai dati del contesto. I dati del contesto di un utente che si possono estrapolare potrebbero essere, per esempio:
  - la posizione geografica
  - il tempo cronologico, come data e ora
  - le stagioni
  - il tempo meteorologico
  - il profilo utente
  - la cronologia delle pagine visitate da un utente
  - ecc...
- Individuazione di supporti per la **memorizzazione dei dati** (database relazionali, graph database, strutture dati ad-hoc, ecc...) e **algoritmi** adatti alla gestione efficiente dei contesti (accesso, aggiornamento dinamico, apprendimento, ecc...);
- Progettazione di un'**architettura** software generale e una **piattaforma** prototipo di supporto ad applicazioni client/server dipendenti dal contesto, e l'esecuzione lato client su dispositivi mobile;
- **Verticalizzazione** della piattaforma rispetto ad applicazioni di test individuate;

Il progetto si pone anche come scopo di avere diversi risultati pratici, come per esempio migliorare la competitività dei partner operativi, che può essere raggiunta (soprattutto nel settore informatico) solo producendo costantemente

innovazione. Per le società coinvolte, un primo risultato sarà misurato in termini di un maggiore know-how. Le attività di AMBIT permetteranno ai partner di acquisire competenze in tecnologie software che difficilmente sarebbero state sviluppate in un contesto isolato.

Il progetto ha inoltre obiettivi interessanti anche a livello territoriale. Infatti, i servizi context-dependent offerti da associazioni di comuni e/o particolari categorie di imprese (ad esempio, i consorzi per l'Aceto Balsamico Tradizionale di Modena, o per il Parmigiano Reggiano), possono contribuire a migliorare l'offerta e ad incrementare il volume degli affari per mezzo di notifiche agli utenti di punti vendita di interesse, percorsi culturali e fiere gastronomiche, mercatini periodici o itineranti, ecc...

2.1.1 Applicazioni previste dal progetto

Il progetto dovrà individuare e definire diversi campi di applicazione, con il massimo livello di dettaglio possibile, fornendo importanti casi studio in questi campi. In questo modo, gli scenari immaginati contribuiranno alla identificazione di un modello generale per la rappresentazione di informazioni contestuali che potrà essere utilizzato per migliorare gli attuali servizi forniti agli utenti.

Le applicazioni (o i casi studio) sono identificati dai partner operativi nei rispettivi settori di interesse, fornendo così informazioni preziose e dettagliate per l'implementazione della piattaforma. Alcuni contesti applicativi e casi studio che sembrano adatti per l'analisi e il controllo della piattaforma sono:

- **Pubblicità context-aware.** Fornire pubblicità su misura è un'applicazione ben definita che si nasconde in molti siti e portali online, nella presentazione di siti sponsorizzati da motori di ricerca, ecc... Una personalizzazione più efficace può essere fatta nel caso in cui sia disponibile il profilo di un utente, ottenuto per esempio a partire dalla storia degli accessi registrati dai motori di ricerca o altri siti che forniscono servizi online. L'introduzione della dipendenza dal contesto permette di fornire un bersaglio di messaggi promozionali più preciso.

- **Valorizzazione culturale del territorio (ebook e applicazioni proattive).** Gli ebooks stanno diventando sempre più comuni come strumento di lettura di libri e, in generale, per l'accesso a informazioni in varie situazioni; la possibilità di avere ebooks dipendenti dal contesto, come le guide

intelligenti, in grado di fornire diversi modi di lettura che, adattandosi al contesto del lettore, possono rendere la lettura più coinvolgente, vale a dire adattando i contenuti per renderli più efficaci, e fornendo informazioni di interesse per l'utente. Le applicazioni proattive sono in grado di monitorare gli interessi degli utenti e contattarli per suggerire eventi o attività che potrebbero non essere notati o esaminati. La creazione di ebooks e applicazioni proattive permette agli utenti di essere più coinvolti nella conoscenza del loro territorio e delle sue iniziative.

- **Soluzioni intelligenti per l'help-desk.** Una delle comunicazioni più frequenti tra una società (o ente pubblico) e i suoi clienti è certamente l'help-desk. Tenendo conto del contesto in cui viene posta la domanda (comprese le comunicazioni precedenti della società con il cliente, i prodotti/servizi utilizzati, la sua cronologia passata di navigazione sulle pagine di supporto, ecc...), un servizio di help-desk può evolvere in un sistema intelligente e fornire automaticamente risposte più affidabili e mirate. Un sistema di help-desk intelligente avrà anche la capacità di auto-apprendimento basato sulla presenza di domande/risposte in un ambiente multi-utente, e può configurarsi distinguendo tra i diversi tipi di utenti, ad esempio, tra utenti privati e aziendali.

2.2 Funzionalità applicative

In questa sezione verranno presentate tutte le funzionalità applicative del progetto AMBIT, dall'estrazione dei dati dalla collezione di documenti e dai profili utente, alla costruzione dei glossari e degli inverted index, l'utilizzo della similarità per calcolare quali documenti siano più importanti di altri e da questo estrarre un ranking, ed infine le tecniche di ranking fusion utilizzate.

2.2.1 Profili utente e collezione di documenti

Per il funzionamento dell'applicativo, in prima battuta servono due insiemi specifici:

- **Collezione di documenti.** Una serie di documenti, pagine web, ecc... che compongono lo scenario e che verranno confrontati con il profilo e le richie-

ste dell'utente per capire se uno specifico documento possa fare parte del ranking e con quanto peso.

- **Profili Utente.** Vengono definiti una serie di profili dipendenti dal contesto, che contengono la richiesta dell'utente sottoposta all'applicativo (se l'applicazione sviluppata lo consente, come in un sistema di help-desk), oppure capirne i gusti o più semplicemente verificarne le pagine visualizzate in passato, attraverso per esempio la sua cronologia di navigazione.

Questi due oggetti sono la base dell'applicativo AMBIT, da cui avranno inizio tutte le fasi successive che porteranno alla costruzione di un profilo utilizzato all'interno delle funzioni di similarità per poter presentare un ranking finale coerente con il profilo e il contesto dell'utente in questione.

2.2.2 Glossari e Inverted Index

Data una collezione di documenti, la prima fase effettuata dal software AMBIT è quella del pre-processamento testuale degli oggetti, in modo da trasformare un documento in un insieme di termini indice. A questi termini indice verrà poi associato il valore di Term Frequency e Inverse Document Frequency[1], che esprime l'importanza di ciascun termine all'interno dei documenti analizzati, ed infine verranno raccolti tutti in un glossario.

L'operazione di pre-processamento testuale degli oggetti, da cui vengono ricavati una serie di termini indice, viene effettuata anche per il profilo utente. A questi termini però viene associato un solo peso, corrispondente inizialmente al valore di Term Frequency e che esprime l'importanza del termine nel profilo, ma che può essere modificato per dare più peso ad alcuni termini rispetto ad altri.

Dopo aver costruito il glossario, è possibile proseguire con la costruzione di un inverted index, cioè una struttura dati che per ogni termine del glossario, indica in quali documenti questo termine è contenuto. L'inverted index è un componente centrale che ha l'obiettivo di ottimizzare l'elaborazione, trovando velocemente tutti i documenti in cui è presente il termine X.

In questa fase vengono estratti anche i termini collegati alle parole all'interno del glossario in base a legami di sinonimia e correlazione; i legami di sinonimia vengono scoperti utilizzando i "synset" della libreria "WordNet", mentre

[1] vedi sottosezione 1.6.2

per i legami di correlazione si naviga l'albero degli iponimi e degli iperonimi di "WordNet".

Sinonimi e termini correlati sono importanti nella fase di ranking, perchè permettono di avere un ranking migliore e più preciso, rispetto all'elaborazione dei soli termini uguali.

2.2.3 COGITO

Nel software AMBIT, oltre alla classificazione classica dei documenti attraverso il calcolo di TF-IDF e della similarità, viene effettuata anche un'altra classificazione attraverso un web service basato sul software **COGITO**, una tecnologia semantica della ditta Expert System di Modena.

Questo software estrae tantissime informazioni e categorizzazioni differenti del testo, ma si è scelto di utilizzare la categorizzazione per classi IPTC.

Le classi IPTC sono categorie standard, strutturare ad albero, messe a punto dall'International Press Telecommunications Council (abbreviato in IPTC) ed utilizzate da tutte le principali agenzie di stampa e aziende editoriali del mondo, per la descrizione e classificazione di notizie, testi, foto, grafica, video e altri media.

Passando i documenti della collezione e quelli del profilo utente al web service, vengono estratte e restituite le classi IPTC che caratterizzano ogni documento. Le categorie sono elencate insieme ad un punteggio che indica la rilevanza di ciascuna categoria rispetto al documento nel suo complesso.

Una volta categorizzati tutti i documenti, attraverso il calcolo di un valore di similarità apposito, sarà possibile estrarre un altro ranking basato proprio sull'utilizzo delle classi IPTC e della similarità tra le classi del profilo e quelle di ogni documento della collezione.

2.2.4 Funzioni di similarità

La necessità di un efficiente ed efficace calcolo di similarità è fondamentale in contesti come quello del progetto AMBIT. In questa sezione quindi verranno presentate le diverse funzioni utilizzate dal software AMBIT per calcolare il livello di similarità, e di conseguenza l'importanza dei documenti della collezione rispetto al profilo utente, attraverso l'utilizzo del modello Vettoriale e delle classi IPTC.

Dal calcolo della similarità verrà infine estratto un ranking dei documenti più importanti per l'utente.

Similarità attraverso il modello Vettoriale

Il calcolo della similarità con il modello Vettoriale[2] nel software AMBIT, è stato ripreso dall'algoritmo di similarità descritto nell'articolo di Bergamaschi et al.[2].

La formula di similarità $TextSim(U, D)$ quantifica, dato un profilo utente U ed un documento di destinazione D, la somiglianza del documento di origine rispetto al documento di destinazione sulla base dei loro termini associati $t^U \in U$ e $t^D \in D$.

In particolare, il calcolo di $TextSim$ tra un determinato profilo U e ogni possibile documento D (cioè, ogni documento disponibile nell'index) prevede le seguenti fasi:

1. Considerare ogni termine U e trovare il termine più simile o i termini a disposizione in D, sfruttando una formula di similarità del termine $TSim$. Calcolare $TSim$ significa individuare:
   - Termini uguali;
   - Termini sinonimi (se non sono stati trovati termini uguali);
   - Termini correlati semanticamente (se non sono stati trovati termini sinonimi);
2. Indurre un ranking dei documenti a disposizione (sulla base di $TextSim$), prevedendo in tal modo quali sono i documenti importanti e quali non lo sono rispetto a U.

In dettaglio, la formula di similarità tra un documento sorgente U e un documento finale D è dato dalla somma di tutte le similarità tra ogni termine in U e ogni termine in D, massimizzando la similarità con il termine in D:

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

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

[2]vedi sottosezione 1.6.1

dove $w_i^U = tf_i^U \cdot idf_i$ e $w_{\bar{j}(i)}^D = tf_{\bar{j}(i)}^D \cdot idf_{\bar{j}(i)}$. In questo modo, ogni termine contribuisce alla similarità finale con un peso diverso, cioè termini più frequenti e più significativi contribuiscono maggiormente alla similarità tra i due documenti.

Le funzione $Tsim$ considera sinonimi (quindi, implicitamente termini uguali) e termini semanticamente correlati e viene definita in questo modo:

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

Più in particolare, il caso di massima somiglianza (valore 1) corrisponde al caso in cui i due termini sono uguali oppure sinonimi (relazione SYN). Inoltre, la formula prevede un ulteriore caso in cui i due termini non sono uguali o sinonimi, ma sono in qualche modo correlati tra loro da un punto di vista semantico: questi termini contribuiranno con una somiglianza di valore r, dove $0 < r < 1$ è un valore fisso definito dall'utente. Nel caso invece i due termini non siano uguali, sinonimi o correlati, allora la loro somiglianza avrà valore 0.

L'utilizzo di sinonimi e/o termini correlati può essere deciso dal programmatore in quanto tutte queste informazioni sono memorizzate all'interno dell'inverted index. Grazie a questo potranno essere sfruttate più funzioni di similarità, in base alle necessità:

1. Funzione di similarità che utilizza soltanto i valori di Term Frequency e Inverse Document Frequency[3];

2. Funzione di similarità che sfutta anche il peso di eventuali termini sinonimi;

3. Funzione di similarità che sfrutta anche il peso di eventuali termini sinonimi e correlati.

Similarità attraverso le classi IPTC

Sfruttando, per una ulteriore categorizzazione dei documenti, anche il web service basato sul software COGITO, non è possibile utilizzare la formula di similarità appena vista, ma bisogna utilizzare un'ulteriore formula di similarità che sfrutti le classi IPTC e il relativo score, estratti da ogni singolo documento della collezione e dal profilo utente.

[3]vedi sottosezione 1.6.2

Prima di arrivare al calcolo della similarità, il software effettua altre due fasi che permettono di elaborare i dati e prepararli per la fase successiva:

1. Estrazione per ogni documento della collezione e per ogni documento contenuto nel profilo utente, delle classi IPTC e dei relativi score, per mezzo del software COGITO.

2. Unione delle classi IPTC uguali tra i documenti che formano il profilo utente e somma degli score per ogni classe; in questo modo per ogni profilo utente non si avranno classi duplicate e ci saranno alcune classi con uno score maggiore, indicanti le classi prevalenti di quel profilo utente.

La funzione di similarità utilizzata dal software AMBIT con le classi IPTC, è ispirata dalla funzione di similarità esposta da Leacock et al.[14], la quale sfrutta le relazioni tra i termini provenienti da WordNet. Infatti, in WordNet i termini sono associati ad uno o più differenti significati (o sensi) e ciascuno di essi è collegato a significati di altri termini attraverso le relazioni tra gli iperonimi.

La formula di similarità $ClassSim(U, D)$ quantifica, dato un profilo utente U ed un documento di destinazione D, nella stessa filosofia di $TextSim(U, D)$, la somiglianza del documento di origine rispetto al documento di destinazione, in questo caso sulla base delle loro classi IPTC associate $c^U \in U$ e $c^D \in D$.

$$ClassSim(U, D) = \sum_{c_i \in U} CSim(c_i, c_{\bar{j}(i)}) \cdot s(c_i) \cdot s(c_{\bar{j}(i)})$$

$$c_{\bar{j}(i)} = argmax_{c_j \in D}(CSim(c_i, c_j))$$

$$CSim(c_i, c_j) = \begin{cases} -ln\frac{len(c_i,c_j)}{2 \cdot H}, & se\ len(c_i, c_j) < Th \\ 0, & altrimenti \end{cases}$$

La similarità $CSim$ tra le classi c_i e c_j deriva dalle metriche degli iperonimi sfruttati per la similarità dei termini; è calcolata come uno score che è inversamente proporzionale alla lunghezza del percorso più breve che collega le due classi nella gerarchia IPTC; nel caso in cui la lunghezza del percorso superi una soglia configurabile Th, la similarità è nulla. H è una costante che rappresenta la profondità massima della gerarchia, che per IPTC è 5.

2.2.5 Ranking e Ranking fusion

Una volta calcolati tutti i valori di similarità per ogni documento della collezione, i valori vengono ordinati in modo decrescente e restituiti sotto forma di ranking. I documenti con un valore maggiore di similarità si troveranno nelle prime posizioni e saranno i documenti più importanti; più si scende nel ranking, più l'importanza del documento al fine del recupero sarà minore.

Nel caso del software AMBIT avremo due differenti ranking, uno derivante dall'utilizzo del modello Vettoriale, l'altro derivante dalla classificazione dei documenti attraverso le classi IPTC[4].

Questi due ranking, proprio per la loro natura e per come sono stati costruiti, catturano diverse informazioni:

- il ranking derivante dal modello Vettoriale cattura informazioni legate ai singoli termini contenuti nei documenti analizzati, quindi ad informazioni più specifiche, come sostantivi, nomi propri, ecc ...
- il ranking derivante dalle classi IPTC cattura invece informazioni di natura più generale, in quanto categorizza i documenti in varie classi generiche; ci informa cioè sull'argomento di cui tratta il documento.

Per poter usufruire dei benefici legati all'uno e all'altro ranking, c'è la necessità di utilizzare un algoritmo che capisca se esistono, e in quali situazioni, un ranking restituisce risultati migliori rispetto all'altro e costruisca un ranking finale tenendo conto di tutte queste considerazioni, partendo proprio dai due ranking analizzati ed effettuando una fusione.

Proprio per questa attività di unione, o meglio di fusione, dei due ranking iniziali, l'algoritmo viene definito algoritmo di **Ranking Fusion** e in letteratura sono presenti tantissimi esempi di algoritmi che hanno questo scopo.

I ranking τ_{text}, τ_{class} indotti dalle equazioni $TextSim(U, D)$ e $ClassSim(U, D)$ rispettivamente, sui documenti D dato un profilo U, sono eventualmente fusi in un ranking finale $\hat{\tau}$ attraverso un metodo di combinazione lineare[20], sfruttando sia i termini sia i contributi delle classi:

$$s^{\hat{\tau}}(D) = \sum_{\tau \in \{\tau_{text}, \tau_{class}\}} \alpha_\tau (1 - \frac{\tau(D)-1}{|\tau|})$$

dove $|\tau|$ è la lunghezza del ranking e $\alpha_\tau >= 0$ è un peso di preferenza (il valore predefinito è 1 per entrambi i ranking).

[4] vedi sottosezione 2.2.4

I documenti che fanno parte di entrambi i ranking appariranno sicuramente nel ranking finale, perchè il termine $\tau(D)$ indica la posizione del documento D nel ranking considerato.

Parte II

Tecniche di Ottimizzazione e Valutazione Sperimentale

Capitolo 3

Ottimizzazione del progetto

In questo capito verrà presentato il progetto che è stato realizzato, descrivendo e analizzando in dettaglio le idee, le scelte e le ottimizzazioni effettuate, in cui si sono toccati un pò tutti gli aspetti generali del software AMBIT[1].

In particolare, il contenuto di questo capitolo si divide nelle seguenti sezioni:

- Presentazione generale del progetto e dello scenario utilizzato (*Sezione 3.1*).
- Descrizione di tutte le tecniche di ottimizzazione effettuate, dettagliando cosa è stato fatto a livello di: collezione di documenti e profili utente, ranking con il modello Vettoriale e con le classi IPTC e tecniche di ranking fusion (*Sezione 3.2*).

3.1 Presentazione del progetto

L'applicazione realizzata in questa Tesi, implementa un motore di ricerca semantico basato sul contesto. Il progetto prevede l'implementazione di soluzioni intelligenti per l'help-desk, che è la forma più utilizzata per la comunicazione tra una società e i propri clienti[2].

Il sistema di help-desk sarà fornito di una serie di documenti che permettano all'utente, in base alla propria richiesta e al proprio bisogno di informazioni, di trovare il documento che più lo soddisfa senza particolari sforzi, in modo affidabile e mirato.

[1] Algorithms and Models for Building context-dependent Information delivery Tools
[2] vedi sottosezione 2.1.1

Il sistema preso in considerazione è il sistema di help-desk di una società elettronica, che vende, ripara e fornisce assistenza sui propri prodotti come televisioni, cellulari, home theatre, tablet, computer, ecc... I documenti presenti all'interno di questo sistema potrebbero essere:

- manuali utente;
- guide per la risoluzione di problemi;
- documenti per l'installazione di particolari dispositivi;
- articoli specifici sulle funzionalità di un tipo di dispositivo;
- istruzioni per l'utilizzo di funzionalità del servizio di help-desk;
- documentazione generale come garanzie e informazioni legali;
- ecc...

Lo scopo di questo sistema di help-desk è quello di consigliare a teorici utenti quali pagine navigare, perchè sono quelle che più si avvicinano a quanto richiesto, in base ad un ordinamento delle pagine secondo il grado di similarità rispetto alle preferenze di ciascun utente.

Le preferenze dell'utente si compongono di una richiesta di informazioni a cui vengono aggiunti i dati sul suo contesto, come per esempio vengono elaborate anche le informazioni relative alla cronologia di navigazione passata sulle pagine di supporto. Queste preferenze vengono infine raggruppate in quello che viene chiamato **Profilo Utente** e passato alle funzioni di similarità, per il quale verrà calcolato il grado di rilevanza rispetto ad ogni documento della collezione.

L'applicazione è stata sviluppata partendo da una base, quella del software AMBIT, ed è stata modificata ad-hoc in modo da adattarsi agli obiettivi del progetto e alle conseguenti prove sperimentali.

3.2 Tecniche di Ottimizzazione

In questa sezione verranno dettagliate quali tecniche sono state utilizzate per effettuare ottimizzazioni sul software AMBIT, partendo dalla collezione di documenti utilizzata e da come è stato composto il profilo utente, passando per i ranking estratti sia con il modello Vettoriale che con le classi IPTC, ed arrivando alla descrizione delle tecniche di ranking fusion utilizzate.

3.2.1 Collezione di documenti e Profili utente

La collezione utilizzata è composta da molti documenti di testo provenienti da sistemi di assistenza e supporto di società operanti nei settori dell'elettronica e della comunicazione realmente esistenti. Si è cercato in questo modo di rendere più reali e veritieri gli sviluppi e le prove sperimentali effettuati sul progetto.

Insieme alla collezione di documenti sono stati creati alcuni profili utente che raccogliessero tutte le caratteristiche del progetto, in modo da avere una collezione di test abbastanza grande da coprire tutte o comunque la maggior parte delle casistiche riguardanti lo scenario utilizzato.

Come già accennato, il profilo utente è composto da due tipi di informazioni principali:

- *Richiesta.* Si tratta della richiesta che l'utente effettua al sistema di help-desk, cioè una stringa che rappresenta il bisogno informativo dell'utente in quel preciso momento, come quando si inserisce una stringa di ricerca in un motore di ricerca web.

- *Contesto.* Una serie di documenti, relativi alla collezione e quindi alla documentazione di supporto posseduta dal sistema di help-desk, che l'utente ha già navigato e visualizzato in quella sessione. In altre parole, si tratta della cronologia di navigazione passata dell'utente, sul sistema di supporto.

3.2.2 Ranking con modello Vettoriale

Con il modello Vettoriale[3], in cui vengono estratti i termini indice dalla collezione e dai profili utente, è stato incrementato il peso dei termini estratti dalla richiesta dell'utente, rispetto al resto dei termini derivanti dal suo contesto, in quanto si vuole dare più risalto alla domanda posta dall'utente al sistema di help-desk.

Sicuramente, quando un utente pone una domanda al sistema, si aspetta una risposta che rispecchi quanto da lui richiesto; proprio per questo si è voluto dare maggiore importanza ai termini della richiesta, mentre ai pesi dei termini riguardanti il contesto dell'utente non viene effettuato nessun aggiustamento e vengono normalmente utilizzati dalla funzione di similarità.

In questo modo, grazie all'utilizzo sia della richiesta dell'utente con un peso maggiorato, sia del contesto riguardante la cronologia di navigazione passata,

[3]vedi sottosezione 1.6.1

il sistema di help-desk potrà fornire automaticamente risposte più affidabili e mirate.

I dati derivanti dall'estrazione dei termini del profilo e di tutti i documenti della collezione, vengono passati alla funzione di similarità descritta della sezione 2.2.4, che restituirà per ogni documento, uno score che indica quanto quel documento è rilevante rispetto al profilo utente.

Questi score vengono infine ordinati in ordine decrescente, costruendo di fatto un ranking, dove nelle prime posizioni troveremo i documenti più rilevanti e l'importanza calerà mano a mano che si scende verso il fondo del ranking.

Il ranking non viene costruito con tutti i documenti presenti nella collezione, ma considerando soltanto i primi k documenti con score maggiore. Il parametro k è definibile a livello di progetto e in questo caso è stato impostato a 30.

3.2.3 Ranking con classi IPTC

In un sistema di help-desk, gli argomenti trattati sono limitati al settore in cui lavora la società proprietaria di questo servizio, sono cioè solitamente correlati tra loro (per esempio se sto utilizzando un help-desk di una società di dispositivi elettronici, non troverò sicuramente guide su viaggi e paesi del mondo).

Proprio per questo motivo, anche le classi estratte dal software COGITO[4], saranno più o meno correlate tra loro e soprattutto la maggior parte dei documenti conterrà le classi più argomentative del sistema di help-desk preso in considerazione. Nel nostro caso, cioè una società operante nel settore dell'elettronica e della comunicazione, le classi più ricorrenti nei documenti sono quelle comprese nella classe IPTC *"IPTC/Economy, Business and Finance/Computing and Information Technology"*: *Hardware, Software, Wireless Technology, Networking, Telecommunication Service, Telecommunication Equipment, ecc....*

Ci sono classi che sono molto frequenti nella collezione e che quindi perdono di importanza al fine del recupero, mentre altre classi, essendo meno frequenti, potrebbero caratterizzare un documento in modo più specifico.

Si è pensato così di utilizzare la stessa logica del calcolo TF-IDF[5] utilizzata nel modello Vettoriale, anche per le classi IPTC.

E' stato calcolato per ogni classe estratta dalla collezione, un valore di **Inverse Class Frequency** (ICF), definito in questo modo:

[4] vedi sottosezione 2.2.3
[5] vedi sottosezione 1.6.2

$$icf_c = \frac{N}{cf_c}$$

dove N è il numero totale di documenti che compongono la collezione, c è la classe presa in considerazione e cf_c rappresenta il numero di volte che la classe viene estratta dai documenti della collezione.

Questo valore è stato utilizzato nel calcolo della similarità per le classi IPTC, andandolo a moltiplicare per lo score della classe del profilo, nel caso in cui le classi del documento di origine e di quello di destinazione siano le stesse.

La formula di similarità per le classi IPTC[6] è stata quindi modificata nel seguente modo:

$$ClassSim(U, D) = \sum_{c_i \in U} CSim(c_i, c_{\bar{j}(i)})$$

$$c_{\bar{j}(i)} = argmax_{c_j \in D}(CSim(c_i, c_j))$$

$$CSim(c_i, c_j) = \begin{cases} -ln\frac{len(c_i,c_j)}{2 \cdot H} \cdot w(c_i), & se\ c_i = c_j \\ -ln\frac{len(c_i,c_j)}{2 \cdot H}, & altrimenti \end{cases}$$

La lunghezza del percorso più breve che collega le classi c_i e c_j nella gerarchia IPTC viene sempre calcolata ($-ln\frac{len(c_i,c_j)}{2 \cdot H}$), ma se la classe del profilo e la classe del documento che sto esaminando sono uguali, allora il valore di similarità viene incrementato moltiplicandolo per un termine $w(c_i)$ che rappresenta il peso di quella classe tra tutte quelle estratte.

Il termine $w(c_i)$ viene calcolato in questo modo:

$$w(c_i) = s(c_i) \cdot icf_c$$

dove $s(c_i)$ è lo score della classe c nell'i-esimo profilo utente, mentre icf_c è il valore di ICF della classe c.

3.2.4 Importanza del Ranking

Verificando i ranking estratti dal software AMBIT, si è cercato di capire come attribuirgli un valore di importanza, studiando gli score dei documenti recuperati.

L'idea di base è quella di considerare più informativo, e di conseguenza più importante, un ranking che ha documenti con un alto grado di similarità, cioè

[6] vedi sottosezione 2.2.4

con uno score maggiore. Seguendo questa logica, si è riusciti ad ottenere due valori rappresentanti l'importanza dei due ranking.

Per calcolare il grado di rilevanza dei due ranking, sono state effettuate le seguenti operazioni:

- Somma, per ogni ranking, degli score di tutti gli item:

$$Sum_{\tau_{text}} = \sum_{i \in \tau_{text}} s(i)$$

$$Sum_{\tau_{class}} = \sum_{j \in \tau_{class}} s(j)$$

- Normalizzazione dei due valori ottenuti con la fase precedente, in modo che la loro somma sia 1;

$$w_{\tau_{text}} = \frac{Sum_{\tau_{text}}}{Sum_{\tau_{text}} + Sum_{\tau_{class}}}$$

$$w_{\tau_{class}} = \frac{Sum_{\tau_{class}}}{Sum_{\tau_{text}} + Sum_{\tau_{class}}}$$

$$w_{\tau_{text}} + w_{\tau_{class}} = 1$$

Si ottengono così due valori $w_{\tau_{text}}$ e $w_{\tau_{class}}$, dove τ_{text} e τ_{class} sono rispettivamente il ranking calcolato con il modello Vettoriale e il ranking calcolato con le classi IPTC, che rappresentano l'importanza dei due ranking in base a quanto più informativi sono stati i documenti recuperati. Come si evince, questi due valori non sono fissati a priori, ma vengono attribuiti automaticamente in base al profilo utente che si sta elaborando in quel momento.

3.2.5 Ranking Fusion

Per effettuare al meglio la fase di ranking fusion, sono stati individuati alcuni fattori che permettano di capire e studiare quali siano le tecniche più efficaci per fondere due ranking:

- Capire quando due ranking hanno pesi differenti, cioè quando un ranking sta funzionando meglio dell'altro;

- Capire se uno dei due ranking è talmente prevalente in modo da poter scartare automaticamente l'altro dal ranking finale;

- Capire quali tecniche di ranking fusion sono le più efficaci ed efficienti, tra quelle che utilizzano lo score e quelle che utilizzano la posizione dell'oggetto nel ranking;

Tecniche di Ranking Fusion

Per capire quali tecniche di ranking fusion siano le più efficaci in base allo scenario utilizzato, ma più in generale nell'intero progetto AMBIT, si è scelto di studiare due algoritmi che si basano su tecniche completamente differenti.

Il primo algoritmo si basa sulla posizione degli oggetti nei ranking (rank), descritto nella sottosezione 2.2.5; il secondo algoritmo invece, si basa sullo score normalizzato degli oggetti nei ranking, e sfrutta comunque un metodo di combinazione lineare, come il primo.

Riprendendo la notazione del primo, il secondo algoritmo è descritto in questo modo:

$$s^{\hat{\tau}}(D) = \sum_{\tau \in \{\tau_{text}, \tau_{class}\}} \alpha_\tau \cdot \ w^\tau(D)$$

dove $0 <= \alpha_\tau <= 1$ è il peso assegnato al ranking, mentre $w^\tau(D)$ è lo score relativo al documento D in quel ranking. τ_{text} e τ_{class} sono i due ranking presi in considerazione, e che vengono eventualmente fusi in un ranking finale $\hat{\tau}$.

Utilizzando tutte e due le tecniche di ranking fusion nelle prove sperimentali effettuale, si è cercato di verificare se e perchè un metodo fosse migliore rispetto all'altro, e se l'utilizzo dei rank piuttosto che degli score porti a significativi miglioramenti in questa fase.

Ranking Fusion con Valori di Soglia

Con il calcolo dell'importanza dei due ranking sotto forma di valori normalizzati[7], è possibile capire quando un ranking sta funzionando meglio rispetto all'altro e di conseguenza escludere quello di minore importanza dal ranking finale.

L'idea quindi è di utilizzare un valore di soglia Th, in base al quale escludere un ranking sotto tale soglia, oppure includerlo e utilizzarlo nella fase di ranking fusion. Questo valore di soglia è calcolato sulla base dei valori di importanza che vengono attribuiti ai ranking.

Per la scelta del valore di soglia, si è deciso che se il peso di un ranking è maggiore del doppio rispetto al peso dell'altro ranking, allora il ranking con il

[7] vedi sottosezione 3.2.4

peso minore viene scartato e non parteciperà alla fase di ranking fusion. Si è scelto cioè, di non considerare i ranking ritenuti poco informativi, dove il termine "poco informativo" rispecchia un ranking che non raggiunge il livello di soglia prefissato.

Essendo *1* la somma dei due pesi, questo valore di soglia si riflette in modo pratico sul valore *0,7*. Un ranking con peso maggiore o uguale a *0,7* verrà restituito così com'è ed utilizzato nel ranking finale; viceversa, un ranking con peso minore o uguale a *0,3* verrà scartato automaticamente. Il valore di soglia è comunque definibile dal programmatore in base alle esigenze.

Capitolo 4

Valutazioni Sperimentali

In questo capitolo verranno presentati i risultati della valutazione e delle prove sperimentali effettuate all'interno del progetto AMBIT, per lo specifico scenario di help-desk utilizzato[1]. Varie analisi e valutazioni hanno interessato le funzionalità principali del software, verificando se e come queste tecniche influiscono sul risultato finale.

In particolare, il contenuto di questo capitolo si divide nelle seguenti sezioni:

- Panoramica generale sulla collezione di documenti e sui profili utente utilizzati (*Sezione 4.1*);
- Prove sperimentali e valutazioni sull'utilizzo dei sinonimi e/o dei termini correlati nel calcolo della similarità con il modello Vettoriale (*Sezione 4.2*);
- Prove sperimentali e valutazioni sull'utilizzo del valore di ICF nel calcolo della similarità per le classi IPTC (*Sezione 4.4*);
- Confronto e valutazione degli algoritmi di ranking fusion, basati sul rank e sullo score, utilizzati per fondere i due ranking derivanti dai calcoli di similarità (*Sezione 4.5*);
- Prove sperimentali e valutazioni sull'utilizzo dei valori di soglia nella fase di ranking fusion (*Sezione 4.6*).

[1] vedi sezione 3.1

Profili	Termini rilevanti estratti dalle richieste di ogni profilo	Classi IPTC prevalenti per ogni profilo
P1	reset, camera, setting, factory, reset camera	.../Hardware, .../Software
P2	radio	.../Software, .../Radio, .../Mass Media
P3	dvd, term, guarantee, dvd guarantee	.../Hardware, .../Wireless Technology
P4	problem, computer, mouse	.../Hardware, .../Wireless Technology
P5	signal	.../Television, .../Hardware
P6	sound, car, car speaker, speaker	.../Wireless Technology, .../Language
P7	signal, television	.../Television, .../Hardware
P8	phone, radio	.../Software, .../Radio, .../Mass Media
P9	print, doesnt, printer	.../Hardware, .../Computer Crime
P10	reset, account, password	.../Computer Sciences, .../Hardware, .../Computer Crime
P11	battery, charge, camera, camera battery	.../Hardware, .../Chemistry
P12	problem, network	.../Hardware, .../Wireless Technology
P13	downloaded, video	.../Cinema, .../Hardware
P14	credit, problem, payment problem, payment, card	.../Parliament, .../Computer Sciences, .../Credit and Debt

Figura 4.1: Termini rilevanti delle richieste e classi IPTC prevalenti di ogni profilo utente, estratti per la valutazione del software AMBIT.

4.1 Panoramica Generale

Per la sperimentazione e la valutazione del software AMBIT, sono stati raccolti una serie di 260 documenti derivanti da sistemi di help-desk e supporto agli utenti di compagnie operanti nei settori dell'elettronica e della comunicazione, come ad esempio la documentazione dei prodotti della ditta "Sony". Sono poi stati creati 14 profili utente che si riferiscono a questa collezione; ogni profilo è composto da un breve testo contenente la richiesta per il sistema e dal contesto utente riguardante la cronologia di navigazione passata sul sistema di help-desk.

Ogni profilo è stato sottoposto al software AMBIT, che tramite l'analisi del testo, le classificazioni e il calcolo della similarità già descritti nei capitoli precedenti[2], ha generato un ranking contenente una serie di possibili "suggerimenti", cioè puntatori ai documenti rilevanti nella collezione. Per valutare l'efficacia di questo approccio, per ogni profilo sono stati selezionati manualmente i documenti rilevanti nella collezione e sono stati confrontati con quanto restituito dal software.

Nella Figura 4.1 sono rappresentati sia i termini più rilevanti che saranno utilizzati dal modello Vettoriale[3], cioè i termini relativi alla richiesta dell'utente al sistema di help-desk (lasciando fuori per il momento il contesto dell'utente, che dettaglieremo successivamente), sia alcune delle classi IPTC (quelle aventi

[2]vedi capitoli 2 e 3
[3]vedi sottosezione 1.6

Profili	Modello Vettoriale			Classificazione IPTC		
	Precision	Recall	F-measure	Precision	Recall	F-measure
P1	0,6667	1,0000	0,8000	0,0000	0,0000	0,0000
P2	0,1333	1,0000	0,2353	0,5000	1,0000	0,6667
P3	0,5000	1,0000	0,6667	0,2857	1,0000	0,4444
P4	0,3333	1,0000	0,5000	0,0455	1,0000	0,0870
P5	0,7500	1,0000	0,8571	0,2500	1,0000	0,4000
P6	0,1765	1,0000	0,3000	0,2000	0,6667	0,3077
P7	0,4286	1,0000	0,6000	0,2500	1,0000	0,4000
P8	0,1250	1,0000	0,2222	0,5000	1,0000	0,6667
P9	0,6667	1,0000	0,8000	0,0417	0,5000	0,0769
P10	0,3333	1,0000	0,5000	0,0000	0,0000	0,0000
P11	0,6667	1,0000	0,8000	0,0000	0,0000	0,0000
P12	1,0000	1,0000	1,0000	0,2222	0,5000	0,3077
P13	1,0000	1,0000	1,0000	0,5000	1,0000	0,6667
P14	0,5000	1,0000	0,6667	0,3333	1,0000	0,5000

Figura 4.2: Analisi sull'efficacia: Precision, Recall e F-Measure (risultati standard sull'utilizzo del modello Vettoriale a sinistra e sull'utilizzo della classificazione IPTC a destra).

un maggiore score) estratte dal software COGITO[4], per tutti i 14 profili utente costruiti appositamente e presi in considerazione per la valutazione del software AMBIT e delle sue funzionalità.

Come si nota, nella maggior parte dei casi, le classi IPTC sono correlate ai termini rilevanti della richiesta, per esempio:

- Ai profili 2 e 8, che contengono il termine "radio", con COGITO viene associata la classe omonima "Radio";
- Anche al profilo 7, che contiene il termine "television", viene associata la classe "Television";
- Al profilo 6 viene associata la classe "Language", correlata concettualmente al termine "speaker", estratto come uno dei termini rilevanti del profilo;
- Stessa cosa succede anche al profilo 14, dove con termini come "credit", "payment" e "card", viene associata la classe correlata "Credit and Debt";
- A tutti gli altri profili vengono associate le classi "Hardware", "Software" e "Wireless Tecnology", correlate in quasi tutti i casi ai termini della richiesta. Questo dipende anche dalla tipologia di sistema di help-desk utilizzato, che in questo caso si basa proprio su questi argomenti;

[4]vedi sottosezione 2.2.3

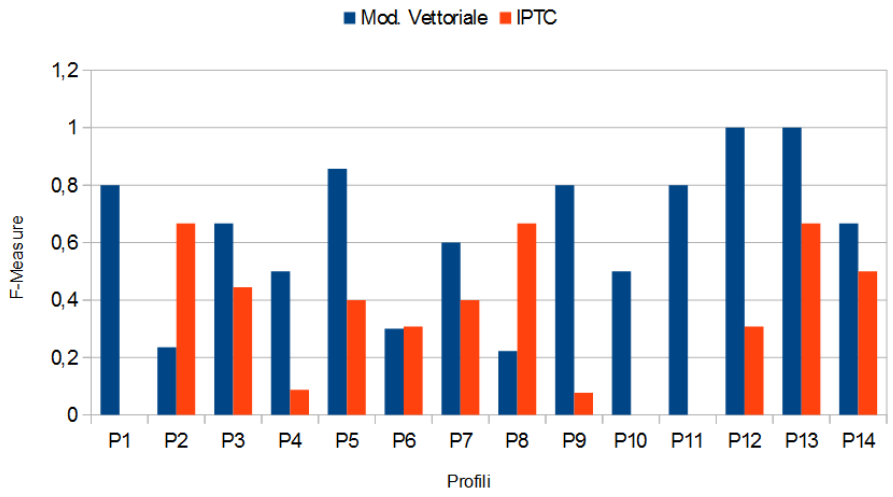

Figura 4.3: Media armonica (F-Measure) tra i valori di Recall e Precision dei documenti recuperati dai due ranking derivanti dal modello Vettoriale e dalla classificazione IPTC.

Come già descritto nei capitoli precedenti, i termini estraggono informazioni precise, mentre le classi esprimono concetti più generali[5], possiamo però dire che anche con le classi IPTC la classificazione dei documenti nella collezione avviene in maniera abbastanza precisa e può essere utilizzata traendone beneficio, come vedremo nelle sezioni immediatamente successive, per arrivare ad avere un ranking finale che sia una fusione ottimale dei due metodi.

Come si nota dalla Figura 4.3, dove vengono confrontati graficamente i valori di Recall e Precision per ogni profilo utente rispetto ad entrambi i ranking, tramite l'utilizzo della media armonica (F-Measure), il modello migliore a livello di recupero è il modello Vettoriale; comunque, tranne alcuni profili dove manca completamente il valore aggiunto dato dalle classi IPTC (come per i profili 1, 10 e 11), nel resto dei casi anche quest'ultima classificazione si comporta abbastanza bene. In alcuni casi oltretutto, compensa i bassi valori assunti dai ranking derivanti dal modello Vettoriale come per i profili 2 e 8, mantenendo soddisfacenti le performance generali.

Nel dettaglio, in Figura 4.2 sono mostrati i valori numerici di Precision, Recall e F-Measure che hanno portato alla definizione del grafico appena descritto.

[5]vedi sottosezione 2.2.5

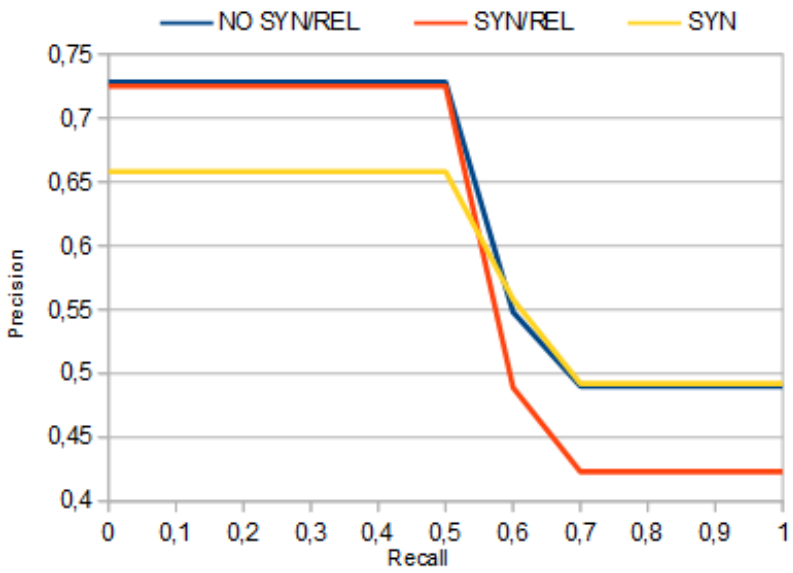

Figura 4.4: Precision media ad ogni livello standard di Recall per i ranking che variano a seconda dell'utilizzo o meno di sinonimi o termini correlati.

4.2 Utilizzo di sinonimi e termini correlati

Utilizzando il modello Vettoriale è possibile effettuare misure di similarità differenti nel caso in cui si utilizzino o meno sinonimi e/o termini correlati. E' possibile ottenere quindi diversi ranking, calcolando i pesi dei termini, in questo modo:

- utilizzando la tecnica di TF-IDF pura, che considera solamente i termini uguali;
- utilizzando anche i termini sinonimi e i termini correlati di ogni parola all'interno del vocabolario di riferimento;
- utilizzando solamente i termini sinonimi di ogni parola all'interno del vocabolario di riferimento.

Per questo motivo è stato analizzato l'impatto nell'uso di sinonimi e/o termini correlati. A questo proposito, possiamo considerare i profili 1, 3, 6, 8 e 11 in cui vengono utilizzati tali termini e il ranking varia in base alla tecnica prescelta.

Come mostrato dalla Figura 4.4, l'utilizzo combinato di sinonimi e termini correlati (indicato nella legenda come "SYN/REL") porta ad una perdita di

prestazioni nella fase di recupero dei documenti rilevanti ad alti livelli di Recall, si potrebbe prediligere quindi l'utilizzo dei sinonimi o addirittura dei soli termini uguali, che hanno un andamento migliore.

Ad esempio, analizzando il profilo 3 possiamo verificare come l'utilizzo dei sinonimi porti ad un effettivo miglioramento nel recupero dei documenti importanti. Utilizzando i termini "videodisk" e "warranty" rispettivamente come sinonimi dei termini "dvd" e "guarantee", viene correttamente recuperato in prima posizione il documento più rilevante per questo profilo; utilizzando solamente i termini uguali invece, questo documento è molto più in basso nel ranking abbassando di molto l'efficacia del recupero.

Analizzando il profilo 11 invece, possiamo verificare come i termini correlati possano portare ad un peggioramento delle prestazioni.

Questo profilo contiene i termini "camera" e "battery" che rispettivamente hanno come termini correlati "camcorder" e "factory". Il termine "camera" e il suo termine correlato "camcorder" possono essere utili al recupero perchè esprimono lo stesso concetto, come avviene per esempio anche per il profilo 1; il termine "battery" e il suo termine correlato "factory" invece, fanno capo a due concetti completamente differenti in questo scenario e conseguentemente vengono recuperati documenti che non c'entrano nulla con il profilo utente che si sta analizzando.

In conclusione, con l'utilizzo dei sinonimi, il recupero si comporta correttamente, infatti dal livello di recall 0.6 l'andamento è leggermente migliore rispetto all'utilizzo dei soli termini uguali; mentre utilizzando anche i termini correlati l'efficacia nel recupero di documenti importanti si abbassa.

4.3 Utilizzo di pesi maggiorati per i termini importanti

Come descritto nella sottosezione 3.2.2, per quanto riguarda lo scenario applicativo dell'help-desk, la cosa più importante per un utente che cerca informazioni nel sistema, è quella di avere risultati coerenti con quanto è stato richiesto.

Per dare risalto alla domanda posta dall'utente è stato incrementato il peso dei termini estratti dalla richiesta, rispetto al resto dei termini derivanti dal contesto.

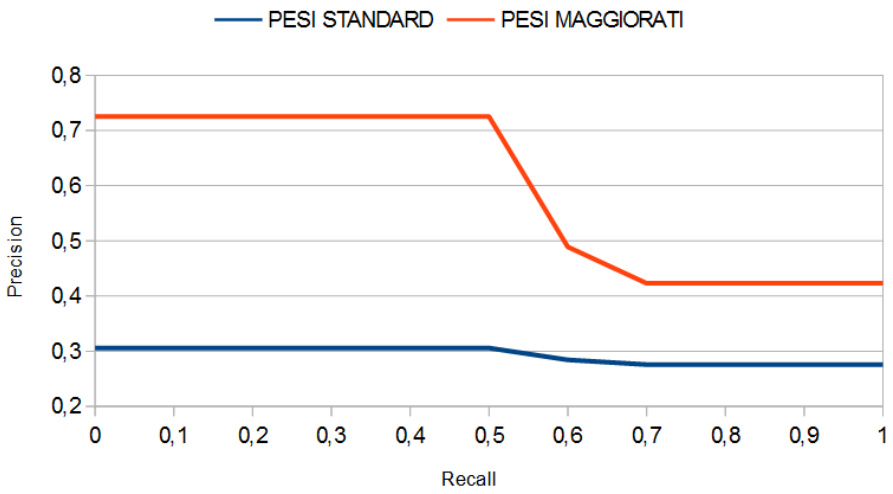

Figura 4.5: Precision media ad ogni livello standard di Recall per i ranking che utilizzano pesi standard o pesi maggiorati dei termini rilevanti.

I risultati ottenuti con questa ottimizzazione della tecnica standard utilizzata dal software AMBIT, sono rappresentati in Figura 4.5. Da questo grafico si può notare che in media il recupero delle informazioni è molto più performante aumentando il peso dei termini importanti relativamente alla richiesta dell'utente.

Questo comportamento è causato dal fatto che introducendo il contesto, la ricerca si popola di termini che potrebbero, in certi casi, non essere adeguati alle informazioni ricercate, si deve quindi dare più importanza alla richiesta di informazioni dichiarata dall'utente.

Analizzando tutti i profili si può notare che praticamente tutti i ranking derivanti da termini con pesi standard, hanno performance peggiori rispetto agli stessi profili in cui vengono utilizzati i termini con pesi maggiorati.

Ad esempio:

- per il profilo 1, utilizzando pesi maggiorati, vengono recuperati tutti i documenti importanti che troviamo nelle prime posizioni, mentre utilizzando pesi standard, viene recuperato solamente un documento importante che troviamo in una posizione più bassa del ranking;

- per il profilo 3, il ranking con pesi maggiorati migliora di una posizione tutti i documenti importanti, rispetto a quello con pesi standard;

- per il profilo 5, vengono recuperati tutti i documenti importanti da entrambi i ranking, ma in quello che utilizza i pesi maggiorati, i documenti si trovano in posizioni migliori;
- ecc...

In conclusione quindi, si può affermare che in un sistema di help-desk è fondamentale poter aumentare i pesi relativi ai termini della richiesta che l'utente sottopone al sistema, in quanto aumenta l'efficacia del recupero.

4.4 Utilizzo di ICF nel calcolo della similarità per le classi IPTC

Come descritto nella sottosezione 3.2.3, nell'ottimizzazione del software si è utilizzata una formula per il calcolo della similarità diversa da quella standard prevista dal progetto AMBIT[6].

In questa sezione verificheremo se questa modifica ha impatto sui ranking derivanti dalle classi IPTC e se sia utile ai fini del recupero o peggiori solamente le prestazioni.

Sono stati estratti con il software tutti ranking dei profili sia utilizzando, che non utilizzando, il calcolo della similarità con ICF. Sono stati calcolati Precision e Recall per ogni ranking ed infine è stato costruito un grafico che mostra la Precision media ad ogni livello standard di Recall, come mostrato in Figura 4.6.

Come si evince da questo grafico, possiamo notare come l'utilizzo di ICF migliori, anche se non in modo estremamente marcato, il recupero di documenti rilevanti utilizzando le classi IPTC.

Possiamo prendere come esempio il profilo 4: senza l'utilizzo del valore di ICF, come mostrato anche in Figura 4.1, le classi più rilevanti e con score maggiore sono quelle estratte nella maggior parte dei documenti, cioè *Hardware*, *Wireless Technology*, *Software*, ecc.... Sono presenti però anche un'altra grossa fetta di classi che si distanziano da quelle appena descritte, che sono molto meno comuni nella collezione, ma che invece caratterizzano il profilo, come *Waste Management and Pollution Control*, *Patent, Copyright and Trademark*, *Computer Crime*, *Insurance*, ecc.... Queste classi vengono invece sfruttate con l'utilizzo di ICF e prese in considerazione nel calcolo della similarità.

[6] vedi sottosezione 2.2.4

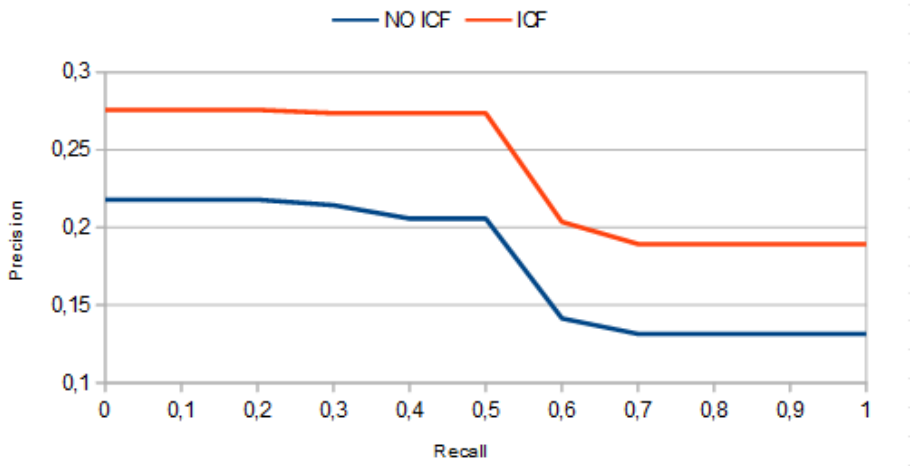

Figura 4.6: Precision media ad ogni livello standard di Recall utilizzando o meno il valore di ICF nel calcolo della similarità per le classi IPTC.

In termini pratici, senza l'utilizzo di ICF, il ranking non estrae nessun documento rilevante per il profilo utente, mentre con l'introduzione di questo valore, nel ranking sono presenti tutti e due i documenti rilevanti, uno dei quali è addirittura in prima posizione.

Con l'utilizzo di ICF quindi, viene dato più valore a queste classi, in modo che anche le classi meno frequenti nella collezione, ma che caratterizzano il profilo che si sta elaborando, possano essere comunque rilevanti per un recupero efficace dei documenti.

Per avere una visione più generale, riepiloghiamo di seguito i risultati ottenuti con l'utilizzo di ICF:

- *Prestazioni migliorate.* Per i profili 2, 3, 4, 6, 8, 12, 13 e 14, cioè più della metà dei profili considerati, il recupero è migliorato sia come score, sia come posizioni dei documenti rilevanti nei ranking;

- *Prestazioni uguali.* Per i profili 1, 10 e 11 le prestazioni sono rimaste le stesse, cioè nessun documento importante viene recuperato dai ranking.

- *Prestazioni peggiorate (con riserva).* Per i profili 5, 7 e 9, il recupero è leggermente peggiorato per quanto riguarda la posizione dei documenti importanti nei ranking, ma migliorato come score (utile in un'ottica di ranking fusion con tecniche che sfruttano lo score dei documenti, come vedremo successivamente).

In base ai risultati ottenuti, si può concludere che l'utilizzo di ICF in generale è più efficace rispetto al suo non utilizzo, anche se sicuramente ci sono margini di miglioramento. Studi e ottimizzazioni future potrebbero apportare miglioramenti a questa tecnica in modo che per tutti i profili venga recuperato almeno un documento rilevante e che le prestazioni non peggiorino rispetto al calcolo standard della similarità.

4.5 Confronto degli algoritmi di Ranking Fusion

Una delle funzionalità studiate in questo progetto, sono le tecniche di ranking fusion utilizzate per fondere i ranking derivanti dal calcolo della similarità con il modello Vettoriale e con le classi IPTC[7].

Come descritto precedentemente, sono state confrontate due tecniche differenti: la fusione tramite l'utilizzo della posizione dei documenti nei ranking e la fusione tramite l'utilizzo degli score assegnati ad ogni documento tramite il calcolo della similarità, che chiameremo rispettivamente **Rank Fusion** e **Score Fusion** per motivi di chiarezza e semplicità.

Come mostrato in Figura 4.7, confrontando i risultati ottenuti per ogni profilo e calcolandone una media, si nota come più o meno le due tecniche di fusione si equivalgano. Fino al livello di Recall 0.6 è leggermente migliore la tecnica basata sullo score dei documenti, mentre dopo questa soglia è la tecnica basata sul rank quella vincente.

Questo vuol dire che la tecnica di Score Fusion ha prestazioni migliori ad alti valori di Precision, mentre la tecnica di Rank Fusion funziona meglio su alti valori di Recall.

Possiamo quindi dire che nessuno dei due algoritmi di ranking fusion è nettamente migliore dell'altro, e che al momento possono essere utilizzati senza perdita di informazioni.

In effetti, verificando in dettaglio i vari risultati, si può notare come tutti i documenti rilevanti siano recuperati da tutte e due le tecniche e come compaiano comunque tra le prime posizioni variando di pochissimo.

[7] vedi sottosezione 3.2.5

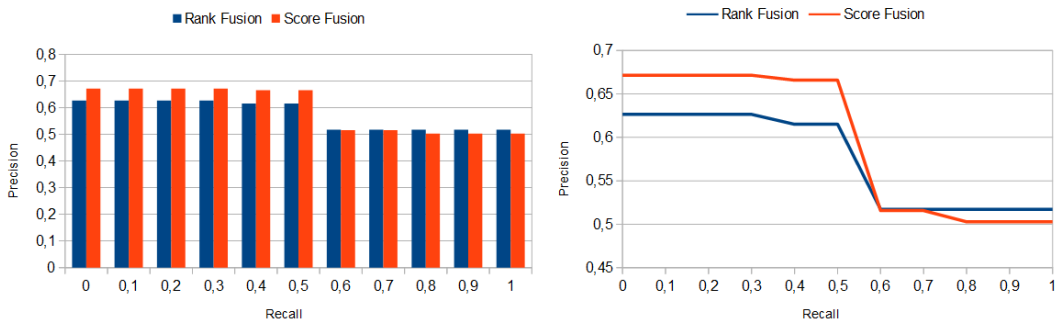

Figura 4.7: Precision media ad ogni livello standard di Recall per le tecniche di ranking fusion.

I profili dove questa differenza è più marcata sono ad esempio il numero 7 e il numero 9. Per il profilo 7 funziona meglio la tecnica di Rank Fusion, dove i documenti importanti si trovano dalla seconda alla quarta posizione, migliore della tecnica di Score Fusion dove troviamo gli stessi documenti nelle posizioni 2, 6 e 7.

Per il profilo 9 invece, vengono recuperati documenti importanti in prima e in quarta posizione dalla tecnica di Score Fusion, mentre con la tecnica di Rank Fusion troviamo questi documenti in posizione 5 e 11.

Come sviluppo e miglioramento futuro si dovrebbe arrivare ad utilizzare una sola tecnica di ranking fusion; una tecnica studiata e considerata la migliore per tutte le applicazioni del software AMBIT.

4.6 Utilizzo dei valori di Soglia

Nell'ottica del miglioramento delle prestazioni legate al recupero efficace ed efficiente dei documenti, si è pensato di utilizzare una tecnica che permettesse di effettuare il ranking fusion nel caso in cui i due ranking fossero entrambi rappresentativi, oppure scartarne uno qualora quest'ultimo non sia abbastanza informativo.

Sulla base di questa tecnica, descritta nella sottosezione 3.2.5, sono stati studiati i risultati ottenuti dal recupero dei documenti con i valori di soglia, confrontandoli con gli algoritmi di ranking fusion presi in considerazione nel progetto, Rank Fusion e Score Fusion.

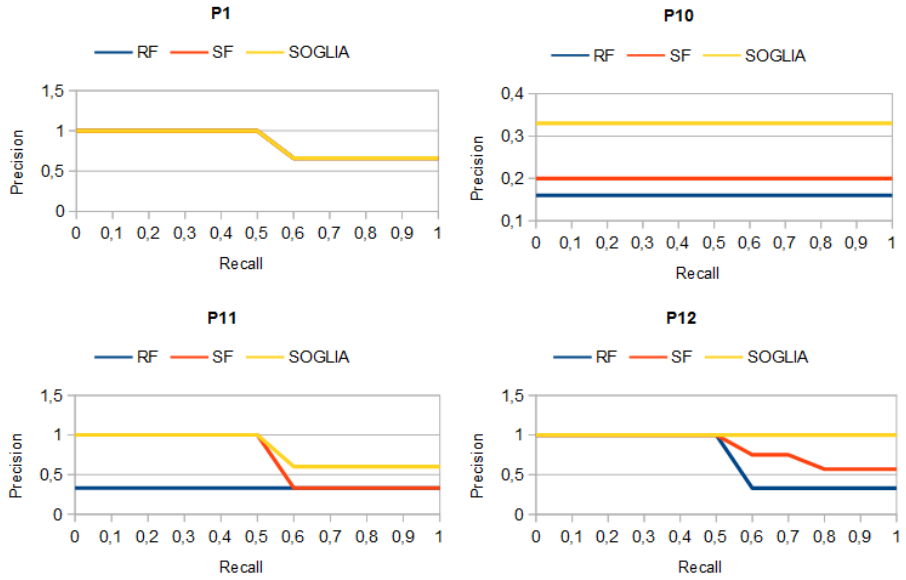

Figura 4.8: Precision media ad ogni livello standard di Recall per le tecniche di ranking fusion con l'utilizzo dei valori di soglia, in dettaglio per i profili 1 (P1), 10 (P10), 11 (P11) e 12 (P12).

I profili per i quali vengono utilizzati i valori di soglia sono i profili numero 1, 10, 11 e 12, mostrati più dettagliatamente in Figura 4.8. In figura è descritta la performance generale del sistema in base a questi quattro profili, cioè ad ogni livello standard di recall si misura la precisione del recupero.

In questi grafici si nota come l'utilizzo del valore di soglia nella scelta del ranking ottimale possa migliorare la fase del recupero dati. Infatti, mentre per il profilo 1 tutti i valori si sovrappongono restituendo quindi un ranking uguale per tutte le tecniche mostrate nel grafico, per gli altri profili notiamo che l'utilizzo del valore di soglia, migliora effettivamente il recupero dei dati (nei grafici la linea che rappresenta il ranking con l'utilizzo dei valori di soglia è sempre quella che assume i valori più alti).

Ovviamente, non per tutti i ranking verrà utilizzato questo valore di soglia, perchè dipendente dall'importanza del ranking in base al totale dello score dei documenti che lo compongono. Per tutti gli altri ranking, verranno utilizzate le tecniche di ranking fusion basate sullo score o sul rank dei documenti.

Proprio per questo motivo si è cercato anche di verificare se i risultati ottenuti in media su tutti i profili, possano essere soddisfacenti o meno.

I risultati sono visibili in Figura 4.9, dove si può notare che le due tecniche di

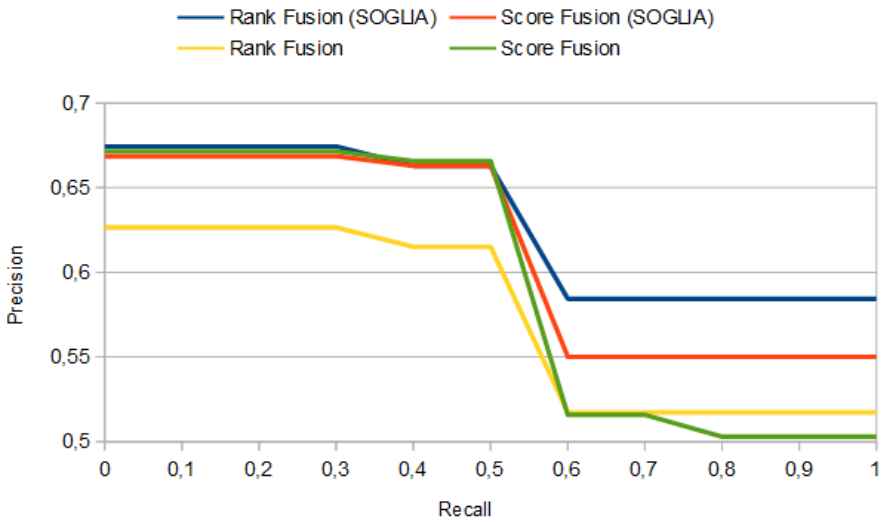

Figura 4.9: Precision media ad ogni livello standard di Recall per le tecniche di ranking fusion e ranking fusion con l'utilizzo dei valori di soglia.

ranking fusion a cui viene applicato l'utilizzo dei valori di soglia per i profili che lo prevedono, aumentano in generale l'efficacia del recupero rispetto alle stesse tecniche che invece non utilizzano questi valori.

Capitolo 5

Implementazione del software

In questo capitolo verranno presentate in modo pratico, tramite l'utilizzo del codice sorgente del software AMBIT, le varie ottimizzazioni e le varie funzionalità del progetto presentate nei capitoli 2 e 3.

L'intero progetto è stato realizzato in **Python**, un linguaggio di programmazione ad alto livello e orientato agli oggetti, molto dinamico, semplice e flessibile, in modo da poter facilmente effettuare modifiche e/o aggiungere nuovi moduli.

Il modulo principale è **ambit.py**[1] dal quale vengono gestite ed eseguite tutte le funzionalità del progetto, che verranno descritte in seguito.

In particolare, il contenuto di questo capitolo si divide nelle seguenti sezioni:

- Come avviene l'estrazione dei dati (*Sezione 5.1*)
- Come viene creato il Glossario dei termini rilevanti (*Sezione 5.2*)
- Come viene creato l'Inverted Index (*Sezione 5.3*)
- Come avviene l'estrazione dei termini importanti dai Profili Utente (*Sezione 5.4*)
- Come vengono realizzate le funzioni di similarità (*Sezione 5.5*)
- Come vengono realizzati gli algoritmi di Ranking Fusion (*Sezione 5.6*)

[1] vedi appendice A.1

5.1 Estrazione dei dati

In questa sezione verrà presentato come, partendo da un insieme di documenti e di profili utente, vengono estratti i dati nel software, sia attraverso la lettura vera e propria dei documenti, sia attraverso la categorizzazione per classi effettuata dal web service basato su COGITO.

5.1.1 Estrazione dei dati dai documenti

L'estrazione del testo sia dai documenti della collezione che dai profili utente, è affidata al modulo **extractText.py**[2].

La fase vera e propria di estrazione del testo avviene all'interno del metodo **extractText.extractText**, che differenzia il suo comportamento in base al tipo di dato ricevuto in ingresso:

- se si vuole elaborare un documento di testo, allora il metodo apre il file, legge ogni riga eliminando i caratteri di andata a capo e ritorna il testo concatenato.

```
if docType == "TXT":
        in_file = open(docContent,"r")
        text = in_file.readlines()
        no_n1 = [text[i].rstrip('\r\n') for i in range(len(text))]
        text[:] = no_n1
        no_n2 = ""
        for x in text:
                no_n2 += x + " "
        text = (docId,no_n2)
        in_file.close()
        return text
```

- se si tratta della richiesta che l'utente fa al sistema, il metodo non fa niente, ritorna semplicemente la stringa passata in input.

```
if docType == "QUERY":
        return ('QUERY',docContent)
```

Queste operazioni di estrazione vengono applicate ad ogni documento contenuto nella lista **docsList**, passata al metodo **extractText.multiExtractText**. Per ogni elemento della lista, vengono estratte le informazioni riguardanti ogni documento (nome, contenuto e tipo) e passate al metodo di estrazione appena

[2]vedi appendice A.5

descritto; ciò che viene ritornato, viene memorizzato in un file di testo **docsFile** che sarà utilizzato in altri moduli del software.

```
def multiExtractText(docsList,docsFile):
        pars = []
        for doc in docsList:
                docId = doc[0]
                docContent = doc[1]
                docType = doc[2]
                par = extractText(docId, docContent, docType)
                pars.append(par)
        outF = open(docsFile,'w')
        for p in pars:
                try:
                        if len(p) > 1:
                                if p!=pars[0]:
                                        outF.write('\n')
                                outF.write(p[0]+"\t"+p[1])
                except:
                        raise TypeError("Error")
        outF.close()
        print "Done!"
        return pars
```

5.1.2 Estrazione delle classi IPTC

L'estrazione dei dati rilevanti effettuata dal web service basato su COGITO, è affidata al modulo **cogito.py**[3] il quale, grazie al metodo **cogito.analyzeText** invia il contenuto del documento o del profilo utente, al web service.

```
def analyzeText(docContent, docType):
        print "Analysing "+docContent+"... API COGITO UFFICIALI ..."
        urlS = "https://services.cogitoapi.com/1.0/media/cat/iptc"
        headers = { 'Apikey':'rrn6a4pg2pgev363k3kgqvjc', 'Content-Type':'
            application/x-www-form-urlencoded', 'Accept':'application/xml' }
        if docType == "QUERY":
                query_args = { 'text':docContent }
        if docType == "TXT":
                in_file = open(docContent,"r")
                text = in_file.read()
                in_file.close()
                query_args = { 'text':text }
        data = urllib.urlencode(query_args)
        request = urllib2.Request(urlS, data, headers)
        response = urllib2.urlopen(request)
        xml = response.read()
        d = extractDomains(xml)
        e = extractEntities(xml)
        r = extractRelevants(xml)
```

[3] vedi appendice A.3

```
19          return (d,e,r)
```

Il web service risponde inviando una pagina XML, con tutti i dati importanti relativi alla categorizzazione effettuata per quel documento. Dalla pagina XML, si possono così estrarre le classi IPTC che caratterizzano quel documento, tramite il metodo **cogito.extractDomains**.

```
1  def extractDomains(data):
2          ris = []
3          pat = re.compile(r'<category name="([^"]*?)" score="([^"]*?)" taxonomy="
               IPTC">', re.I)
4          result = re.findall(pat, data)
5          for r in result:
6                  ris.append((r[0],r[1]))
7          return ris
```

Queste operazioni di estrazione vengono applicate ad ogni documento contenuto nella lista **docsList**, passata al metodo **cogito.multiAnalyzeText**. Per ogni elemento della lista, vengono estratte le informazioni riguardanti ogni documento (nome, contenuto e tipo) e passate al metodo di estrazione appena descritto; ciò che viene ritornato, viene memorizzato in un file .dat creato per mezzo della libreria "Pickle", che in seguito verrà utilizzato in altri moduli del software.

```
1  def multiAnalyzeText(docsList,outFile):
2          analyses = []
3          for doc in docsList:
4                  docId = doc[0]
5                  docContent = doc[1]
6                  docType = doc[2]
7                  if docType == "QUERY":
8                          docContent *= 10
9                  an = analyzeText(docContent, docType)
10                 time.sleep(1)
11                 analyses.append((docId,an))
12         f_out = open(outFile,"w")
13         pickle.dump(analyses,f_out)
14         f_out.close()
15         print "Pickling OK!"
16         return analyses
```

5.2 Glossario

In questa sezione verrà presentato come i dati estratti nella sezione 5.1 vengano utilizzati per l'estrazione dei termini rilevanti, il calcolo dei valori di Term Fre-

quency e Inverse Document Frequency e la generazione del glossario utilizzato nelle fasi successive del progetto.

5.2.1 Estrazione dei termini rilevanti dai documenti

I dati estratti tramite le operazioni descritte nella sottosezione 5.1.1, vengono utilizzati per costruire un glossario che comprende tutti i termini rilevanti di ogni documento, e per ogni termine i valori di Term Frequency e Inverse Document Frequency. Tutte queste operazioni vengono effettuate nel modulo **glossaryExtractor.py**[4].

L'estrazione dei termini rilevanti è gestita dal metodo **glossaryExtractor.extractTermData** che, tramite la libreria "Topia", estrapola dal contenuto del documento che sta elaborando i termini rilevanti, filtrando i caratteri non voluti (come parentesi e simboli). Questo metodo restituisce due array di tuple: **docTerms** contenente per ogni termine estratto, il termine, il documento a cui appartiene e quante volte ricorre nel documento; **docLengths** contente per ogni documento, il nome del documento e il numero di termini rilevanti estratti.

```
f = open(inputFile,'rU')
for line in f:                    # for each doc: extract term data
        curDocLen = 0
        line = line.strip()
        qmCode = line[:line.find("\t")]  # qm code
        qmText = line[(line.find("\t")+1):].lower() # qm text description (lower
             case)
        for term in sorted(extractor(qmText)):   # extracted terminology
                if ( "." in term[0] or "(" in term[0] or ")" in term[0] or "+"
                    in term[0]   # filter out unwanted terms
                     or "\xa0" in term[0] or "\x80" in term[0] or "/" in term[0]
                          or "\xe2\x84\xa2" in term[0]
                     or ":" in term[0] or term[0].isdigit() or len(term[0])<3):
                        continue
                docTerm = (term[0].lower(), qmCode, term[1], term[2]) # term,
                    code, freq, strenght
                docTerms.append(docTerm)
                curDocLen = curDocLen+1
        docLengths.append((qmCode,curDocLen))
        docTerms.append(("zzzzz","zzzzz",-1,-1)) # last docTerm(terminator)
f.close()
```

[4]vedi appendice A.6

5.2.2 Generazione del glossario

La costruzione del glossario è affidata al metodo **glossaryExtractor.go** il quale si occupa di definire i valori di Term Frequency e Inverse Document Frequency dei termini estratti nella fase precedente.

Il valore di Term Frequency è calcolato attraverso il metodo **glossaryExtractor.generateReport** che effettua il rapporto tra il numero di occorrenze di un termine (**curTerm**) all'interno di un documento (**curDoc**) e la lunghezza del documento (**curDocLen**). Alla fine, viene restituito un array di tuple formate da: il codice del documento, il termine e il valore di Term Frequency calcolato.

```
def generateReport(docTerms, docLengths, termIDFs):                    # report
    generation
        termTFs = []    # doc-code, term, tf
        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
```

Il valore di Inverse Document Frequency è ricavato attraverso il metodo **glossaryExtractor.generateGlossary** che calcola per ogni termine il logaritmo del rapporto tra il numero di documenti (**numDocs**) e il valore di Document Frequency del termine (**curFreq**). Alla fine, il metodo restituisce il numero di termini nel glossario (**termCount**) e un dizionario che ha come chiavi i termini del glossario e come argomenti i rispettivi valori di Inverse Document Frequency (**termIDFs**).

```
def generateGlossary(docTerms,numDocs,USE_COGITO): # glossary generation
        curTerm = ""    # current term
        termIDFs = {}    # term, idf
        termCount = 0    # number of terms in glossary
        for docTerm in sorted(docTerms):
                if (docTerm[0]!=curTerm):    # new term
                        if curTerm!="":
                                curFreq = len(curDocList)
```

```
                                    curIDF = math.log(numDocs/curFreq)
                                    termIDFs[curTerm] = curIDF
                                    termCount +=1
                            curTerm=docTerm[0]
                            curDocList = []
                    if (USE_COGITO):
                            curDoc = docTerm[1]
                    else:
                            curDoc = docTerm[1].split('-')[0] # extract doc id from
                                paragraph id
                    if curDoc not in curDocList:
                            curDocList.append(curDoc)
            return termCount, termIDFs
```

Una volta terminate queste operazioni, il metodo **glossaryExtractor.go** salva su un file (**outFile**), tramite libreria "Pickle", il contenuto di un array (**glossaryData**) formato da i valori di Term Frequency e Inverse Document Frequency appena calcolati.

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

5.3 Inverted Index

La costruzione dell'Inverted Index, descritto nella sottosezione 2.2.2, è affidata al metodo **similarityComp.genIndexFile** che legge il glossario generato nella sezione 5.2 attraverso il metodo **similarityComp.readGlossary** e crea quattro dizionari:

- Tramite il metodo **similarityComp.genInvertedIndex** viene creato il dizionario delle occorrenze, che contiene per ogni termine nel glossario, la lista dei documenti in cui occorre quel termine.

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

```
10                          list.append(doc)
11                          invIndex[term]=list
12          return invIndex
```

- Tramite il metodo **similarityComp.genDocTerms** viene creato il dizionario delle frequenze, che contiene per ogni documento una lista di tuple formate dal termine appartenente al documento e il relativo 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
```

- Tramite il metodo **similarityComp.genAllWnSyns** viene creato il dizionario dei sinonimi,

```
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
```

che contiene per ogni termine la lista dei suoi termini sinonimi, scelti attraverso il 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.lower())
8          syns = sorted(list(set(syns)))
9          return syns
```

- Tramite il metodo **similarityComp.genAllWnRelated** viene creato il dizionario dei termini correlati,

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

  che contiene per ogni termine la lista dei suoi termini correlati, scelti attraverso il 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):
                                        rel.append(he.lower())
                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):
                                                rel.append(h.lower())
        rel = sorted(list(set(rel)))
        return rel
```

5.4 Estrazione dei termini dai Profili Utente

L'estrazione dei termini rilevanti dai profili utente, viene effettuata nel modulo **ambitDef.py**[5], attraverso il metodo **ambitDef.queryTerm**.

Questo metodo utilizza due metodi già visti:

- il metodo **glossaryExtractor.extractTermData** per estrarre i termini importanti;

[5] vedi appendice A.2

- il metodo **glossaryExtractor.generateReport** per estrarre il valore di Term Frequency di ogni termine nel glossario.

Il valore di Term Frequency è utilizzato come peso da assegnare al termine; più peso ha, più un termine è importante nel profilo. Infatti, il peso dei termini relativi alla richiesta che l'utente effettua al sistema di help-desk vengono aumentati di 50 volte, perchè dovranno essere quelli in assoluto più importanti. Il peso dei termini uguali viene sommato e tutti i termini vengono inseriti in un dizionario che verrà poi restituito e utilizzato nelle fasi successive.

```
1  def queryTerm(inputFile,indexFile):
2          docTerms, docLengths = glossaryExtractor.extractTermData(inputFile)
3          termTFs = glossaryExtractor.generateReport(docTerms, docLengths, None)
4          TERM = {}
5          for doc in termTFs:
6                  w = doc[2]
7                  if doc[0] == "QUERY":
8                          w *= 50.0
9                  if doc[1] in TERM:
10                         TERM[doc[1]] += w
11                 else:
12                         TERM[doc[1]] = w
13         QUERY_TERM = TERM.items()
14         return QUERY_TERM
```

5.5 Funzioni di similarità

In questa sezione verranno presentate le due funzioni di similarità con modello Vettoriale e con le classi IPTC, descritte nelle sezioni 3.2.2 e 3.2.3, nel modo in cui sono state implementate nel software AMBIT.

5.5.1 Similarità attraverso il modello Vettoriale

Il calcolo della similarità con il modello Vettoriale ed il relativo ranking estratto, descritto nella sottosezione 2.2.4, viene effettuato nel modulo **ambitDef.py**[6], attraverso il metodo **ambitDef.ranking**.

La similarità è modellata per mezzo del metodo **similarityComp.executeQuery** che, oltre ai parametri di input come i termini del profilo (**queryTerms**), il glossario (**glossaryFile**) e l'inverted index (**indexFile**), permette di specificare anche se includere o meno i termini sinonimi (**USE_SYNS**) e i termini

[6] vedi appendice A.2

correlati (**USE_REL**) nel calcolo della similarità. In questo modo è stato possibile utilizzare un solo metodo per il calcolo delle tre differenti similarità: solo termini uguali, includendo anche sinonimi e termini correlati, oppure includendo solo i termini sinonimi.

```
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
```

La funzione di similarità vera e propria, è implementata all'interno del metodo **similarityComp.ambitSimilaritySingleDocV2** che applicato ad ogni documento della collezione, non fa altro che la somma dei punteggi, calcolati con la formula descritta nella sottosezione 2.2.4, di tutti i termini estratti dal documento che sono uguali o se specificato, sinonimi e/o correlati.

```
if (term1==term2): # equal terms
        bestT1Score=(EQ_SCORE*w1*w2)
        break
if (USE_SYNS and isWnSynonym(term1,term2,synTerms)): # synonym terms
        if (bestT1Score<SYN_SCORE*w1*w2):
                bestT1Score=SYN_SCORE*w1*w2
        continue
if ( USE_REL and isRelated(term1,term2,relTerms) ): # related terms
        if (bestT1Score<REL_SCORE*w1*w2):
                bestT1Score=REL_SCORE*w1*w2
```

Una volta calcolata la similarità per ogni documento della collezione, questa viene ritornata al metodo iniziale **ambitDef.ranking** che normalizza tutti i punteggi attraverso il metodo **similarityComp.normalizeRank**,

```
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
```

e calcola il punteggio totale sommando gli score di tutti i documenti nel ranking attraverso il metodo **similarityComp.totScoreRank**, per la successiva fase di ranking fusion.

```
def totScoreRank(ranking):
        totScore = 0.0
        for rankItem in ranking:
                totScore += rankItem[0]
        return totScore
```

5.5.2 Similarità attraverso le classi IPTC

Il calcolo della similarità attraverso le classi IPTC ed il relativo ranking estratto, descritto nella sottosezione 2.2.4, viene effettuato nel modulo **ambitDef.py**[7], attraverso il metodo **ambitDef.rankingIPTC**.

Per prima cosa è necessario estrarre le classi IPTC di tutti i documenti della collezione e dei profili utente dai file creati nella sottosezione 5.1.2, attraverso il metodo **similarityComp.extractIPTCs**.

Dopodichè verranno effettuate altre due fasi preliminari:

- Calcolo del valore di "Inverse Class Frequency", descritto nella sottosezione 3.2.3, attraverso il metodo **similarityComp.icfIPTCs**;
- Elaborazione delle classi estratte dai profili utente, in modo da fondere le classi uguali e sommare i relativi score, atraverso il metodo **similarityComp.mergeIPTCs**.

Il calcolo della similarità tra le classi del profilo utente che si sta elaborando e le classi di ogni documento della collezione, è affidata al metodo **similarityComp.similarityIPTC**.

In prima battuta, viene calcolata la distanza sull'albero delle classi IPTC, indicata dalla variabile **iPath**; se la distanza tra le due classi è uguale a 1, quindi le classi sono uguali, il punteggio (varibile **score**) viene moltiplicato per il peso della classe IPTC del profilo utente (variabile **w**), peso calcolato moltiplicando lo score della classe per il relativo valore di ICF.

```
if iPath==1:
        w = float(profile[1])
        for icf in icfIPTC:
                if icf[0] == profile[0]:
```

[7] vedi appendice A.2

```
5                           w *= icf[1]
6          score *= w
```

Infine vengono sommati tutti i punteggi ottenuti per ogni documento della collezione, per definire il punteggio finale di quel documento.

```
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]])
```

Anche in questo caso, una volta calcolata la similarità per ogni documento della collezione, questa viene ritornata al metodo **ambitDef.rankingIPTC** che normalizza tutti i punteggi attraverso il metodo **similarityComp.normalizeRank** e calcola il punteggio totale sommando gli score di tutti i documenti nel ranking attraverso il metodo **similarityComp.totScoreRank**, visto nella sottosezione precedente.

5.6 Ranking Fusion

In questa sezione verranno esposte in modo pratico tutte le tecniche di ranking fusion utilizzate nel progetto e descritte nella sottosezione 3.2.5, tra cui Rank Fusion, Score Fusion e l'utilizzo dei valori di soglia.

5.6.1 Tecniche di Ranking Fusion

Le tecniche di ranking fusion utilizzate nel progetto, chiamate Rank Fusion e Score Fusion, e descritte nella sottosezione 3.2.5, vengono definite nel modulo **ambitDef.py**[8], rispettivamente attraverso i metodi **ambitDef.rankingFusion_rankFusion** e **ambitDef.rankingFusion_scoreFusion**.

```
1 def rankingFusion_rankFusion(rank1,ranking2, w1, w2):
2          ranking=[]
3          EWF1 = len(rank1)+1
4          EWF2 = len(ranking2)+1
5          count1 = 0
```

[8] vedi appendice A.2

```
        for doc1 in rank1:
                count1+=1
                isIn=False
                count2 =0
                for doc2 in ranking2:
                        count2+=1
                        if doc1[1]==doc2[1]:
                                isIn=True
                                num = (1.0 - ((count1 - 1.0) / EWF1)) + (1.0 -
                                    ((count2 - 1.0) / EWF2) )
                                ranking.append([num,doc1[1]])
                if isIn==False:
                        num = (1.0 - ((count1 - 1.0) / EWF1))
                        ranking.append([num,doc1[1]])

        count2 = 0
        doc = [r[1] for r in ranking]
        for doc2 in ranking2:
                count2 += 1
                if doc2[1] not in doc:
                        num = (1.0 - ((count2 - 1.0) / EWF2))
                        ranking.append([num,doc2[1]])

        sortedRank=sorted(ranking, key = lambda x: float(x[0]),reverse=True)
        return sortedRank
```

```
def rankingFusion_scoreFusion(rank1,ranking2, w1, w2):
        ranking=[]
        count1 = 0
        for doc1 in rank1:
                count1+=1
                isIn=False
                count2 =0
                for doc2 in ranking2:
                        count2+=1
                        if doc1[1]==doc2[1]:
                                isIn=True
                                num = 2 * ((w1 * doc1[0]) + (w2 * doc2[0]))
                                ranking.append([num,doc1[1]])
                if isIn==False:
                        num = w1 * doc1[0]
                        ranking.append([num,doc1[1]])

        count2 = 0
        doc = [r[1] for r in ranking]
        for doc2 in ranking2:
                count2 += 1
                if doc2[1] not in doc:
                        num = w2 * doc2[0]
                        ranking.append([num,doc2[1]])

        sortedRank=sorted(ranking, key = lambda x: float(x[0]),reverse=True)
        return sortedRank
```

5.6.2 Ranking Fusion con valori di soglia

Le teciche di ranking fusion appena viste, possono essere completate utilizzando dei valori di soglia che esprimono l'importanza dei due ranking presi in considerazione, quello derivante dal modello Vettoriale e quello derivante dalle classi IPTC; tale concetto è descritto nella sottosezione 3.2.5.

Nel software il calcolo del ranking nel caso del raggiungimento dei valori di soglia, viene effettuato nel modulo **ambitDef.py**[9], attraverso il metodo **rankingFusion_threshold**.

Questo motodo accetta in ingresso i punteggi totali dei due ranking calcolati nelle fasi descritte nelle sottosezioni 5.5.1 e 5.5.2 e non fa altro che calcolare il peso di entrambi i ranking tramite il semplice metodo **similarityComp.rankweight**, che somma tali punteggi e li normalizza, in modo che la loro somma sia 1.

```
1 def rankweight(score1, score2):
2         tot = score1 + score2
3         w1 = round(score1/tot, 1)
4         w2 = 1.0 - w1
5         return w1, w2
```

Questi pesi vengono ritornati al metodo **rankingFusion_threshold** che verifica se uno dei due pesi è maggiore del doppio rispetto all'altro; in questo caso viene restituito come ranking finale quello di importanza maggiore, scartando invece quello con peso minore.

```
1 rankFusM=[]
2 if w1 > w2 * 2:
3         rankFusM = rank1[:]
4 elif w2 > w1 * 2:
5         rankFusM = rank2[:]
```

[9] vedi appendice A.2

Conclusioni e sviluppi futuri

In questa Tesi sono state proposte alcune ottimizzazioni e valutate con prove sperimentali tutte le funzionalità del software AMBIT, utilizzando uno scenario applicativo ben definito come quello dell'help-desk intelligente. In questo capitolo verranno esposti tutti i risultati e gli obiettivi raggiunti rispetto a quelli dichiarati nella parte introduttiva di questa Tesi.

Gli obiettivi prefissati sono stati raggiunti in base alle capacità acquisite tramite lo studio teorico degli argomenti trattati e ai limiti di tempo da dover rispettare.

I principali risultati e l'efficacia dell'approccio proposto, come documentato nella sezione sperimentale, sono i seguenti:

- **Ottimizzazione** del ranking estratto tramite l'utilizzo del modello Vettoriale. Come descritto nella sottosezione 3.2.2 si è scelto di dare maggiore risalto ai termini della richiesta dell'utente al sistema di help-desk. I risultati dell'utilizzo di questa tecnica sono stati analizzati e valutati nella sezione 4.3.

- **Ottimizzazione** del ranking estratto tramite il software COGITO, che utilizza le classi IPTC per categorizzare i documenti. Come descritto nella sottosezione 3.2.3 è stato studiato l'utilizzo del valore di ICF nel calcolo della similarità e i risultati ottenuti sono stati presentati e valutati nella sezione 4.4;

- **Studio e valutazione** delle funzionalità standard del software AMBIT, come l'utilizzo dei termini sinonimi e/o correlati. Lo studio e i risultati ottenuti sono stati esposti nella sezione 4.2.

- **Studio e valutazione** di differenti tecniche di ranking fusion. Le tecniche utilizzate sono due (Rank Fusion e Score Fusion), sono state esposte nella

sottosezione 3.2.5 e i risultati ottenuti sono stati analizzati nella sezione 4.5.

- **Studio e valutazione** di una tecnica che in modo efficiente possa attribuire ai ranking un valore di importanza, da utilizzare nella fase di ranking fusion. La tecnica utilizzata è stata esposta nella sottosezione 3.2.4 e i risultati ottenuti sono stati analizzati in dettaglio nella sezione 4.9.

Nonostante il raggiungimento di questi obiettivi, il progetto è sicuramente migliorabile sotto alcuni aspetti, come:

- Utilizzo di altre forme di contesto oltre alla cronologia di navigazione passata dell'utente, per poter effettuare ricerche in modo ancora più mirato;
- Utilizzo, studio e valutazione di metodi per la disambiguazione dei termini estratti dai documenti (Word Sense Disambiguation), cioè l'operazione con cui si precisa il significato di ogni termine, assegnandogli quello più opportuno a seconda del contesto in cui si trova;
- Sviluppo e valutazione sperimentale di tutti gli scenari applicativi previsti dal progetto AMBIT;
- Ulteriore studio e ottimizzazione di alcune funzionalità, in base ai risultati ottenuti con le prove sperimentali descritte in questa Tesi, come:
  - l'utilizzo ottimale del valore di ICF nel calcolo della similarità per le classi IPTC;
  - l'utilizzo di un algoritmo di ranking fusion ottimale;
  - l'assegnazione di valori di importanza ai vari ranking;
  - l'utilizzo di questi valori di importanza per rendere ancora più efficace il recupero delle informazioni.

Appendice A

Archivio dei codici sorgente

In questa appendice vengono raccolti tutti i codici sorgente del software AMBIT, su cui si basa questa Tesi.

A.1 ambit.py

Codice sorgente di ambit.py, modulo principale del software.

```
# -*- coding: utf-8 -*-
import extractText
import cogito
import pickle
import glossaryExtractor
import similarityComp
import numpy as np
import numpy.linalg as LA
from time import sleep
import configList as d
import ambitDef as a

########## CONFIGURATIONS #############
PROFILE = 14
COGITO = True
RANKING_SIZE = 30

# 1 - Extract Terms, 2 - Extract Profile, 4 - Extract Glossary, 8 - Ranking
#OP = 1 | 2 | 4
OP = 8
#######################################

PROF_INDEX = PROFILE-1

if OP & 1 == 1:
        # 1a. EXTRACT TEXT WITH TOPIA
```

```
27          para = extractText.multiExtractText(d.DOCS_LIST, d.DOCS_FILE)    # list:
                docId-numPar, text
28
29          # 1b. EXTRACT TEXT WITH COGITO
30          if COGITO:
31                  resp = cogito.multiAnalyzeText(d.DOCS_LIST, d.COG_FILE)    # list
                        : docId, (domains, entities, relevants)
32
33  if OP & 2 == 2:
34          #2a. EXTRACT PROFILE WITH TOPIA
35          para = extractText.multiExtractText(d.PROFILE_LIST[PROF_INDEX], d.
                PROFILE_FILE[PROF_INDEX])    # list: docId-numPar, text
36
37          # 2b. EXTRACT TEXT PROFILE WITH COGITO
38          if COGITO:
39                  resp = cogito.multiAnalyzeText(d.PROFILE_LIST[PROF_INDEX], d.
                        COG_FILE_PROF[PROF_INDEX])    # list: profId, (domains,
                        entities, relevants)
40
41  if OP & 4 == 4:
42          # 3a. GENERATE GLOSSARY AND INDEX WITH COGITO FILE
43          if COGITO:
44                  glossaryExtractor.go(d.COG_FILE, d.COG_GLOSSARY_FILE,True)
45                  similarityComp.genIndexFile(d.COG_GLOSSARY_FILE,d.COG_INDEX_FILE
                        )
46
47          # 3b. GENERATE GLOSSARY AND INDEX WITHOUT COGITO FILE
48          glossaryExtractor.go(d.DOCS_FILE, d.GLOSSARY_FILE,False)
49          similarityComp.genIndexFile(d.GLOSSARY_FILE,d.INDEX_FILE)
50
51  if OP & 8 == 8:
52
53          QUERY_TERM = a.queryTerm(d.PROFILE_FILE[PROF_INDEX],d.INDEX_FILE)
54
55          print '\n1a) RANKING TRA IL PROFILO ESTRAPOLATO SENZA COGITO E IL
                GLOSSARIO DEI DOCUMENTI ESTRAPOLATO SENZA COGITO SENZA SYNS E SENZA
                RELATED'
56          rank1a = a.ranking(QUERY_TERM,d.GLOSSARY_FILE,d.INDEX_FILE,False,False,
                False,RANKING_SIZE, d.QUERY_SOL[PROF_INDEX])
57
58          print '\n1b) RANKING TRA IL PROFILO ESTRAPOLATO SENZA COGITO E IL
                GLOSSARIO DEI DOCUMENTI ESTRAPOLATO SENZA COGITO CON SYNS E RELATED'
59          rank1b, totScore1b = a.ranking(QUERY_TERM,d.GLOSSARY_FILE,d.INDEX_FILE,
                True,True,False,RANKING_SIZE, d.QUERY_SOL[PROF_INDEX])
60
61          print '\n1c) RANKING TRA IL PROFILO ESTRAPOLATO SENZA COGITO E IL
                GLOSSARIO DEI DOCUMENTI ESTRAPOLATO SENZA COGITO CON SYNS SENZA
                RELATED'
62          rank1c = a.ranking(QUERY_TERM,d.GLOSSARY_FILE,d.INDEX_FILE,True,False,
                False,RANKING_SIZE, d.QUERY_SOL[PROF_INDEX])
63
64          if COGITO:
65
```

```
66                  print "\n3a) RANKING CON LE CLASSI IPTC DI COGITO"
67                  rank3a, totScore3a = a.rankingIPTC(d.COG_FILE,d.COG_FILE_PROF[
                        PROF_INDEX],RANKING_SIZE,d.PROFILE_LIST[PROF_INDEX],d.
                        QUERY_SOL[PROF_INDEX])
68
69                  print "\n RANK FUSION"
70                  rankFus1b = a.rankingFusion_rankFusion(rank1b, rank3a,
                        totScore1b, totScore3a, d.QUERY_SOL[PROF_INDEX])
71
72                  print "\n SCORE FUSION"
73                  rankFus1b = a.rankingFusion_scoreFusion(rank1b, rank3a,
                        totScore1b, totScore3a, d.QUERY_SOL[PROF_INDEX])
74
75                  print "\n RANKING FUSION CON VALORI DI SOGLIA"
76                  rankFus1b = a.rankingFusion_threshold(rank1b, rank3a, totScore1b
                        , totScore3a, d.QUERY_SOL[PROF_INDEX])
```

A.2 ambitDef.py

Codice sorgente di ambitDef.py, modulo in cui vengono definiti i metodi utilizzati in ambit.py.

```
1  # -*- coding: utf-8 -*-
2  import extractText
3  import cogito
4  import pickle
5  import glossaryExtractor
6  import similarityComp
7  import numpy as np
8  import numpy.linalg as LA
9  from time import sleep
10 from os.path import split
11 import operator
12
13 def queryTerm(inputFile,indexFile):
14         docTerms, docLengths = glossaryExtractor.extractTermData(inputFile)
15         termTFs = glossaryExtractor.generateReport(docTerms, docLengths, None)
16         TERM = {}
17         for doc in termTFs:
18                 w = doc[2]
19                 if doc[0] == "QUERY":
20                         w *= 50.0
21                 if doc[1] in TERM:
22                         TERM[doc[1]] += w
23                 else:
24                         TERM[doc[1]] = w
25         QUERY_TERM = TERM.items()
26         return QUERY_TERM
27
```

```
28 def ranking(queryTerms,glossaryFile,indexFile,USE_SYNS,USE_REL,IS_COG,
       RANKING_SIZE,QUERY_SOL):
29         ranking1 = similarityComp.executeQuery(queryTerms,glossaryFile,indexFile
               ,USE_SYNS,USE_REL)
30         rank1 = similarityComp.normalizeRank(ranking1)
31         totScore = similarityComp.totScoreRank(rank1[:RANKING_SIZE])
32         similarityComp.printRanking(rank1[:RANKING_SIZE])
33         #similarityComp.evaluateResults(QUERY_SOL,rank1[:RANKING_SIZE])
34         return rank1[:RANKING_SIZE],totScore
35
36 def rankingIPTC(cogFile, cogFileProf, RANKING_SIZE, profList,QUERY_SOL):
37         docIPTCs,profIPTCs = similarityComp.extractIPTCs(cogFile,cogFileProf)
38         icfIPTC = similarityComp.icfIPTCs(docIPTCs)
39         mergedIPTCs = similarityComp.mergeIPTCs(profIPTCs)
40         #print 'mergedIPTCs :' +str(mergedIPTCs)
41         rank3 = similarityComp.similarityIPTC(docIPTCs,mergedIPTCs,icfIPTC)
42         rank4 = similarityComp.normalizeRank(rank3)
43         totScore = similarityComp.totScoreRank(rank4[:RANKING_SIZE])
44         similarityComp.printRanking(rank4[:RANKING_SIZE])
45         #similarityComp.evaluateResults(QUERY_SOL,rank4[:RANKING_SIZE])
46         return rank4[:RANKING_SIZE],totScore
47
48 def rankingFusion_rankFusion(rank1, rank2, totScore1, totScore2,QUERY_SOL):
49         print 'totScore1: ' +str(totScore1)+', totScore2: '+str(totScore2)
50         w1, w2 = similarityComp.rankweight(totScore1, totScore2)
51         print 'w1: ' +str(w1)+', w2: '+str(w2)
52         rankFusM = similarityComp.rankingFusion_rankFusion(rank1, rank2, w1, w2)
53         similarityComp.printRanking(rankFusM)
54         #similarityComp.evaluateResults(QUERY_SOL,rankFus1)
55         return rankFusM
56
57 def rankingFusion_scoreFusion(rank1, rank2, totScore1, totScore2, QUERY_SOL):
58         print 'totScore1: ' +str(totScore1)+', totScore2: '+str(totScore2)
59         w1, w2 = similarityComp.rankweight(totScore1, totScore2)
60         print 'w1: ' +str(w1)+', w2: '+str(w2)
61         rankFusM = similarityComp.rankingFusion_scoreFusion(rank1, rank2, w1, w2
               )
62         #rankFusM = similarityComp.normalizeRank(rankFusM)
63         similarityComp.printRanking(rankFusM)
64         return rankFusM
65
66 def rankingFusion_threshold(rank1, rank2, totScore1, totScore2, QUERY_SOL):
67         print 'totScore1: ' +str(totScore1)+', totScore2: '+str(totScore2)
68         w1, w2 = similarityComp.rankweight(totScore1, totScore2)
69         print 'w1: ' +str(w1)+', w2: '+str(w2)
70
71         rankFusM=[]
72         if w1 > w2 * 2:
73                 rankFusM = rank1[:]
74         elif w2 > w1 * 2:
75                 rankFusM = rank2[:]
76
77         similarityComp.printRanking(rankFusM)
```

```
78          return rankFusM
79
80
81  def mergePickleFile(file1, file2, fileout):
82          f_in = open(file1,"rb")
83          f_in2 = open(file2,"rb")
84          x = pickle.load(f_in)
85          y = pickle.load(f_in2)
86          xy = x+y
87          print 'x + y: ' +str(x+y)
88          f_in.close()
89          f_in2.close()
90          f_out = open(fileout,"wb")
91          pickle.dump(xy,f_out)
92          f_out.close()
93          print "Pickling OK!"
94          f = open('allAnalyses.dat',"rb")
95          z = pickle.load(f)
96          print 'z:' +str(z)
```

A.3 cogito.py

Codice sorgente di cogito.py, modulo in cui viene effettuata la connessione al web service basato su COGITO.

```
1   import re
2   import urllib
3   import urllib2
4   import pickle
5   import time
6
7   def analyzeText(docContent, docType):
8           print "Analysing "+docContent+"... API COGITO UFFICIALI ..."
9           urlS = "https://services.cogitoapi.com/1.0/media/cat/iptc"
10          headers = { 'Apikey':'rrn6a4pg2pgev363k3kgqvjc', 'Content-Type':'
                application/x-www-form-urlencoded', 'Accept':'application/xml' }
11          if docType == "QUERY":
12                  query_args = { 'text':docContent }
13          if docType == "TXT":
14                  in_file = open(docContent,"r")
15                  text = in_file.read()
16                  in_file.close()
17                  query_args = { 'text':text }
18          data = urllib.urlencode(query_args)
19          request = urllib2.Request(urlS, data, headers)
20          response = urllib2.urlopen(request)
21          xml = response.read()
22          d = extractDomains(xml)
23          e = extractEntities(xml)
24          r = extractRelevants(xml)
```

```
25          return (d,e,r)
26
27  def multiAnalyzeText(docsList,outFile):
28          analyses = []
29          for doc in docsList:
30                  docId = doc[0]
31                  docContent = doc[1]
32                  docType = doc[2]
33                  if docType == "QUERY":
34                          docContent *= 10
35                  an = analyzeText(docContent, docType)
36                  time.sleep(1)
37                  analyses.append((docId,an))
38          f_out = open(outFile,"w")
39          pickle.dump(analyses,f_out)
40          f_out.close()
41          print "Pickling OK!"
42          return analyses
43
44  def extractDomains(data):
45          ris = []
46          pat = re.compile(r'<category name="([^"]*?)" score="([^"]*?)" taxonomy="
                IPTC">', re.I)
47          result = re.findall(pat, data)
48          for r in result:
49                  ris.append((r[0],r[1]))
50          return ris
51
52  def extractEntities(data):
53          ris = []
54          lastType = ""
55          pat = re.compile(r'<ENTITY NAME="([^"]*?)">', re.I)
56          patType = re.compile(r'<ENTITIES TYPE="([^"]*?)">', re.I)
57          for line in data.splitlines():
58                  result = re.findall(pat, line)
59                  if len(result)>0:
60                          ris.append((result[0],lastType))
61                  else:
62                          result = re.findall(patType, line)
63                          if len(result)>0:
64                                  lastType = result[0].lower()
65          return ris
66
67  def extractRelevants(data):
68          ris = []
69          lastType = ""
70          pat = re.compile(r'<RELEVANT NAME="([^"]*?)"[^<>]*?(RANKING="([^"]*?)")
                ?[^<>]*?SCORE="([^"]*?)"/?>', re.I)
71          patType = re.compile(r'<RELEVANTS TYPE="([^"]*?)">', re.I)
72          for line in data.splitlines():
73                  result = re.findall(pat, line)
74                  if len(result)>0:
75                          ris.append((result[0][0],result[0][3],lastType))
```

```
76                  else:
77                          result = re.findall(patType, line)
78                          if len(result)>0:
79                                  lastType = result[0].lower()
80          return ris
```

A.4 configList.py

Codice sorgente di configList.py, modulo in cui vengono definiti i profili utente.

```
1  from os import listdir
2  from os.path import isfile, join
3
4  #=============================================================
5  # DOCS_LIST: collezione di documenti su cui basare i test
6  # I documenti vengono letti dalla directory configurata in DIR_PATH
7  #=============================================================
8
9  DIR_PATH = '/home/martina/Scrivania/Ambit2/DOCLIST/'
10 count = 0
11 DOCS_LIST = []
12 for f in listdir(DIR_PATH):
13         count += 1
14         if isfile(join(DIR_PATH,f)):
15                 DOCS_LIST.append((f, join(DIR_PATH,f), 'TXT'))
16 #print DOCS_LIST
17
18
19 #=============================================================
20 # PROFILE_LIST: profili utente da utilizzare per i test
21 # Tutti i profili contengono:
22 #       - una query effettuata sul sistema di help desk
23 #       - una serie di pagine della cronologia dell'utente relative alla
       collezione di documenti
24 #=============================================================
25
26 PROFILE_LIST = ( \
27 #=============================================================
28 # Il profilo 1 contiene:
29 #       - query: 'how to reset camera to factory settings'
30 # QUERY_SOL = ["CamcorderReset.txt","CameraReset.txt"]
31 #=============================================================
32 [('QUERY','how to reset camera to factory settings ','QUERY'),
33 ('P_1', join(DIR_PATH,'ImageRecoveryCamera.txt'),'TXT'),
34 ('P_2', join(DIR_PATH,'ModelNumberCamera.txt'),'TXT'),
35 ('P_3', join(DIR_PATH,'userguideCamera.txt'),'TXT'),
36 ('P_4', join(DIR_PATH,'userguideCamcorder2.txt'),'TXT'),
37 ('P_5', join(DIR_PATH,'TVNoSignal.txt'),'TXT')
38 ], \
39 #=============================================================
40 # Il profilo 2 contiene:
```

```
#       - query: 'how to listen to radio '
# QUERY_SOL = ["MUSIC_ListenMusicThroughRadio.txt","MUSIC_SmarthphoneListenRadio.txt"]
#=============================================================
[('QUERY','how to listen to Radio ','QUERY'),
('P_1', join(DIR_PATH,'MUSIC_EnablingFMRadio.txt'),'TXT'),
('P_2', join(DIR_PATH,'MUSIC_DigitalRadioWithoutDAB.txt'),'TXT'),
('P_3', join(DIR_PATH,'MUSIC_PlayRadioOnPC.txt'),'TXT'),
('P_4', join(DIR_PATH,'OnlineRadioTv.txt'),'TXT'),
('P_5', join(DIR_PATH,'TABTracksideRadio.txt'),'TXT')
], \
#=============================================================
# Il profilo 3 contiene:
#       - query: 'what are the terms of dvd guarantee ?'
# QUERY_SOL = ["WarrantyInformation.txt","WarrantyInformationVideodisk.txt"]
#=============================================================
[('QUERY','what are the terms of dvd guarantee ?','QUERY'),
('P_1', join(DIR_PATH,'WirelessIssue.txt'),'TXT'),
('P_2', join(DIR_PATH,'DeviceSupport.txt'),'TXT'),
('P_3', join(DIR_PATH,'WarrantyInformationPC.txt'),'TXT'),
('P_4', join(DIR_PATH,'WarrantyInformationPhone.txt'),'TXT'),
('P_5', join(DIR_PATH,'LimitedWarranty.txt'),'TXT')
], \
#=============================================================
# Il profilo 4 contiene:
#       - query: 'I have a problem with wireless computer mouse '
# QUERY_SOL = ["MouseSetupIssue.txt"]
#=============================================================
[('QUERY','I have a problem with wireless computer mouse ','QUERY'),
('P_1', join(DIR_PATH,'MouseNotMovingProperly.txt'),'TXT'),
('P_2', join(DIR_PATH,'ConnectionProblems.txt'),'TXT'),
('P_3', join(DIR_PATH,'MouseNotWorking.txt'),'TXT'),
('P_4', join(DIR_PATH,'GeneralPrinterTroubleshooting.txt'),'TXT'),
('P_5', join(DIR_PATH,'MUSIC_PlayRadioOnPC.txt'),'TXT')
], \
#=============================================================
# Il profilo 5 contiene:
#       - query: 'tv is not receiving any signals '
# QUERY_SOL = ["TVNtReceiveSignal.txt","TVNoSignal.txt","TVSignalReception.txt"]
#=============================================================
[('QUERY','tv is not receiving any signals ','QUERY'),
('P_1', join(DIR_PATH,'userguideBRAVIATelevision.txt'),'TXT'),
('P_2', join(DIR_PATH,'userguideBRAVIATelevision2.txt'),'TXT'),
('P_3', join(DIR_PATH,'userguideBRAVIATelevision3.txt'),'TXT'),
('P_4', join(DIR_PATH,'ResetTelevision.txt'),'TXT'),
('P_5', join(DIR_PATH,'WarrantyInformation.txt'),'TXT')
], \
#=============================================================
# Il profilo 6 contiene:
#       - query: 'car speakers produce no sound'
# QUERY_SOL = ["SpeakerNoSound.txt","SpeakerNoise.txt","SpeakerSoundInterrupted.txt",""]
#=============================================================
```

```
92 [('QUERY','car speakers produce no sound ','QUERY'),
93 ('P_1', join(DIR_PATH,'SpeakerSoundInterrupted.txt'),'TXT'),
94 ('P_2', join(DIR_PATH,'SpeakerBluetoothOperation.txt'),'TXT'),
95 ('P_3', join(DIR_PATH,'ModelNumberHomeShare.txt'),'TXT'),
96 ('P_4', join(DIR_PATH,'ModelNumberBookself.txt'),'TXT'),
97 ('P_5', join(DIR_PATH,'userguideBRAVIATelevision3.txt'),'TXT')
98 ], \
99 #=============================================================
100 # Il profilo 7 contiene:
101 #       - query 5 modificata: 'television is not receiving any signals '
102 # QUERY_SOL = ["TVNtReceiveSignal.txt","TVNoSignal.txt","TVSignalReception.txt"]
103 #=============================================================
104 [('QUERY','television is not receiving any signals ','QUERY'),
105 ('P_1', join(DIR_PATH,'userguideBRAVIATelevision.txt'),'TXT'),
106 ('P_2', join(DIR_PATH,'userguideBRAVIATelevision2.txt'),'TXT'),
107 ('P_3', join(DIR_PATH,'userguideBRAVIATelevision3.txt'),'TXT'),
108 ('P_4', join(DIR_PATH,'ResetTelevision.txt'),'TXT'),
109 ('P_5', join(DIR_PATH,'WarrantyInformation.txt'),'TXT')
110 ], \
111 #=============================================================
112 # Il profilo 8 contiene:
113 #       - query 2 modificata: 'how to listen radio by phone '
114 # QUERY_SOL = ["TVNtReceiveSignal.txt","TVNoSignal.txt"]
115 #=============================================================
116 [('QUERY','how to listen radio by phone ','QUERY'),
117 ('P_1', join(DIR_PATH,'MUSIC_EnablingFMRadio.txt'),'TXT'),
118 ('P_2', join(DIR_PATH,'MUSIC_DigitalRadioWithoutDAB.txt'),'TXT'),
119 ('P_3', join(DIR_PATH,'MUSIC_PlayRadioOnPC.txt'),'TXT'),
120 ('P_4', join(DIR_PATH,'OnlineRadioTv.txt'),'TXT'),
121 ('P_5', join(DIR_PATH,'TABTracksideRadio.txt'),'TXT')
122 ], \
123 #=============================================================
124 # Il profilo 9 contiene:
125 #       - query: 'My printer doesn''t print '
126 # QUERY_SOL = ["PrinterNotPrint.txt","GeneralPrinterTroubleshooting.txt"]
127 #=============================================================
128 [('QUERY','My printer doesn''t print ','QUERY'),
129 ('P_1', join(DIR_PATH,'PrinterPaperCaught.txt'),'TXT'),
130 ('P_2', join(DIR_PATH,'PrinterPrintSlowly.txt'),'TXT'),
131 ('P_3', join(DIR_PATH,'WarrantyInformation.txt'),'TXT')
132 ], \
133 #=============================================================
134 # Il profilo 10 contiene:
135 #       - query: 'how to reset my account password '
136 # QUERY_SOL = ["ResetPassword.txt"]
137 #=============================================================
138 [('QUERY','how to reset my account password ','QUERY'),
139 ('P_1', join(DIR_PATH,'ResetPasswordPS4.txt'),'TXT'),
140 ('P_2', join(DIR_PATH,'ChangePasswordPS3.txt'),'TXT'),
141 ('P_3', join(DIR_PATH,'ReaderSoftReset.txt'),'TXT'),
142 ('P_4', join(DIR_PATH,'userguideVAIO6.txt'),'TXT')
143 ], \
144 #=============================================================
```

```
# Il profilo 11 contiene:
#       - query: 'how to maximize battery life '
# QUERY_SOL = ["CameraBatteryNotCharge.txt","CameraBatteryProblems.txt"]
#=========================================================
[('QUERY','camera battery not charge ','QUERY'),
('P_1', join(DIR_PATH,'Battery.txt'),'TXT'),
('P_2', join(DIR_PATH,'userguideCamera3.txt'),'TXT'),
('P_3', join(DIR_PATH,'CamcorderReset.txt'),'TXT'),
('P_4', join(DIR_PATH,'MaximizeBatteryLife.txt'),'TXT')
], \
#=========================================================
# Il profilo 12 contiene:
#       - query: 'connect to a wireless network '
# QUERY_SOL = ["ConnectionProblems.txt","ConnectionToWirelessNetwork","
    ConnectionAutomatedTroubleshooter.txt","ConnectionCheckHardware.txt"]
#=========================================================
[('QUERY','problems connecting to a wireless network ','QUERY'),
('P_1', join(DIR_PATH,'BoostWiFiSignal.txt'),'TXT'),
('P_2', join(DIR_PATH,'CheckWirelessConnectionSpeed.txt'),'TXT'),
('P_3', join(DIR_PATH,'WiFiSignalDropsOut.txt'),'TXT'),
], \
#=========================================================
# Il profilo 13 contiene:
#       - query: 'play dowloaded video '
# QUERY_SOL = ["userguideSonyZ317.txt"]
#=========================================================
[('QUERY','play downloaded video ','QUERY'),
('P_1', join(DIR_PATH,'userguideCamera4.txt'),'TXT'),
('P_2', join(DIR_PATH,'CamcorderRenameMovieClip.txt'),'TXT'),
('P_3', join(DIR_PATH,'userguideTablet15.txt'),'TXT'),
], \
#=========================================================
# Il profilo 14 contiene:
#       - query: 'payment problems with credit card '
# QUERY_SOL = ["BillingInformation.txt"]
#=========================================================
[('QUERY','payment problems with credit card ','QUERY'),
('P_1', join(DIR_PATH,'PaymentMethods.txt'),'TXT'),
('P_2', join(DIR_PATH,'userguideTablet17.txt'),'TXT'),
('P_3', join(DIR_PATH,'CreateAccount.txt'),'TXT'),
] \
)

QUERY_SOL = (\
["CamcorderReset.txt","CameraReset.txt"], \
["MUSIC_ListenMusicThroughRadio.txt","MUSIC_SmarthphoneListenRadio.txt"], \
["WarrantyInformation.txt","WarrantyInformationVideodisk.txt"], \
["MouseSetupIssue.txt"], \
["TVNtReceiveSignal.txt","TVNoSignal.txt","TVSignalReception.txt"], \
["SpeakerNoSound.txt","SpeakerNoise.txt","SpeakerSoundInterrupted.txt",""], \
["TVNtReceiveSignal.txt","TVNoSignal.txt","TVSignalReception.txt"], \
["TVNtReceiveSignal.txt","TVNoSignal.txt"], \
["PrinterNotPrint.txt","GeneralPrinterTroubleshooting.txt"], \
```

```
197 ["ResetPassword.txt"], \
198 ["CameraBatteryNotCharge.txt","CameraBatteryProblems.txt"], \
199 ["ConnectionProblems.txt","ConnectionToWirelessNetwork","
    ConnectionAutomatedTroubleshooter.txt","ConnectionCheckHardware.txt"], \
200 ["userguideSonyZ317.txt"], \
201 ["BillingInformation.txt"] \
202 )
203
204 DOCS_FILE = 'allDocs.txt'
205 COG_FILE = 'allAnalyses.dat'     # list: docId, (domains, entities, relevants)
206 GLOSSARY_FILE = 'gloss.dat'                                  # termTFs:  doc-code,
    term,tf (array), termIDFs: term:idf (dict)
207 COG_GLOSSARY_FILE = 'cogGloss.dat'      # termTFs:  doc-code,term,tf (array),
    termIDFs: term:idf (dict)
208 INDEX_FILE = 'index.dat'          # term:termlist (dict), doc:docTerms (dict),
    term:relTermsList (dict)
209 COG_INDEX_FILE = 'cogIndex.dat'          # term:termlist (dict), doc:docTerms (
    dict), term:relTermsList (dict)
210
211 PROFILE_FILE = 'allProfile1.txt', 'allProfile2.txt', 'allProfile3.txt', '
    allProfile4.txt', 'allProfile5.txt', 'allProfile6.txt', 'allProfile7.txt', '
    allProfile8.txt', 'allProfile9.txt', 'allProfile10.txt', 'allProfile11.txt',
     'allProfile12.txt', 'allProfile13.txt', 'allProfile14.txt'
212 COG_FILE_PROF = 'prof1Analyses.dat', 'prof2Analyses.dat', 'prof3Analyses.dat', '
    prof4Analyses.dat', 'prof5Analyses.dat', 'prof6Analyses.dat', 'prof7Analyses
    .dat', 'prof8Analyses.dat', 'prof9Analyses.dat', 'prof10Analyses.dat', '
    prof11Analyses.dat', 'prof12Analyses.dat', 'prof13Analyses.dat', '
    prof14Analyses.dat'
213 PROF_GLOSSARY = 'prof1Gloss.dat', 'prof2Gloss.dat', 'prof3Gloss.dat', '
    prof4Gloss.dat', 'prof5Gloss.dat', 'prof6Gloss.dat', 'prof7Gloss.dat', '
    prof8Gloss.dat', 'prof9Gloss.dat', 'prof10Gloss.dat', 'prof11Gloss.dat', '
    prof12Gloss.dat', 'prof13Gloss.dat', 'prof14Gloss.dat'
214 PROF_COG_GLOSSARY = 'prof1CogGloss.dat', 'prof2CogGloss.dat', 'prof3CogGloss.dat
    ', 'prof4CogGloss.dat', 'prof5CogGloss.dat', 'prof6CogGloss.dat', '
    prof7CogGloss.dat', 'prof8CogGloss.dat', 'prof9CogGloss.dat', '
    prof10CogGloss.dat', 'prof11CogGloss.dat', 'prof12CogGloss.dat', '
    prof13CogGloss.dat', 'prof14CogGloss.dat'
```

A.5 extractText.py

Codice sorgente di extractText.py, modulo in cui viene effettuata l'estrazione dei dati dai documenti.

```
1 import re
2 import HTMLParser
3 import urllib
4
5 def extractText(docId, docContent, docType):
6         if docType == "QUERY":
7                 return ('QUERY',docContent)
```

```
8
9           if docType == "TXT":
10                  in_file = open(docContent,"r")
11                  text = in_file.readlines()
12                  no_n1 = [text[i].rstrip('\r\n') for i in range(len(text))]
13                  text[:] = no_n1
14                  no_n2 = ""
15                  for x in text:
16                          no_n2 += x + " "
17                  text = (docId,no_n2)
18                  in_file.close()
19                  return text
20
21          if docType == "HTML":
22                  print "Extracting "+docContent+"..."
23
24                  paragrafi = []
25                  count = 1
26                  h = HTMLParser.HTMLParser()
27                  MIN_PAR_LEN = 150
28                  MIN_TEXT_RATIO = 0.2
29
30                  # apre pagina web
31                  data = urllib.urlopen(docContent).read()
32                  data = data.decode("ISO-8859-1")
33
34                  # elimina a capo
35                  data = data.replace("\n", " ")
36                  data = data.replace("\r", " ")
37
38                  # estrae il body
39                  bodyPat = re.compile(r'<body[^<>]*?>(.*?)</body>', re.I)
40                  result = re.findall(bodyPat, data)
41                  data = result[0]
42
43                  # elimina java script
44                  p = re.compile(r'<script[^<>]*?>.*?</script>')
45                  data = p.sub('', data)
46
47                  # elimina css
48                  p = re.compile(r'<style[^<>]*?>.*?</style>')
49                  data = p.sub('', data)
50
51                  # elimina commenti
52                  p = re.compile(r'<!--(.)*?-->')
53                  data = p.sub('', data)
54
55                  # estrae i paragrafi
56                  parPat = re.compile(r'<(p|div)[^<>]*?>(.*?)</(p|div)>', re.I)
57                  result = re.findall(parPat, data)
58                  for par in result:
59                          par = par[1]
60                          pHTMLLen = len(par)
```

```
61
62                          # elimina tutti i tag
63                          p = re.compile(r'<[^<]*?>')
64                          par = p.sub(' ', par)
65                          par = p.sub(' ', par)
66
67                          # rimuove spazi consecutivi
68                          par = " ".join(par.split())
69
70                          # unescape
71                          par = h.unescape(par)
72
73                          # unicode to ascii
74                          par = par.encode('ascii', 'ignore')
75
76                          # calcolo rapporto testo/codice
77                          pTextLen = len(par)
78                          if pTextLen == 0:
79                                  continue
80                          pTextRatio = float(pTextLen)/pHTMLLen
81
82                          # filtra i paragrafi per lunghezza e per rapporto testo/
                                codice
83                          if pTextLen>MIN_PAR_LEN and pTextRatio > MIN_TEXT_RATIO:
84                                  paragrafi.append((docId+"-"+str(count),par))
85                                  count = count+1
86                  return paragrafi
87
88  def multiExtractText(docsList,docsFile):
89          pars = []
90          for doc in docsList:
91                  docId = doc[0]
92                  docContent = doc[1]
93                  docType = doc[2]
94                  par = extractText(docId, docContent, docType)
95                  pars.append(par)
96          outF = open(docsFile,'w')
97          for p in pars:
98                  try:
99                          if len(p) > 1:
100                                 if p!=pars[0]:
101                                         outF.write('\n')
102                                 outF.write(p[0]+"\t"+p[1])
103                 except:
104                         raise TypeError("Error")
105         outF.close()
106         print "Done!"
107         return pars
```

A.6 glossaryExtractor.py

Codice sorgente di glossaryExtractor.py, modulo in cui viene effettuata la creazione del glossario e dell'inverted index.

```
# -*- coding: utf-8 -*-
import math
import pickle
from topia.termextract import tag
from topia.termextract import extract
from nltk.corpus import wordnet as wn
import re

def countNumberOfDocuments(inputFile):              # count number of paragraphs and documents
        f = open(inputFile,'rU')
        numDocs = 0
        numPars = 0
        lastDoc = ""
        for line in f:
                line = line.strip()
                docId = line[:line.find("\t")]        # doc code
                if sameDoc(lastDoc,docId) == False:
                        numDocs +=1
                        lastDoc = docId
                numPars += 1
        f.close()
        return numPars, numDocs

def sameDoc(id1,id2):      # check if two paragraphs are from same doc
        if id1.split('-')[0] == id2.split('-')[0]:
                return True
        else:
                return False

def extractTermData(inputFile):                      # extract term data with Topia package
        MIN_OCCURS = 2     # minimum number of occurrences for term extraction (if not using DefaultFilter)
        docTerms = []     # term, code, freq, strength (temp data)
        docLengths = []   # code, len
        extractor = extract.TermExtractor()
        extractor.filter = extract.permissiveFilter
        # extractor.filter = extract.DefaultFilter(singleStrengthMinOccur=MIN_OCCURS)
        f = open(inputFile,'rU')
        for line in f:                           # for each doc: extract term data
                curDocLen = 0
                line = line.strip()
                qmCode = line[:line.find("\t")]        # qm code
                qmText = line[(line.find("\t")+1):].lower()    # qm text description (lower case)
                #print extractor.tagger(qmText)         # POS-tagged terms
```

```
44                  for term in sorted(extractor(qmText)):        # extracted
                        terminology
45                          if ( "." in term[0] or "(" in term[0] or ")" in term[0]
                                or "+" in term[0]   # filter out unwanted terms
46                                      or "\xa0" in term[0] or "\x80" in term[0]
                                            or "/" in term[0] or "\xe2\x84\xa2"
                                           in term[0]
47                                      or ":" in term[0] or term[0].isdigit() or
                                            len(term[0])<3):
48                                  continue
49                          docTerm = (term[0].lower(), qmCode, term[1], term[2]) #
                                term, code, freq, strenght
50                          docTerms.append(docTerm)
51                          curDocLen = curDocLen+1
52                  docLengths.append((qmCode,curDocLen))
53                  docTerms.append(("zzzzz","zzzzz",-1,-1))    # last docTerm (
                        terminator)
54          f.close()
55          return docTerms, docLengths
56
57
58
59  #la funzione estrae dal file di cogito i relevants i mainlemmas e i domains
60  #così da utilizzarli in seguito per costruire il glossario e l'inverted index
61  def extractCogitoData(cogFile):
62          print "Reading cogito data..."
63          docTerms=[]
64          docLenghts=[]
65          pickle_file = open(cogFile)
66          cogito_file = pickle.load(pickle_file)
67          pickle_file.close()
68          for docId in cogito_file:
69                  i=0
70                  currDocLen=0
71                  for ent_rel in docId[1]:
72                          if (i==0) :
73                                  i+=1
74                                  continue
75                          for currWord in ent_rel:
76                                  currDocLen+=1
77                                  if (currDocLen==1):
78                                          docTerms.append((currWord[0].lower(),
                                                docId[0],1))
79                                  else:
80                                          count=0
81                                          isInside=False
82                                          for pastTerm in docTerms:
83                                                  if (pastTerm[1]== docId[0]):
84                                                          if (currWord[0].lower()
                                                                ==pastTerm[0]):
85                                                                  docTerms[count]
                                                                        = list(
                                                                        docTerms[
```

```
                                                                    count])
86                                                                  docTerms[count
                                                                        ][2]+=1
87                                                                  docTerms[count]
                                                                        = tuple(
                                                                        docTerms[
                                                                        count])
88                                                                  isInside=True
89                                              count+=1
90                                          if (isInside == False):
91                                                  docTerms.append((
                                                        currWord[0].lower(),
                                                        docId[0],1))
92                          i+=1
93                  docLenghts.append((docId[0],currDocLen))
94                  docTerms.append(("zzzzz","zzzzz",-1))
95          numDocs = len(cogito_file)
96          return docTerms,docLenghts,numDocs
97
98  def generateGlossary(docTerms,numDocs,USE_COGITO):          # glossary
        generation
99          curTerm = ""    # current term
100         termIDFs = {}    # term, idf
101         termCount = 0    # number of terms in glossary
102         for docTerm in sorted(docTerms):
103                 if (docTerm[0]!=curTerm):    # new term
104                         if curTerm!="":
105                                 curFreq = len(curDocList)
106                                 curIDF = math.log(numDocs/curFreq)
107                                 termIDFs[curTerm] = curIDF
108                                 termCount +=1
109                         curTerm=docTerm[0]
110                         curDocList = []
111                 if (USE_COGITO):
112                         curDoc = docTerm[1]
113                 else:
114                         curDoc = docTerm[1].split('-')[0]    # extract doc id
                                from paragraph id
115                 if curDoc not in curDocList:
116                         curDocList.append(curDoc)
117         return termCount, termIDFs
118
119 def generateReport(docTerms, docLengths, termIDFs):                    # report
        generation
120         termTFs = []    # doc-code, term, tf
121         for doc in sorted(docLengths):        # for each document
122                 curDoc = doc[0]
123                 curDocLen = doc[1]
124                 for docTerm in docTerms:
125                         if (docTerm[1]==curDoc):    # relevant term
126                                 curTerm = docTerm[0]    # term
127                                 curFreq = docTerm[2]    # current term frequency
                                        (in current doc)
```

```
128                                     if curDocLen==0:
129                                             continue
130                                     curTF = float(curFreq)/curDocLen      # current
                                            term TF
131                                     curOutData = (curDoc, curTerm, curTF)
132                                     termTFs.append(curOutData)
133             return termTFs
134
135     def go(inputFile,outFile,USE_COGITO):
136             if (USE_COGITO) :
137                     print 'with cogito...'
138                     docTerms,docLengths,numDocs=extractCogitoData(inputFile)
139             else:
140                     numPars, numDocs = countNumberOfDocuments(inputFile)
141                     print "Total number of documents: "+str(numDocs)
142                     print "Total number of paragraphs: "+str(numPars)
143                     print "Extracting glossary data..."
144                     print 'without cogito...'
145                     docTerms, docLengths = extractTermData(inputFile)
146             termCount, termIDFs = generateGlossary(docTerms,numDocs,USE_COGITO)
147             termTFs = generateReport(docTerms, docLengths, termIDFs)
148             print "...done! ("+str(termCount)+" terms, "+str(len(termTFs))+" occs
                    extracted)"
149             f_out = open(outFile,"w")
150             glossaryData = (termTFs,termIDFs)     # list:  doc-code,term,tf (array),
                    term:idf (dict)
151             pickle.dump(glossaryData,f_out)
152             f_out.close()
153             print "Pickling OK!"
```

A.7 similarityComp.py

Codice sorgente di similarityComp.py, modulo in cui viene effettuato il calcolo della similarità e il ranking fusion.

```
1   # -*- coding: utf-8 -*-
2   import pickle
3   import locale
4   import math
5   from nltk.corpus import wordnet as wn
6   import numpy as np
7   import matplotlib.pyplot as plt
8
9   def disambiguateTerms(terms,verbose=False):
10          for t in terms:    # t is target term
11                  selSense = None
12                  selScore = 0.0
13                  for s_ti in wn.synsets(t, wn.NOUN):
14                          score_i = 0.0
15                          for w_j in terms:    # w_j word in t's context windows
```

```
16                                  if (t==w_j):
17                                          continue
18                                  bestScore = 0.0
19                                  for s_jk in wn.synsets(w_j, wn.NOUN):
20                                          tempScore = s_ti.path_similarity(s_jk)
21                                          if (tempScore>bestScore):
22                                                  bestScore=tempScore
23                                  score_i = score_i + bestScore
24                          if (verbose):
25                                  print t+" "+str(score_i) + " " + str(s_ti)+", "+
                                        s_ti.definition
26                          if (score_i>selScore):
27                                  selScore = score_i
28                                  selSense = s_ti
29                  if (not verbose):
30                          if (selSense is not None):
31                                  print t+": "+str(selSense)+", "+selSense.
                                        definition
32                          else:
33                                  print t+": --"
34
35  def wnSimilarity(term1, term2):     # similarity between 0 and 1
36          bestScore = 0.0
37          for s_t1 in wn.synsets(term1, wn.NOUN):
38                  for s_t2 in wn.synsets(term2, wn.NOUN):
39                          tempScore = s_t1.path_similarity(s_t2)
40                          if (tempScore>bestScore):
41                                  bestScore=tempScore
42          return bestScore
43
44  def glossSimilarity(term1, term2, ieeeDefs):
45          bestScore = 0
46          if (term1 in ieeeDefs) and (term2 in ieeeDefs):
47                  stemmedDefs1 = ieeeDefs[term1][1]
48                  stemmedDefs2 = ieeeDefs[term2][1]
49                  for def1 in stemmedDefs1:
50                          for def2 in stemmedDefs2:
51                                  tempScore = extGlossOverlap(def1,def2)
52                                  if (tempScore>bestScore):
53                                          bestScore=tempScore
54          return bestScore
55
56  def extGlossOverlap(def1, def2):
57          score = 0
58          d1 = def1.split()
59          d2 = def2.split()
60          i=0
61          while(i<len(d1)):   # cycle on def1 terms
62                  term1=d1[i]
63                  bestScore = 0
64                  j=0
65                  while(j<len(d2)):    # cycle on def2 terms
66                          term2=d2[j]
```

```
67                          if term1==term2:
68                                  length = getMatchLength(d1,i,d2,j)
69                                  if bestScore<length*length:
70                                          bestScore=length*length
71                                  i=i+length-1
72                                  j=j+length-1
73                          j=j+1
74                  score = score+bestScore
75                  i=i+1
76          return score
77
78  def getMatchLength(def1, i, def2, j):
79          l=1
80          while ((i+l < len(def1)) and (j+l < len(def2))):
81                  term1=def1[i+l]
82                  term2=def2[j+l]
83                  if term1==term2:
84                          l=l+1
85                  else:
86                          break
87          return l
88
89  def isWnRelated(term1, term2):
90          if term1 == term2:
91                  return True
92          if wnSimilarity(term1, term2) >= WN_THR:
93                  return True
94          return False
95
96  def isRelated(term1, term2, relTerms):
97          if term1 == term2:
98                  return True
99          if term1 in relTerms:
100                 if term2 in relTerms[term1]:
101                         return True
102         return False
103
104 def isWnSynonym(term1, term2, synTerms):
105         if term1 == term2:
106                 return True
107         if ((term1 in synTerms) and (term2 in synTerms)):  #AGGIUNTO
108                 syns1 = synTerms[term1]
109                 syns2 = synTerms[term2]
110                 if (term1 in syns2) or (term2 in syns1):
111                         return True
112         return False
113
114 def genAllWnRelated(termIDFs):
115         relTerms = {}
116         for term1 in sorted(iter(termIDFs)):
117                 #print ("TERM: "+term1)
118                 terms = getWnRel(term1)
119                 #print terms
```

```
120                 relTerms[term1]=terms
121         return relTerms
122
123 def genAllWnSyns(termIDFs):
124         synTerms = {}
125         for term1 in sorted(iter(termIDFs)):
126                 #print ("TERM: "+term1)
127                 terms = getWnSyns(term1)
128                 #print (terms)
129                 synTerms[term1]=terms
130         return synTerms
131
132 def getWnRel(term):
133         rel = []
134         for s in wn.synsets(term):
135                 for hype in s.hypernym_distances():
136                         he = hype[0].name.split('.')[0].replace('_',' ')
137                         if ((hype[1]<=2) and (he not in rel)and(he!=term)):
138                                 if (len(wn.synsets(he))==1):
139                                         rel.append(he.lower())
140                 for hypo in s.hyponyms():
141                         dist = 0
142                         for ho in hypo.hypernym_distances():
143                                 if s == ho[0]:
144                                         dist = ho[1]
145                         for ho in hypo.hypernym_distances():
146                                 h = ho[0].name.split('.')[0].replace('_',' ')
147                                 if ((ho[1]<= dist)and((dist-ho[1])<=2)and(h not
                                        in rel)and(h!=term)):
148                                         if (len(wn.synsets(h))==1):
149                                                 rel.append(h.lower())
150         rel = sorted(list(set(rel)))
151         return rel
152
153 def getWnSyns(term):
154         syns = []
155         for s in wn.synsets(term):
156                 for lemma in s.lemmas:
157                         if (len(wn.synsets(lemma.name.replace('_',' ')))==1):
158                                 syn = lemma.name.replace('_',' ')
159                                 syns.append(syn.lower())
160         syns = sorted(list(set(syns)))
161         return syns
162
163 def genInvertedIndex(termTFs):
164         invIndex = {}
165         for t in termTFs:
166                 doc = t[0]
167                 term = t[1]
168                 if (term in invIndex):
169                         invIndex[term].append(doc)
170                 else:
171                         list = []
```

```
172                         list.append(doc)
173                         invIndex[term]=list
174         return invIndex
175
176 def genDocTerms(termTFs):
177         docTerms = {}
178         for t in termTFs:
179                 doc = t[0]
180                 term = t[1]
181                 tf = t[2]
182                 if (doc in docTerms):
183                         docTerms[doc].append((term,tf))
184                 else:
185                         list = []
186                         list.append((term,tf))
187                         docTerms[doc]=list
188         return docTerms
189
190 def genIndexFile(glossaryFile,indexFile):
191         termTFs, termIDFs = readGlossary(glossaryFile)
192         invIndex = genInvertedIndex(termTFs)
193         docTerms = genDocTerms(termTFs)
194         synTerms = genAllWnSyns(termIDFs)
195         relTerms = genAllWnRelated(termIDFs)
196         f_out = open(indexFile,"w")
197         indexData = (invIndex, docTerms, synTerms, relTerms)    # term:termlist
                (dict), doc:docTerms (dict), term:synTermsList (dict), term:
                relTermsList (dict)
198         pickle.dump(indexData,f_out)
199         print "Done!"
200         f_out.close()
201
202 def ambitSimilaritySingleDocV2(terms1,termsTF2,termIDFs,synTerms,relTerms):
203         score = 0.0
204         for termTf1 in terms1:
205                 term1 = termTf1[0]
206                 tf1 = termTf1[1]
207                 w1=1
208                 if (USE_WEIGHTS):
209                         if (term1 in termIDFs):
210                                 idf1=termIDFs[term1]
211                                 w1=tf1*idf1
212                         else:
213                                 w1=NO_WEIGHT
214                 bestT1Score = 0
215                 for docTermTf2 in termsTF2:
216                         term2 = docTermTf2[0]
217                         tf2 = docTermTf2[1]
218                         if term2 not in termIDFs:
219                                 continue
220                         idf2=termIDFs[term2]
221                         w2=1
222                         if (USE_WEIGHTS):
```

```
223                                 w2=tf2*idf2
224                         if (term1==term2):    # equal terms
225                                 bestT1Score=(EQ_SCORE*w1*w2)
226                                 break
227                         if (USE_SYNS and isWnSynonym(term1,term2,synTerms)):
                                # synonym terms
228                                 if (bestT1Score<SYN_SCORE*w1*w2):
229                                         bestT1Score=SYN_SCORE*w1*w2
230                                 continue
231                         if ( USE_REL and isRelated(term1,term2,relTerms) ):    #
                              related terms
232                                 if (bestT1Score<REL_SCORE*w1*w2):
233                                         bestT1Score=REL_SCORE*w1*w2
234                 score = score+bestT1Score
235         return score
236
237 def ambitSimilarityV2(terms1,termTFs,termIDFs,synTerms,relTerms,invIndex,
    docTerms):
238         ranking = []
239         unionDocSet = set([])
240         i=0
241         for termTf1 in terms1:
242                 term1 = termTf1[0]
243                 if (term1 in invIndex):
244                         docSet = set(invIndex[term1])
245                         unionDocSet = unionDocSet.union(docSet)
246                 if (USE_SYNS):
247                         if (term1 in synTerms):
248                                 syns = synTerms[term1]
249                                 for syn in syns:
250                                         if (syn in invIndex):
251                                                 docSet = set(invIndex[syn])
252                                                 unionDocSet = unionDocSet.union(
                                                      docSet)
253                 if (USE_REL):
254                         if (term1 in relTerms):
255                                 rels = relTerms[term1]
256                                 for rel in rels:
257                                         if (rel in invIndex):
258                                                 docSet = set(invIndex[rel])
259                                                 unionDocSet = unionDocSet.union(
                                                      docSet)
260         docs = sorted(unionDocSet)
261         for doc in docs:
262                 termsTF2=docTerms[doc]
263                 score = ambitSimilaritySingleDocV2(terms1,termsTF2,termIDFs,
                        synTerms,relTerms)
264                 if (score>SCORE_THR):
265                         ranking.append((score,doc))
266         ranking = sorted(ranking,reverse=True)
267         return ranking
268
269 def printRanking(ranking):
```

```
270         for res in ranking:
271                 scoreFormatted = locale.format("%0.4f", res[0])
272                 print (res[1]+"\t"+scoreFormatted)
273
274 def plot_res(labels,data):
275         fig = plt.figure()
276         ax = fig.add_subplot(111)
277
278         ## necessary variables
279         ind = np.arange((len(labels)))          # the x locations for the groups
280         width = 0.5                       # the width of the bars
281
282         ## the bars
283         bar = ax.bar(ind, data, width,color='blue')
284                 # axes and labels
285         ax.set_title('Ranking')
286         ax.set_xlim(-width,len(ind)+width)
287         ax.set_ylim(0.00000000,max(data))
288         ax.set_ylabel('Score')
289         ax.set_xticks(ind+width)
290         c = 0
291         for i in data:
292                 labels[c] += "\n(" + str(i)+ ")"
293                 c += 1
294
295         xtickNames = ax.set_xticklabels(labels)
296         plt.setp(xtickNames, rotation=15, fontsize=10)
297         plt.show()
298
299 def evaluateResults(querySol,ranking):
300         queryRetr = []
301         precision = 0.0
302         recall = 0.0
303         print ranking
304         for tuple in ranking:
305                 doc = tuple[1]
306                 queryRetr.append(doc)
307         if (ONLY_FIRST and len(queryRetr)>len(querySol)):     # only first
                results are considered
308                 queryRetr=queryRetr[0:len(querySol)]
309         retrieved = set(queryRetr)     # precision and recall computation
310         relevant = set(querySol)
311         relRetr = retrieved.intersection(relevant)
312         if (len(retrieved) and len(relevant)>0):
313                 precision = (0.0 + len(relRetr))/len(retrieved)
314                 recall = (0.0 + len(relRetr))/len(relevant)
315         f = -1
316         if (recall+precision>0):
317                 f = 2*recall*precision/(recall+precision)
318         print ("")
319         print (str(len(retrieved))+" retrieved docs")
320         print (str(len(relevant))+" relevant docs")
321         print (str(len(relRetr))+" relevant retrieved docs")
```

```
322         print (str(precision)+" precision")
323         print (str(recall)+" recall")
324         print (str(f)+" f measure")
325         print ("")
326         numrel = 0.0    # precision at standard recall levels computation
327         numret = 0.0
328         precisions = []
329         for retr in queryRetr:
330                 numret=numret+1
331                 if retr in relevant:
332                         numrel=numrel+1
333                         print ("P("+str(numrel/len(relevant))+")="+str(numrel/
                                numret))
334                         precisions.append((numrel/len(relevant),numrel/numret))
335         i=1.0
336         precMax=0.0
337         print ("")
338         while (i>=0.0):
339                 for tuple in precisions:
340                         r=tuple[0]
341                         p=tuple[1]
342                         if (r>=(i-0.001) and precMax<p):
343                                 precMax=p
344                 print ("P("+str(i)+")="+str(precMax))
345                 print (locale.format("%0.4f", precMax))
346                 i=i-0.1
347                 if (i>0 and i<0.01):
348                         i=0.0
349         # ranking distance computation
350         dist=0.0
351         print("")
352         for retr in queryRetr:
353                 a=queryRetr.index(retr)
354                 b=len(querySol)
355                 if (retr in querySol):
356                         b=querySol.index(retr)
357                 dist=dist+math.fabs(a-b)
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         profIPTCs_print = []
411         for profId in profFile:
412                 i=0
413                 for IPTC in profId[1]:
414                         if i==0 :
415                                 profIPTCs.append(IPTC)
416                                 profIPTCs_print.append((profId[0],IPTC))
417                         i+=1
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                 for IPTC in profIPTC:
425                         if (mergedIPTCs == []):
426                                 mergedIPTCs.append(list(IPTC))
427                         else:
428                                 isIn=False
429                                 for merged in mergedIPTCs:
430                                         if (merged[0] == IPTC[0]):
431                                                 isIn = True
432                                                 score = float(merged[1]) + float
                                                     (IPTC[1])
433                                                 merged[1] = str(score)
434                                 if (isIn==False):
435                                         mergedIPTCs.append(list(IPTC))
436         sortedIPTCs=sorted(mergedIPTCs, key = lambda x: float(x[1]),reverse=True
                )
437         return sortedIPTCs
438
439 #estrae il valore di icf (inverse class frequency) dalla collezione di documenti
440 def icfIPTCs(docIPTCs):
441         cfIPTCs = []
442         numDoc = 0
443         for doc in docIPTCs:
444                 numDoc += 1
445                 for IPTC in doc[1]:
446                         if IPTC:
447                                 if (cfIPTCs == []):
448                                         t = (IPTC[0],IPTC[1],1)
449                                         cfIPTCs.append(list(t))
450                                 else:
451                                         isIn=False
452                                         for merged in cfIPTCs:
453                                                 if (merged[0] == IPTC[0]):
454                                                         isIn = True
455                                                         merged[2] += 1
456                                                         score = float(merged[1])
                                                             + float(IPTC[1])
457                                                         merged[1] = str(score)
458
459                                         if (isIn==False):
460                                                 t = (IPTC[0],IPTC[1],1)
461                                                 cfIPTCs.append(list(t))
462
463         icfIPTC = []
464         for cf in cfIPTCs:
465                 icf = float(numDoc) / float(cf[2])
466                 l = (cf[0],icf,cf[1])
467                 icfIPTC.append(list(l))
468         sortedIcfIPTC=sorted(icfIPTC, key = lambda x: float(x[1]),reverse=True)
469         return icfIPTC
470
```

```
471 #Calcola gli score facendo la differenza [-log10(numero di passi tra le due
        classi/2*h)],
472 #moltiplicando poi questo score  nel caso in cui
473 #la differenza di cammino sia uguale a 1(non c'è differenza tra le due classi
        IPTC)
474 #per lo score della classe IPTC del
475 #profilo estratto da cogito.
476 def similarityIPTC(documents,profiles,icfIPTC):
477         ranking = []
478         iProf = 1
479         for profile in profiles:
480                 for document in documents:
481                         iDoc=0
482                         for IPTC in document[1]:
483                                 curProf = profile[0].split('/')
484                                 curDoc = IPTC[0].split('/')
485                                 intersection = set(curProf).intersection(curDoc)
486                                 iPath = 1
487                                 score = 0.0
488                                 for profPath in curProf:
489                                         if profPath not in intersection:
490                                                 iPath+=1
491                                 for docPath in curDoc:
492                                         if docPath not in intersection:
493                                                 iPath+=1
494                                 score = -math.log10(iPath/(2.0*5))
495                                 if iPath==1:
496                                         w = float(profile[1])
497                                         for icf in icfIPTC:
498                                                 if icf[0] == profile[0]:
499                                                         w *= icf[1]
500                                         score *= w
501                                 if ranking == []:
502                                         ranking.append([score,document[0]])
503                                 else:
504                                         isIn = False
505                                         for doc in ranking:
506                                                 if doc[1] == document[0]:
507                                                         isIn=True
508                                                         doc[0]=(doc[0]+score)
509                                         if isIn==False:
510                                                 ranking.append([score,document
                                                        [0]])
511
512                                 iDoc+=1
513                 iProf+=1
514         sortedRank=sorted(ranking, key = lambda x: float(x[0]),reverse=True)
515         return sortedRank
516
517 #similarità tra le classi IPTC che considera solo le classi uguali tra di loro
518 def similarityIPTC_raw(documents,profiles):
519         ranking = []
520         iProf = 1
```

```
521         for profile in profiles:
522                 for document in documents:
523                         i=0
524                         for IPTC in document[1]:
525                                 if ((IPTC[0]==profile[0])):
526                                         if ranking == []:
527                                                 ranking.append([float(IPTC[1]),
                                                        document[0]])
528                                         else:
529                                                 isIn = False
530                                                 for doc in ranking:
531                                                         if doc[1] == document
                                                                [0]:
532                                                             isIn=True
533                                                             score = (float(
                                                                    doc[0])+
                                                                    float(IPTC
                                                                    [1]))
534                                                         doc[0]=score
535                                                 if isIn==False:
536                                                         ranking.append([float(
                                                                IPTC[1]),document
                                                                [0]])
537                                 i+=1
538                 iProf+=1
539         sortedRank=sorted(ranking, key = lambda x: float(x[0]),reverse=True)
540         return sortedRank
541
542 #somma gli score dei paragrafi di ogni documento
543 def paragraphFusion(paragRank):
544         ranking = []
545         totScore = 0.0
546         for parag in paragRank:
547                 totScore += parag[0]
548                 curDoc = parag[1].split('-')[0]
549                 if ranking == []:
550                         ranking.append([parag[0],curDoc])
551                 else:
552                         isIn=False
553                         for doc in ranking:
554                                 if doc[1] == curDoc:
555                                         isIn=True
556                                         doc[0] += parag[0]
557                         if isIn==False:
558                                 ranking.append([parag[0],curDoc])
559         sortedRank=sorted(ranking, key = lambda x: float(x[0]),reverse=True)
560         return sortedRank
561
562 #Funzione di ranking fusion (Rank Fusion)
563 def rankingFusion_rankFusion(rank1,ranking2, w1, w2):
564         ranking=[]
565         EWF1 = len(rank1)+1
566         EWF2 = len(ranking2)+1
```

```
567         count1 = 0
568         for doc1 in rank1:
569                 count1+=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 = (1.0 - ((count1 - 1.0) / EWF1)) + (1.0 -
                                        ((count2 - 1.0) / EWF2) )
577                                 ranking.append([num,doc1[1]])
578                 if isIn==False:
579                         num = (1.0 - ((count1 - 1.0) / EWF1))
580                         ranking.append([num,doc1[1]])
581
582
583         count2 = 0
584         doc = [r[1] for r in ranking]
585         for doc2 in ranking2:
586                 count2 += 1
587                 if doc2[1] not in doc:
588                         num = (1.0 - ((count2 - 1.0) / EWF2))
589                         ranking.append([num,doc2[1]])
590
591         sortedRank=sorted(ranking, key = lambda x: float(x[0]),reverse=True)
592         return sortedRank
593
594 #Funzione di ranking fusion (Score Fusion)
595 def rankingFusion_scoreFusion(rank1,ranking2, w1, w2):
596         ranking=[]
597         count1 = 0
598         for doc1 in rank1:
599                 count1+=1
600                 isIn=False
601                 count2 =0
602                 for doc2 in ranking2:
603                         count2+=1
604                         if doc1[1]==doc2[1]:
605                                 isIn=True
606                                 num = 2 * ((w1 * doc1[0]) + (w2 * doc2[0]))
607                                 ranking.append([num,doc1[1]])
608                 if isIn==False:
609                         num = w1 * doc1[0]
610                         ranking.append([num,doc1[1]])
611
612         count2 = 0
613         doc = [r[1] for r in ranking]
614         for doc2 in ranking2:
615                 count2 += 1
616                 if doc2[1] not in doc:
617                         num = w2 * doc2[0]
618                         ranking.append([num,doc2[1]])
```

```
619
620         sortedRank=sorted(ranking, key = lambda x: float(x[0]),reverse=True)
621         return sortedRank
622
623 #ritorna in ordine di ranking i paragrafi dei documenti all'interno di un
        ranking dei paragrafi
624 def returnParagraph(ranking,rankParag):
625         rankedParag = []
626         for doc in ranking:
627                 for parag in rankParag:
628                         curDoc = parag[1].split('-')[0]
629                         if curDoc==doc[1]:
630                                 rankedParag.append(parag)
631         return rankedParag
632
633 def normalizeRank(ranking):
634         normalized=[]
635         totScore = 0.0
636         for rankItem in ranking:
637                 totScore += rankItem[0]
638         for rankItem in ranking:
639                 normalized.append([rankItem[0]/totScore,rankItem[1]])
640         return normalized
641
642 def totScoreRank(ranking):
643         totScore = 0.0
644         for rankItem in ranking:
645                 totScore += rankItem[0]
646         return totScore
647
648 def rankweight(score1, score2):
649         tot = score1 + score2
650         w1 = round(score1/tot, 1)
651         w2 = 1.0 - w1
652         return w1, w2
653
654 USE_SYNS = True
655 USE_REL = True
656 USE_WEIGHTS = True
657 NO_WEIGHT = 0.5
658 EQ_SCORE = 1.0
659 SYN_SCORE = 1.0
660 REL_SCORE = 0.7
661 WN_THR = 0.3
662 SCORE_THR = 0
663 ONLY_FIRST = False
```

Bibliografia

[1] N. J. Belkin. Anomalous states of knowledge as a basis for information retrieval. *Canadian Journal of Information Science (5): 133-143*, 1980.

[2] S. Bergamaschi, R. Martoglia, and S. Sorrentino. A Semantic Method for Searching Knowledge in a Software Development Context. In *Proceedings of the 20th Italian Symposium on Advanced Database System (SEBD 2012), pp. 115-122*, 2012.

[3] R. Bierig and A. Goker. Time, location, and interest: an empirical and user-centred study. In *First symposium on information interaction in context (IIiX)*, 2006.

[4] P. J. Brown. *The Stick-e Document: A Framework For Creating Context-aware Applications*. 1996.

[5] A. K. Dey and G. D. Abowd. *Towards a Better Understanding of Context and Context-Awareness*. 1999.

[6] E. Drymonas. Exploring multi-word similarity measures for Information Retrieval applications: the T-SRM method.

[7] W. B. Frakes and R. Baeza-Yates. *Information Retrieval: Data Structures and Algorithms*. Prentice Hall, 1992.

[8] E. Garfield. A tribute to Calvin N. Mooers, a pioneer of information retrieval. *The Scientist, 11(6), p.9*, 1997.

[9] D. J. Harper and D. Kelly. Contextual relevance feedback. In *Proceedings of the 1st Symposium on Information Interaction in Context (IIiX)*, 2006.

[10] W. Hersh. *Information Retrieval: A Health and Biomedical Perspective*. Springer-Verlag, 2009.

[11] R. Hull, P. Neaves, and J. Bedford-Roberts. *Towards Situated Computing*. 1997.

[12] P. Ingwersen. Cognitive perspectives of information retrieval interaction: elements of a cognitive ir theory. *Journal of Documentation 52(1): 3-50*, 1996.

[13] P. Ingwersen and K. Jarvelin. *The Turn: Integration of Information Seeking and Retrieval in Context*. Springer, 2005.

[14] C. Leacock and M. Chodorow. Combining local context and WordNet similarity for word sense identification. In *WordNet: An electronic lexical database*, 1998.

[15] C. T. Lopes. Context Features and their use in Information Retrieval. In *Third BCS-IRSG Symposium on Future Directions in Information Access*, 2009.

[16] C. D. Manning, P. Raghavan, and H. Schutze. *Introduction to Information Retrieval*. Cambridge University Press, 2008.

[17] C. N. Mooers. Zatocoding applied to mechanical organization of knowledge. *American Documentation*, 1951.

[18] R. Nuray and F. Can. Automatic ranking of information retrieval systems using data fusion. *Information Processing and Management*, 2006.

[19] J. Pascoe. *Adding Generic Contextual Capabilities to Wearable Computers*. 1998.

[20] M. E. Renda and U. Straccia. Web metasearch: rank vs. score based rank aggregation methods. In *SAC '03 Proceedings of the 2003 ACM symposium on Applied computing*, 2003.

[21] N. Ryan, J. Pascoe, and D. Morse. *Enhanced Reality Fieldwork: the Context Aware Archaeological Assistant*. 1997.

[22] T. Saracevic. Evaluation of evaluation in information retrieval. In *SIGIR '95 Proceedings of the 18th annual international ACM SIGIR conference on Research and development in information retrieval*, 1995.

[23] T. Saracevic. The stratified model of information retrieval interaction: Extension and applications. In *Proceedings of the American Society for Information Science, Vol. 34, pp. 313-327*, 1997.

[24] B. Schilit, N. Adams, and R. Want. *Context-Aware Computing Applications*. 1994.

[25] B. Schilit and M. M. Theimer. *Disseminating Active Map Information to Mobile Hosts*. 1994.

[26] X. Shen, B. Tan, and C. Zhai. Context-sensitive information retrieval using implicit feedback. In *SIGIR '05 Proceedings of the 28th annual international ACM SIGIR conference on Research and development in information retrieval*, 2005.