UNIVERSITÀ DEGLI STUDI
DI PADOVA
FACOLTÀ DI INGEGNERIA

DEPARTMENT OF INFORMATION ENGINEERING
UNIVERSITY OF PADOVA

Tesi di Laurea Magistrale in

Ingegneria Informatica

Riconoscere ed elaborare interrogazioni prolisse

Laureanda: Federica Moro

Relatore: Prof. Massimo Melucci
Co-relatore: Dr. Emanuele Di Buccio

Anno Accademico 2011-2012

Indice

Elenco delle figure

Elenco delle tabelle

1

Introduzione

1.1 Argomenti trattati

L'*Information Retrieval* (IR) identifica tutte le attività utilizzate per scegliere, da una collezione di dati, quelli utili o di interesse in relazione ad una specifica esigenza informativa (41). Tale disciplina è la tecnologia alla base dei motori di ricerca, i quali non sono però in grado di comprendere l'esigenza informativa dell'utente se essa non è esplicitata mediante un'interrogazione[1] scritta in un linguaggio elaborabile dal sistema di reperimento stesso (vedi fig. 1.1).

La maggior parte delle interrogazioni che vengono proposte ai motori di ricerca sono composte da poche *parole chiave* e hanno una lunghezza compresa tra tre e sei parole (27). Sono in numero considerevole (più del 15% rispetto al totale), le **interrogazioni prolisse** (3), scritte in *linguaggio naturale*, piuttosto che selezionando un ristretto numero di parole chiave, o che contengono termini ridondanti e inutili ai fini della ricerca. Alcuni termini contenuti nella query potrebbero addirittura introdurre rumore, riducendo l'efficacia del reperimento.

Il riconoscimento e l'elaborazione di tale tipo di interrogazioni costituisce l'argomento centrale di questa tesi.

Poiché in letteratura viene spesso attribuita all'espressione *verbose queries* lo stesso significato di *long queries*, ossia si utilizzano i termini "prolisso" e "lungo" come sinonimi, per rendere più chiara la trattazione, nel presente documento si farà riferimento alla seguente definizione di "prolisso": *troppo lungo e particolareggiato o che utilizza*

[1] spesso nel seguito si utilizzerà il termine inglese *query*.

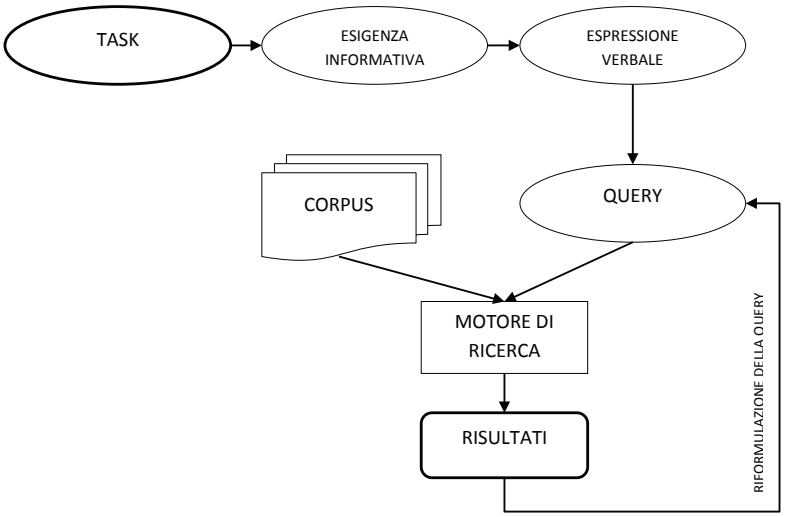


Figura 1.1: Modello classico per l'*information retrieval*, così come viene rappresentato da Broder (7).

un numero di parole superiore al necessario; ridondante; riassumibile senza perdita di informazione. Si noti come questa definizione non implica che una query per essere prolissa debba necessariamente essere lunga.

1.2 Obiettivi della tesi

Gli obiettivi principali della tesi sono i seguenti:

- proporre un metodo automatico per il riconoscimento di interrogazioni prolisse;
- verificare in che misura l'utilizzo di interrogazioni prolisse influisca sulle performance di un sistema di reperimento;
- progettare e valutare sperimentalmente una metodologia per elaborare le interrogazioni prolisse in modo da migliorare le prestazioni del sistema.

Per fare ciò, una volta scelta la collezione sperimentale **TREC Robust 2004** (conosciuta per contenere esigenze informative *difficili* per i comuni motori di ricerca), di

cui si hanno a disposizione i giudizi di rilevanza[1], si è deciso di costruire 4 insiemi di interrogazioni: corte, lunghe, lunghe e prolisse, corte e prolisse. Tali interrogazioni sono state sottoposte a dei giudici per stabilirne l'effettiva classe di appartenenza. L'utilizzo di valutazioni umane viene spesso utilizzato in ambiti di ricerca in cui non è possibile stabilire automaticamente un determinato dato (vedi ad esempio (2)), ma, per quanto riguarda lo studio di interrogazioni prolisse, le ricerche nel settore tendono a considerare esclusivamente interrogazioni lunghe, senza valutarne l'effettivo contenuto informativo (si rimanda a (3), (23), (46)).

Per rendere possibile il riconoscimento di interrogazioni prolisse si propone un predittore ottenuto utilizzando un ***C4.5 decision tree***. Il *dataset* sulla base del quale verrà costruito tale predittore conterrà tra le *feature* una misura calcolata utilizzando una variante del **metodo di Lesk** (34).

Per migliorare le performance nell'elaborazione di query prolisse, si propone un nuovo approccio che sfrutta le relazioni semantiche in esse contenute. A tal fine si utilizzano le informazioni contenute in **ConceptNet 5**, un database di relazioni semantiche, di cui si propone un'analisi, la quale, visto il recente rilascio del tool (maggio 2012), potrebbe risultare utile in contesti diversi. Nella presente tesi vengono implementati vari algoritmi per l'elaborazione delle interrogazioni:

- un algoritmo che estrae i sinonimi dei termini contenuti nelle interrogazioni;
- varie versioni di un algoritmo per il *topic gisting*, ossia per l'estrazione del succo del discorso.

In particolare, quest'ultimo approccio per l'elaborazione di query prolisse si discosta da quelli presenti in letteratura, che propongono una riduzione delle query basata sulla selezione di alcuni dei termini in esse contenuti (vedi ad esempio (3) o (46)).

Una volta indicizzati i documenti della collezione **TREC Robust 2004**, utilizzando il modello di reperimento su cui è basato **Apache Lucene**, si verificherà in che misura l'utilizzo di interrogazioni prolisse (corte e lunghe) influisca sulle performance del sistema di reperimento. Si vuole quindi verificare l'efficacia dei metodi proposti fornendo in input al sistema di reperimento i vari tipi di interrogazioni, dimostrando la significatività statistica dell'aumento di performance.

[1]La *rilevanza* è la proprietà di un documento di contenere dell'informazione importante, utile o necessaria a soddisfare l'esigenza informativa di un utente.

1.3 Struttura della tesi

Il presente documento è articolato nei seguenti capitoli, di cui si riporta una breve presentazione:

Stato dell'arte: scopo di questo capitolo è quello di motivare lo studio e l'utilizzo di interrogazioni prolisse. Nella prima metà del capitolo ci si sofferma sull'**evoluzione dei sistemi di reperimento** nel corso degli ultimi decenni e sulle **varie applicazioni** in cui l'utilizzo di questo tipo di query risulta utile. Nella seconda parte del capitolo si propone una panoramica delle **ricerche nel settore**, soffermandosi sui metodi utilizzati più frequentemente per ridurre le interrogazioni e i modelli di reperimento adottati per elaborarle.

Metodologia: scopo di questo capitolo è quello di presentare i metodi e gli algoritmi utilizzati per valutare in che misura l'utilizzo di interrogazioni prolisse influisca su un sistema di reperimento e in che modo sia possibile sfruttare le relazioni semantiche presenti all'interno dell'interrogazione per migliorarne l'efficacia. Dopo aver esposto il metodo di indicizzazione e il modello di reperimento (basato su **Lucene**), si presenterà **ConceptNet 5**, un database di relazioni semantiche, e si esporrà come sia possibile utilizzarlo per elaborare le interrogazioni. Si presenterà come accedere ad una copia personale dei dati messi a disposizione dal tool attraverso **Solr**. Nell'ultima parte del capitolo si proporrà la costruzione di un classificatore attraverso **Weka**, per stabilire se un'interrogazione sia prolissa o meno. Una delle *feature* presa in esame verrà calcolata utilizzando una variante del **metodo di Lesk**.

Sperimentazione: scopo di questo capitolo è quello di descrivere nel dettaglio la sperimentazione effettuata, concentrandosi in modo particolare sulla strategia utilizzata per la scelta e la costruzione della collezione sperimentale. A partire dai *topic* della **collezione TREC Robust 2004** sono stati costruiti quattro insiemi di interrogazioni: corte, lunghe, lunghe e prolisse, corte e prolisse. Tali interrogazioni sono state sottoposte a valutazione umana per stabilirne la classe di appartenenza. Le valutazioni umane sono state utilizzate anche per allenare il predittore di prolissità costruito con Weka. Nell'ultima parte del capitolo vengono presentati

i risultati, su cui è stato effettuato un **test di significatività**, in particolare il test dei segni.

Conclusioni: sulla base dei valori ottenuti in fase di sperimentazione, si riporta in questo capitolo una panoramica sui risultati. Si propongono quindi alcuni spunti per sperimentazioni future.

2

Stato dell'arte

Scopo di questo capitolo è quello di introdurre il lettore ad alcuni concetti di base dell'information retrieval, in modo da rendere chiare le problematiche legate all'elaborazione di interrogazioni prolisse. In particolare, si vuole motivare lo studio di tale tipo di query, mettendo in luce le possibili applicazioni.

Dopo aver presentato una breve panoramica sull'evoluzione storica dei sistemi di reperimento, dagli anni '70 fino ad arrivare ai motori di ricerca attuali, prestando attenzione al tipo di interrogazioni utilizzate, si esplorerà, almeno in parte, il contesto applicativo delle query prolisse. Successivamente, verranno presentati alcuni degli approcci di ricerca presenti in letteratura, mettendo in risalto quali strategie possono essere utilizzate per l'elaborazione (e, in particolare, per la riduzione) delle interrogazioni, e quali modelli di reperimento vengono implementati più frequentemente.

2.1 Un po' di storia: dagli anni '70 ad oggi

Nel corso degli anni la struttura delle query date in input alla maggior parte dei sistemi di reperimento è significativamente cambiata. Negli anni '70 i principali motori di ricerca erano basati su **logica booleana** e le interrogazioni erano caratterizzate da una struttura complessa, la cui formulazione richiedeva dimestichezza con la sintassi e una certa pratica per effettuare le ricerche. Per questo motivo, in genere, solamente personale qualificato era in grado di interagire con il sistema in modo efficace e spesso era necessario riformulare la query più volte per renderla adatta a reperire i documenti rilevanti ai fini di soddisfare l'esigenza informativa.

A partire dal 1994, con la diffusione di Internet e dei motori di ricerca del WWW, si è passati alla formulazione di query più semplici, generalmente corte e composte da poche *parole chiave*. A discapito dell'apparente semplicità di questo tipo di interrogazioni, dal punto di vista dell'utente la scelta delle parole chiave adatte ad esprimere l'esigenza informativa può risultare un'operazione non banale e, nel caso di esigenze informative particolarmente complesse, addirittura impossibile. Dal punto di vista del sistema invece, l'utilizzo di pochi termini, che in generale risulta più efficace (3), può portare all'impossibilità di distinguere il contesto e pertanto di selezionare i documenti rilevanti. Sono state sviluppate tecniche atte ad espandere le query e basate sul contesto (*query expansion* e *context-based profiles*) per cercare di porre rimedio a questo tipo di problematica, ma i successi ottenuti sono stati limitati (11).

Di recente, piuttosto che cercare di inferire l'intenzione dell'utente a partire da query molto corte, si è cercato di migliorare l'elaborazione delle query e i modelli di reperimento per interrogazioni lunghe. In quest'ottica, si trasformano query lunghe o prolisse, ossia query che utilizzano un numero di termini superiore a quello effettivamente necessario per esprimere l'esigenza informativa, in una o più interrogazioni, utilizzando modelli probabilistici per la generazione automatica. In altri casi invece, si preferisce far riferimento ad archivi di interrogazioni per scomporre le interrogazioni complesse in più query semplici. I vari modelli per la trasformazione verranno descritti nei paragrafi successivi.

Ricerca	**Query**
1970 (Boolean search)	NEGLECT! FAIL! NEGLIG! /5 MAINT! REPAIR! /P NAVIGAT! /5 AID EQUIP! LIGHT BUOY "CHANNEL MARKER"
1994 (web search)	negligence navigation aids
2005 (CQA)	Are there any cases which discuss negligent maintenance or failure to maintain aids to navigation such as lights, buoys, or channel markers?

Tabella 2.1: Esempio di come una stessa esigenza informativa viene espressa in maniera differente a seconda del contesto.

Per rendere più chiaro al lettore l'evoluzione nella struttura delle query presentata in questo paragrafo e che ha avuto luogo negli ultimi 30-40 anni, si riporta in tabella 2.1 un esempio (11) di come viene espressa la stessa esigenza informativa in contesti diversi. Tale evoluzione nel formato delle query può essere fatta risalire a diversi fattori:

- tipo di sistema: ad esempio, sistemi di reperimento booleano utlizzano un linguaggio di tipo booleano per le query;
- fattori umani: query basate su parole chiave sono più facili da scrivere rispetto a quelle booleane per la maggior parte degli utenti; allo stesso modo, risulta più immediato per gli utenti esprimere un'esigenza informativa in linguaggio naturale piuttosto che scegliendo delle parole chiave che possano riassumerla.
- capacità del sistema: ad esempio, query lunghe in linguaggio naturale non risultano efficaci se date in input agli attuali motori di ricerca, ma vengono comunemente usate in applicazioni di tipo *collaborative question answering* (CQA).

L'utilizzo di interrogazioni prolisse è naturale se si considerano alcune applicazioni che si stanno diffondendo di recente (in particolare è possibile citare la *collaborative question answering* (CQA), a cui è stata dedicata la sez. 2.3.2.3, e la *forum search*) e in molti casi è il modo migliore per esprimere esigenze informative complesse o con un elevato grado di specificità. È inoltre possibile osservare che, nel caso di applicazioni che sfruttano il riconoscimento del linguaggio o ricerca in base al contesto, l'utilizzo di motori di ricerca ottimizzati per elaborare interrogazioni prolisse è particolarmente indicato.

2.2 Motori di ricerca attuali

Facendo riferimento agli studi condotti da Bendersky e Croft (4) sul numero di query lunghe in un query log a larga scala (MSN Query Log), è possibile dimostrare come le query estese siano in numero molto minore rispetto alle query corte. I grafici in fig.2.1 e 2.2 mostrano come per un log di circa 15 milioni di query il 90.3% delle interrogazioni hanno una lunghezza minore di 5 parole, mentre il 99.9% sono composte da meno di 13.

Anche se la maggior parte delle interrogazioni presentate ai motori di ricerca ha una lunghezza che varia da uno a tre termini, negli ultimi anni si sta osservando un

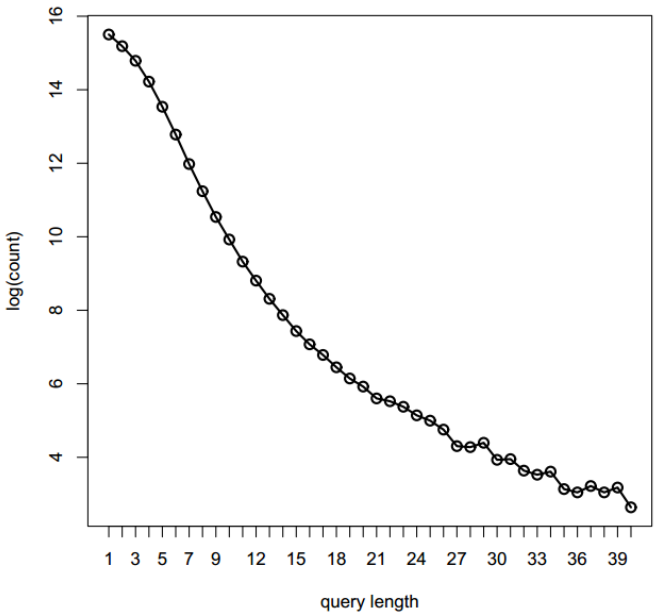


Figura 2.1: Analisi della lunghezza delle query in un log di circa 15 milioni di interrogazioni. Il 99,9% di esse è composta da meno di 12 parole. La lunghezza attesa è di 2,4 parole.

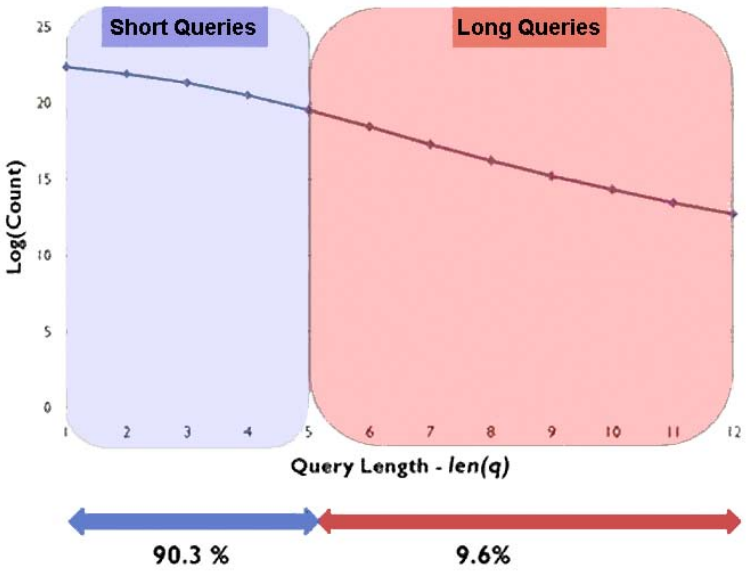


Figura 2.2: Analisi della lunghezza delle query in un log di circa 15 milioni di interrogazioni. Il 90,3% di esse è composta da meno di 5 parole.

graduale aumento della lunghezza media (27). L'utilizzo di query più lunghe, come già accennato, rispecchia il bisogno di soddisfare esigenze informative più complesse, o comunque l'impossibilità di selezionare delle parole chiave per riassumere l'interrogazione. Sfortunatamente, le prestazioni della maggior parte dei motori di ricerca accademici e commerciali peggiorano nel caso di interrogazioni lunghe. Per esempio, una query tipo *ideas for breakfast menu for a morning staff meeting*, pur rappresentando la vera esigenza informativa dell'utente, viene elaborata meglio dalla maggior parte dei motori di ricerca se tradotta in *breakfast meeting menu ideas*. L'atto di *ridurre* l'interrogazione originale in una versione più corta nel corso di una sessione di ricerca viene generalmente lasciato all'utente.

Tra gli studi che hanno dimostrato come l'utilizzo di query corte sia in molti casi più efficace si possono citare le ricerche sui campi *title* e *description* dei topic TREC condotte da Bendersky e Croft (3) e a quelle sulla ricerca in archivi di Q&A condotte da Xue e Croft (66), entrambi risalenti al 2008.

I principali motori di ricerca generalisti, come ad esempio Google e Bing, stanno tentando di introdurre in vario modo la semantica nelle rispettive procedure di indicizzazione e ricerca (18), in modo da migliorare le prestazioni nel caso di interrogazioni scritte in linguaggio naturale. Bing, in particolare, ha introdotto nella versione americana del suo motore la funzione di analisi semantica delle stringhe di ricerca, che permette di esprimere le stesse sotto forma di domande. Lo scarso impatto sui risultati delle ricerche di tali tecniche è da attribuirsi all'approccio adottato, ovvero alla volontà di non stravolgere i modelli di ricerca consolidati, cercando di creare nuove funzionalità semantiche sullo strato preesistente. I tradizionali modelli di ricerca si basano principalmente su modelli probabilistici che ignorano quasi totalmente il contesto semantico. Ignorando tale contesto risulta difficile elaborare efficacemente query prolisse.

2.3 Contesto applicativo

Negli ultimi anni la tecnologia alla base di Internet ha fatto progressi sorprendenti, al punto che si inizia a parlare di *Web pervasivo*. Inoltre, l'avvento di tecnologie per il riconoscimento di comandi complessi dati a voce a dispositivi telefonici mobili permette, a differenza dei decenni scorsi, di associare le parole pronunciate alle funzioni di gestione di comandi del dispositivo.

Spesso le esigenze informative sono complesse, ossia una volte espresse, si manifestano in interrogazioni scritte o parlate, articolate, prolisse e rumorose. La tecnologia attuale richiede all'utente di esprimere mediante interrogazioni molto brevi anche esigenze informative complesse. In questo modo, se da una parte si evita l'introduzione di rumore, dall'altra non è possibile recuperare tutte e sole le informazioni rilevanti, in quanto le interrogazioni sono prive dell'articolazione necessaria.

Poichè le persone sono abituate ad esprimersi in *linguaggio naturale*, risulta spesso difficile riassumere le esigenze informative in maniere sintetica ma espressiva. L'uso di query prolisse aiuta gli utenti a esprimere il loro bisogno informativo eliminando lo sforzo di dover scegliere delle parole chiave. D'altra parte però, i motori di ricerca attuali non hanno buone prestazioni se ricevono in input query molto lunghe e complesse.

Esempi di esigenze informative complesse sono quelli osservabili da utenti "esigenti" che utilizzano tablet o smartphone, tramite cui effettuano prenotazioni di viaggi, leggono numerose e-mail in breve tempo, o cercano rassegne sintetiche su argomenti di interesse immediato. Quello che si vorrebbe da anni arrivare ad ottenere, attraverso il processo tecnologico, è una figura virtuale di assistente personale, che sia in grado di soddisfare le esigenze tipiche degli utenti "esigenti" (38).

Riassumendo, si riportano nel seguito le principali motivazioni che spingono all'utilizzo di interrogazione prolisse:

- Spesso la selezione di parole chiave è difficile per l'utente (basti pensare a siti quali `SearchCloud.net`, che forniscono all'utente strumenti di supporto nella selezione delle *keyword*, vista la difficoltà di tale processo);

- Spesso l'utente riformula la propria esigenza informativa più volta prima che il sistema riesca a soddisfarla. Uno studio condotto da Lau e Horvitz (29) ha dimostrato come questo processo di riformulazione e raffinamento porta generalmente alla scrittura di interrogazioni di lunghezza sempre maggiore;

- In molti casi la lunghezza dell'interrogazione è correlata alla specificità dell'esigenza informativa (47).

2.3.1 Principali tipologie di utenti

Date le premesse sopra riportate, è possibile distinguere principalmente due tipologie di utenti che possono trarre particolare vantaggio dalla possibilità di esprimere la loro esigenza informativa attraverso query prolisse:

- utenti inesperti, che hanno particolare difficoltà nello scegliere le parole chiave e che si trovano più a loro agio ad esprimersi in linguaggio naturale, che spesso può risultare ridondante e introdurre rumore nella ricerca;

- utenti "esigenti", che hanno la necessità di effettuare ricerche complesse, per le quali l'utilizzo di query sintetiche non è adatto, in quanto non permettono di specificare tutti i parametri necessari ai fini del reperimento dei documenti di interesse.

2.3.2 Applicazioni

Quando si parla di motori di ricerca si tende istintivamente a pensare unicamente a quelli che si utilizzano per effettuare ricerche nel WWW, come ad esempio Google. Ogni giorno abbiamo la prova che tali motori di ricerca funzionano bene se ricevono in input query brevi, mentre sono poco efficaci nel caso di interrogazioni lunghe. Quello che si vuole sottolineare in questo paragrafo è come siano moltissime le applicazioni che richiedono di effettuare ricerche e molte di esse utilizzano motori di ricerca, strutturalmente diversi da quelli utilizzati per la ricerca sul web. Giusto per citare degli esempi si riportano nel seguito alcuni tipi di ricerca diversi dalla *web search*: *desktop search*, *enterprise search*, *vertical search*, *social search*, *forum search*, QA, FAQ e CQA, *product search*, *entity and expert search*, *literature search*, *media search*, *database/XML search*, ecc... Quello che queste applicazioni hanno in comune è l'obiettivo di effettuare ricerche efficaci in maniera efficente. Inoltre, devono trattare testo e altri tipi di media con semantiche inesatte, ossia devono gestire rappresentazioni inaccurate o quelli che vengono detti *vocabulary mismatch*.

Nei prossimi paragrafi si propone una selezione di alcune applicazioni fortemente correlate all'utilizzo di query prolisse che a parere personale risultano particolarmente interessanti.

2.3.2.1 OCR & Speech Recognition

I sistemi di riconoscimento ottico dei caratteri, detti anche OCR (dall'inglese *optical character recognition*), analogamente ai sistemi di riconoscimento vocale, producono testo rumoroso, ossia testo contenente errori rispetto all'originale stampato o registrato. Con un buon modello di reperimento, l'efficacia della ricerca non è significativamente influenzata negativamente dal rumore, grazie principalmente alla ridondanza del testo, che ne permette una parziale "correzione". In tale contesto risulta evidente come l'utilizzo di testi corti possa portare a problemi nel reperimento, mentre al contrario testi lunghi permettano risultati migliori.

Per dare un'idea del genere di errori che si possono riscontrare in fase di trascrizione, si può fare riferimento alla figura 2.3 per quanto riguarda i sistemi OCR e alla figura 2.4 per sistemi di riconoscimento vocale (9).

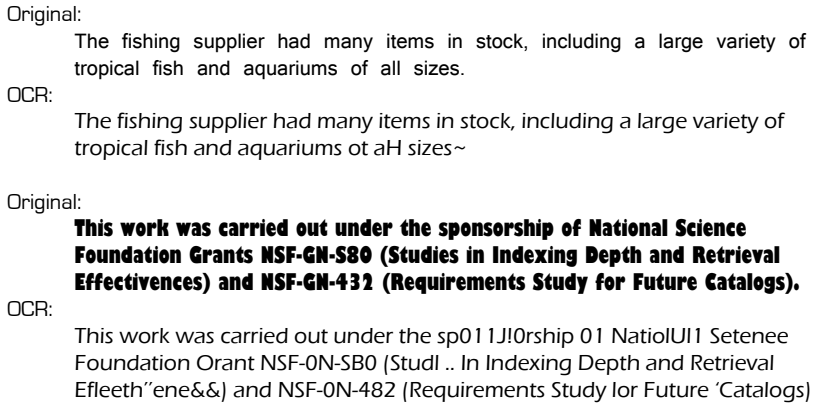


Figura 2.3: Esempi di errori di trascrizione in sistemi OCR.

2.3.2.2 Traduzioni e cross-language retrieval

La *cross-language information retrieval* (CLIR) è il campo dell'IR che si occupa del reperimento di documenti scritti in lingue diverse. Più nello specifico, data un'interrogazione in una lingua, si ricercano documenti in un'altra o più lingue. Poiché in generale i termini in una lingua hanno vari significati, risulta utile mettere in relazione

Transcript:
French prosecutors are investigating former Chilean strongman Augusto Pinochet. The French justice minister may seek his extradition from Britain. Three French families whose relatives disappeared in Chile have filed a Complaint charging Pinochet with crimes against humanity. The national court in Spain has ruled crimes committed by the Pinochet regime fall under Spanish jurisdiction.

Speech recognizer output:
french prosecutors are investigating former chilean strongman of coastal fish today the french justice minister may seek his extradition from britain three french families whose relatives disappeared until i have filed a complaint charging tenants say with crimes against humanity the national court in spain has ruled crimes committed by the tennessee with james all under spanish jurisdiction

Figura 2.4: Esempi di errori di trascrizione in sistemi di riconoscimento vocale.

i termini con il loro contesto, per poter distinguere tra di essi. Inoltre, le parole in una lingua possono essere tradotte in vario modo in un'altra lingua.

In quest'ottica, le interrogazioni corte contengono in generale termini ambigui, che ne rendono difficile la traduzione sfruttando ontologie, dizionari e corpora di testi. Riuscire ad identificare un contesto specifico aiuta a migliorare il reperimento, in quanto possono essere esclusi dalla lista dei risultati tutti quei documenti che non trattano l'argomento di interesse, pur contenendo le parole chiave della ricerca (o una delle loro traduzioni). Per maggiori informazioni sull'argomento si rimanda a (35) e (59).

2.3.2.3 Collaborative question answering (CQA)

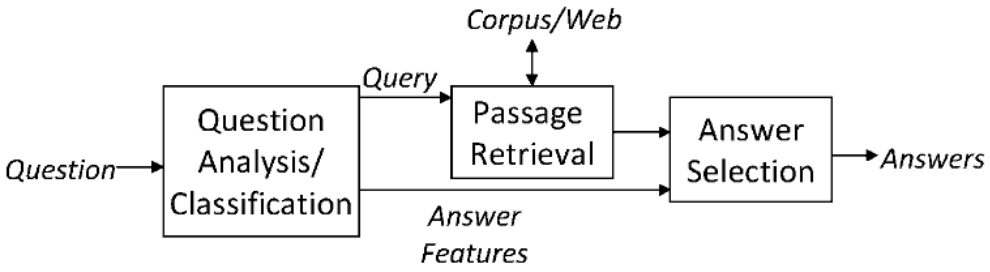


Figura 2.5: Architettura di un sistema di QA.

I sistemi di Question Answering (9) producono in output delle *risposte* anziché una lista ordinata di documenti. Quelli detti *fact-based* limitano il range delle possibili domande a quelle che hanno risposte semplici e corte. Domande che appartengono a questa categoria sono tipicamente quelle che iniziano con *who, where, when,* ecc. e prendono il nome di *factoid-type question*. Si riporta in fig. 2.5 l'architettura di un sistema per il QA. Facendo riferimento ai sistemi *fact-based*, le domande vengono *classificate* in base al tipo di risposta che ci si aspetta dal sistema. Molte di tali categorie corrispondono a nomi di entità (si riporta qualche esempio in tabella 2.2). L'utilizzo di categorie permette di identificare passaggi in potenziali risposte. Attraverso l'elaborazione del linguaggio naturale e l'inferenza semantica è poi possibile assegnare dei punteggi ai vari passaggi al fine di identificare la risposta più plausibile.

Example Question	*Question Category*
What do you call a group of geese?	Animal
Who was Monet?	Biography
How many types of lemurs are there?	Cardinal
What is the effect of acid rain?	Cause/Effect
Boxing Day is celebrated on what day?	Date
What is sake?	Definition
What is another name for nearsightedness?	Disease
What was the famous battle in 1836 between Texas and Mexico?	Event
What is the tallest building in Japan	Facility
What is the most popular sport in Japan?	Game
What is the capital of Sri Lanka?	Geo-Political Entity
Name a Gaelic language.	Language
What is the world's highest peak?	Location
How much money does the Sultan of Brunei have?	Money
Jackson Pollock is of what nationality?	Nationality
Who manufactures Magic Chef appliances?	Organization
What kind of sports team is the Buffalo Sabres?	Org. Description
What color is yak milk?	Other
How much of an apple is water?	Percent

Tabella 2.2: Classificazione delle domande in un sistema di QA di tipo *fact-based*.

I servizi di *Common Question Answering* (CQA), quali **Yahoo! Answers** o **Live QnA**, mettono a disposizione delle domande degli utenti risposte di altri utenti. Le domande sono principalmente di carattere generale, ma includono anche le *factoid-type question*. Tali interrogazioni sono più lunghe di quelle che comunemente vengono utilizzate per effettuare delle ricerche sul Web.

Problema di questo tipo di applicazioni è la latenza nelle risposte: finché un utente in grado di rispondere alla domanda non la legge e decide di rispondere, l'esigenza informativa non viene soddisfatta. Perciò, per migliorare le performance di questo tipo di sistemi, è buona regola che ogniqualvolta un utente sottopone un'interrogazione al sistema, quest'ultimo consulti automaticamente gli archivi delle risposte già date, anziché rimanere in attesa che sia un altro utente a formulare la risposta. Per fare ciò è possibile seguire diversi approcci:

- trattare le risposte come "mini-documenti" e quindi effettuare la ricerca sul corpus di "mini-documenti";
- trattare il problema come un *(factoid) question-answering problem*, e quindi in base alla categoria assegnata alla domanda calcolare i punteggi delle possibili risposte;
- trovare domande simili e reperire le risposte associate.

Studi hanno dimostrato che il terzo approccio è quello che dà i risultati migliori (10).

L'approccio proposto da Croft (9) per reperire le domande simili a quelle a cui si vuole dare risposta fa riferimento a un modello di reperimento basato su traduzione (si riporta una rappresentazione schematica del modello in fig. 2.6). Tale approccio, che, senza scendere nei dettagli, prevede di *riformulare* l'interrogazione, è un estensione del *query likelihood retrieval model* (si veda (6)).

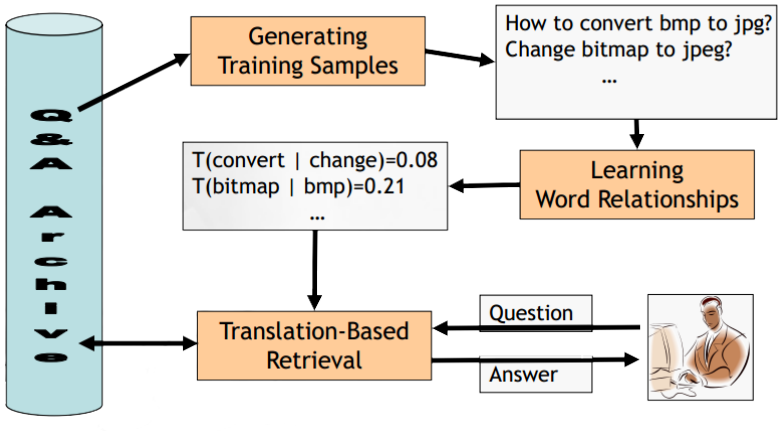


Figura 2.6: Modello per un'applicazione di QA basato su traduzione.

2.4 Approcci di ricerca

2.4.1 Trasformazione delle query

Sono stati fatti vari tentativi per incrementare l'efficacia dei motori di ricerca utilizzando query prolisse. Obiettivo comune dei vari approcci è quello di ridurre le query prolisse e incrementarne la struttura, in modo da poter utilizzare in modo più efficace i modelli di reperimento comunemente diffusi. Nel seguito si propone una panoramica sui vari metodi utilizzati in letteratura per trasformare le interrogazioni, lo scopo dei quali è quello di:

- passare da query lunghe a query corte, attraverso tecniche quali *Natural Language Processing* (NLP), riassunti automatici, parafrasi e traduzioni, segmentazione, ecc.;
- passare da query non strutturate a query strutturate, utilizzando tecniche per la pesatura dei termini.

Naturalmente tali tecniche non si escludono a vicenda. È interessante ad esempio notare come l'assegnazione di pesi nulli equivale ad una riduzione delle interrogazioni. Comunemente vengono combinate più strategie e spesso si utilizza il NLP in una fase di *pre-processing* dell'interrogazione.

2.4.1.1 Da query lunghe a query corte

Il ***Natural Language Processing*** (NLP) è il campo dell'informatica che si occupa delle interazioni tra computer e linguaggio naturale umano. Esso studia i problemi connessi alla generazione automatica o alla comprensione del linguaggio naturale, scritto o parlato. Il processo di elaborazione generalmente viene suddiviso in varie fasi:

1. *tokenization*: scomposizione di un'espressione linguistica in *token* (parole, spazi, punteggiatura, frasi);
2. analisi morfologica: consiste nell'individuazione delle forme di flessione e regole di composizione (*part-of-speech*);
3. analisi lessicale: individuazione dei vocaboli;
4. analisi sintattica: creazione di una struttura sintattica ad albero (*parse tree*);

5. *named entity recognition*: identificazione di entità con significato semantico;
6. analisi semantica: assegnazione di un significato alla struttura sintattica e, di conseguenza, all'espressione linguistica;
7. *co-reference*: individuazione di pronomi e parafrasi;
8. analisi pragmatica: distinzione del testo in dialogo, discorso, metafora;

Per un approfondimento si rimanda a (25). Tali concetti verranno ripresi in parte nel seguito.

Altra tecnica che viene utilizzata per trasformare un'interrogazione lunga in una corta è quella della ***summarization***. L'idea di effettuare dei riassunti in maniera automatica risale a più di 50 anni fa. I primi lavori a tal riguardo ((37) e (16)) si basavano sulle seguenti assunzioni:

- frasi in posizioni privilegiate riassumono l'argomento del testo (come ad esempio il primo paragrafo, o il paragrafo che segue i titoli quali "Introduzione", "Conclusioni", ecc.);
- la presenza di parole chiave, che in inglese vengono dette *lexical cues*, quali ad esempio *in conclusione, in questo documento*, ecc., indicano l'argomento principale del testo.

È però evidente come tali approcci dipendano fortemente dal formato e dallo stile del testo e che pertanto non risultano efficaci in contesti generali.

A partire dagli anni '70 vennero sviluppate tecniche di intelligenza artificiale basata sulla semantica, il cui intento era quello di simulare il processo che porta le persone alla generazione di riassunti. Un vero riassunto richiede la comprensione ed interpretazione del testo, che viene riproposto in una nuova forma, secondo differenti livelli di astrazione ((33)). Altri approcci alla generazione automatica di riassunti si trovano in (39) e (40).

Un altro metodo per ridurre un'interrogazione è quello di usare un ***dependency parsing tree*** tra i termini, come mostrato da Park e Croft (46). Un albero di questo tipo è una collezione di nodi e di archi che dividono le parole in frasi sulla base delle parti del discorso e delle relazioni che intercorrono tra i termini. Un *parsing tree* delle dipendenze, in particolare, mette in relazione le parole sulla base di quali altre parole modificano. Ad esempio, un arco dalla parola A alla parola B significa che B modifica il

significato di A. L'etichetta tra due parole connesse indica il tipo di connessione, come mostrato in fig.2.7.

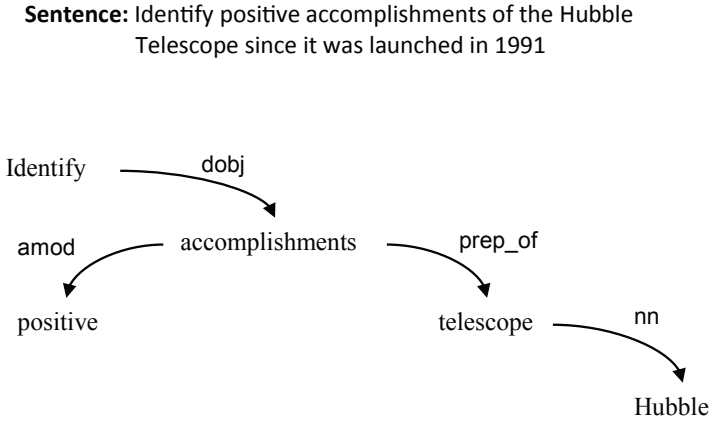


Figura 2.7: Esempio di parsing tree delle dipendenze. Un arco rappresenta una dipendenza tra due parole. L'etichetta sull'arco indica il tipo di dipendenza. Ad esempio, "Accomplishments" e "telescope" sono messi in relazione dalla preposizione "of".

Non tutte le relazioni messe in luce da un *dependency parsing tree* sono utili, e per limitare il numero di caratteristiche sintattiche di cui tenere conto, Park e Croft hanno limitato a due archi il numero di archi (46).

On-line è possibile trovare diversi parser che, tra le funzioni aggiuntive, costruiscono *dependency parsing tree*. Per la costruzione di un albero delle dipendenze è necessario inanzitutto individuare le parti del discorso. A tal fine si fa solitamente riferimento a database semantici-lessicali già esistenti, tra cui uno dei più famosi per la lingua inglese è **WordNet** (64). Inoltre, in genere si utilizzano dei dizionari dei sinonimi e una lista ristretta di termini per descrivere le relazioni di interesse.

In fig. 2.8 si riporta lo schema di funzionamento di **RelEx** (17), un tool open-source per l'estrazione di relazioni sulla base di *dependency parsing tree*, che sintetizza in che modo vengono costruite le dipendenze tra i termini. Sebbene l'immagine faccia riferimento a questo tool in particolare, le componenti sono quelle che vengono utilizzate in generale.

Altro esempio di tool è quello fornito dalla *Stanford University*, la quale ha messo a disposizione lo **Stanford Dependencies (SD)**, che permette di estrarre le relazioni

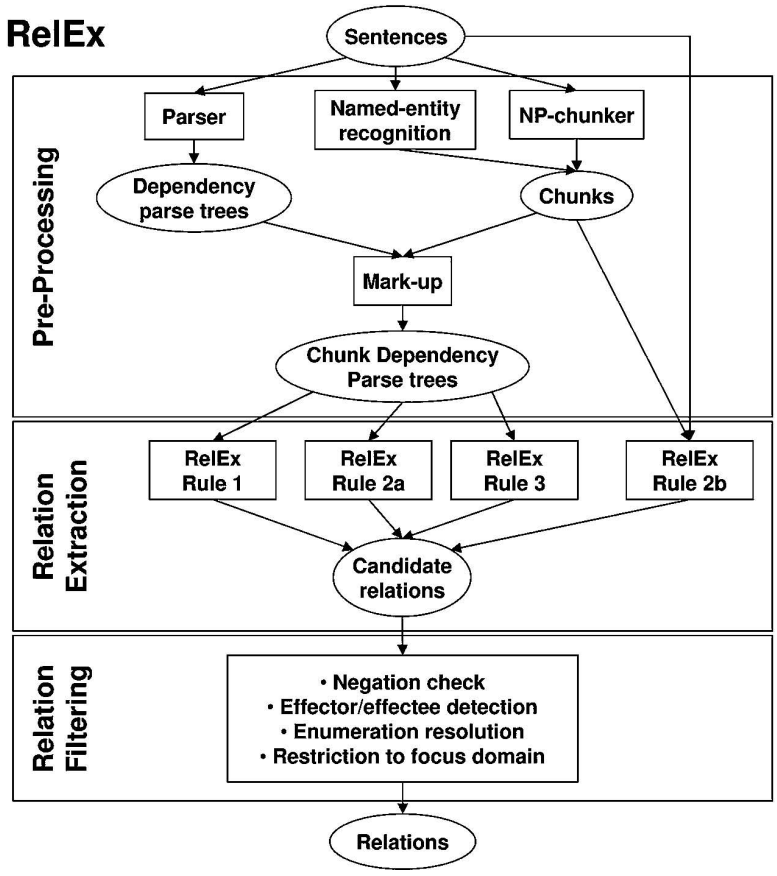


Figura 2.8: Lo schema rappresenta il work-flow di RelEx, suddividendolo in preprocessing, estrazione delle relazioni e filtraggio di quest'ultime, che porta dal testo originale fornito in input a relazioni dirette ed etichettate. Il preprocessing è basato su tool disponibili al pubblico, come lo Stanford Parser.

testuali per testi scritti in inglese e cinese. Le dipendenze standard per la frase - *Bills on ports and immigration were submitted by Senator Brownback, Republican of Kansas* vengono riportate nel seguito come esempio, assieme a due rappresentazioni grafiche: l'albero delle dipendenze esteso (v. fig. 2.11) e ridotto (v. fig. 2.10) e la rappresentazione base in cui ciascuna parola della frase (ad eccezione della testa) dipende da un'altra parola (v. fig. 2.9). Per maggiori informazioni si rimanda al manuale (13).

La **segmentazione** di un'interrogazione lunga in più query è un tipo di trasformazione che aggiunge struttura all'interrogazione. Una volta che la query viene suddivisa

nsubjpass(submitted, Bills)
auxpass(submitted, were)
agent(submitted, Brownback)
nn(Brownback, Senator)
appos(Brownback, Republican)
prep_of(Republican, Kansas)
prep_on(Bills, ports)
conj_and(ports, immigration)
prep_on(Bills, immigration)

Figura 2.9: Dipendenze per la frase *Bills on ports and immigration were submitted by Senator Brownback, Republican of Kansas* prodotte con SD.

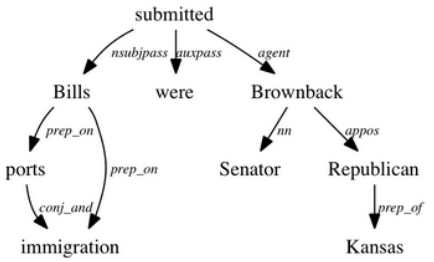


Figura 2.10: Albero delle dipendenze ridotto per la frase *Bills on ports and immigration were submitted by Senator Brownback, Republican of Kansas* prodotto con SD.

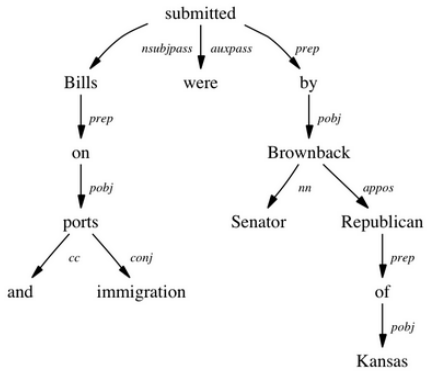


Figura 2.11: Albero delle dipendenze esteso per la frase *Bills on ports and immigration were submitted by Senator Brownback, Republican of Kansas* prodotto con SD.

in più parti, si può decidere di assegnare pesi diversi ai vari segmenti (si tratterà brevemente il problema dell'assegnazione dei pesi nel par. 2.4.1.2).

Xue, Huston e Croft hanno proposto in "*Improving Verbose Queries using Subset Distribution*" (65) un metodo per la selezione dei sottoinsiemi di termini che andranno a formare un segmento. È stato dimostrato che selezionare un sottoinsieme della query originale (una "sub-query"), nel caso di query lunghe, porta ad un aumento di efficacia. La selezione della sub-query può essere considerata come un **sequential labeling problem**, ossia un problema nel quale ogni parola costituente la query prolissa viene etichettata con un etichetta del tipo "selezionata" o "non selezionata". A differenza di un problema di classificazione che assegna ad ogni parola una classe (scelta tra le due possibili), tale problema tiene conto delle relazioni presenti tra le parole.

È possibile osservare come, nonostante l'approccio al problema delle interrogazioni prolisse presentato in questo paragrafo sia concettualmente diverso da quanto presentato nel par. precedente, il risultato che si ottiene è del tutto simile a quello che si avrebbe riducendo le query attraverso l'utilizzo di *dependencies parsing tree*.

Xue, Huston e Croft propongono in (65) un *conditional random field model* per la generazione di sub-query. Il modello si basa sull'assunzione che è ragionevole considerare le relazioni tra parole per scegliere quali selezionare per la sub-query. Se alcune parole sono in forte relazione tra loro, si tenderà a dar loro la stessa etichetta ("selezionata"/"non selezionata"). Ad esempio se si considera il sintagma (in inglese: *noun phrase*) *Spanish Civil War*, si tenderà a selezionare nella sottoquery tutte e tre le parole che lo compongono, o nessuna delle tre.

L'obiettivo della fase di "etichettatura" non è quello di selezionare la migliore subquery, bensì più in generale di modellare una distribuzione sullo spazio di tutte le possibili sub-query. Questo significa che la probabilità di osservare ciascuna sub-query in questa distribuzione può essere usata come peso nel caso in cui sia necessario combinare più sub-query tra loro, ottenendo una query espansa.

Per informazioni più dettagliate riguardo ai *Conditional Random Field* si rimanda a (28).

2.4.1.2 Da query non strutturate a query strutturate

Una volta che una query prolissa viene ridotta, ad esempio attraverso l'utilizzo dei metodi proposti nel paragrafo precedente, resta da stabilire quale peso assegnare a ciascun termine o sottoinsieme di termini. La **pesatura dei termini** è un problema classico dell'information retrieval. Nonostante i lunghi anni di studio, la maggior parte

dei modelli più utilizzati in IR, inclusi i *language model* (48) e il BM25 (52), pesano i concetti delle query in maniera non supervisionata. L'utilizzo di tali approcci, basati ad esempio su una singola statistica globale, come IDF, non tengono conto dei molti fattori che legano la frequenza nel documento all'importanza del concetto. Inoltre, non sono stati condotti molti studi che testano il funzionamento di schemi di pesatura come IDF su concetti composti da più di un termine.

Bendersky, Croft e Metzler (5) hanno proposto un modello in cui il peso di ciascun concetto è determinato utilizzando la combinazione parametrizzata di *feature* con diversa importanza. Il modello proposto apprende i pesi non solo esplicitamente sulla base dei concetti contenuti nella query, ma anche attraverso i concetti latenti associati alla query attraverso un meccanismo di *pseudo-relevance feedback*. L'utilizzo della retroazione permette di ridurre il numero di termini che vengono utilizzati per l'espansione delle query, migliorando significativamente le performance.

2.4.2 Modelli di reperimento

Si presentano brevemente nel seguito alcuni dei modelli di reperimento utilizzati in letteratura. Un modello di reperimento è un insieme di costrutti che sono formalizzati allo scopo di rendere possibile la rappresentazione del contenuto di documenti e interrogazioni, nonché di definire l'algoritmo di IR dei documenti in risposta ad un'interrogazione (41). Un modello è, quindi, uno strumento concettuale per astrarre il funzionamento di un sistema di IR (vedi fig. 2.12), il contenuto dei documenti e l'esigenza informativa espressa nelle interrogazioni.

Alcuni dei modelli che verranno qui proposti sfruttano le relazioni tra i termini che costituiscono la query, come ad esempio il *Limited Dependencies Language Model* e il *Markov Random Field Model*, e risultano pertanto particolarmente efficaci nel caso di interrogazioni strutturate, come quelle ottenibili da query scritte in linguaggio naturale. Gli altri modelli proposti, come il *Relevance Model*, i *Translational Model* e gli *Unified Model*, pur non sfruttando direttamente le relazioni tra i termini, possono risultare efficaci in particolare se utilizzati in combinazione ai metodi di riduzione delle interrogazioni presentati nella sez. 2.4.1.

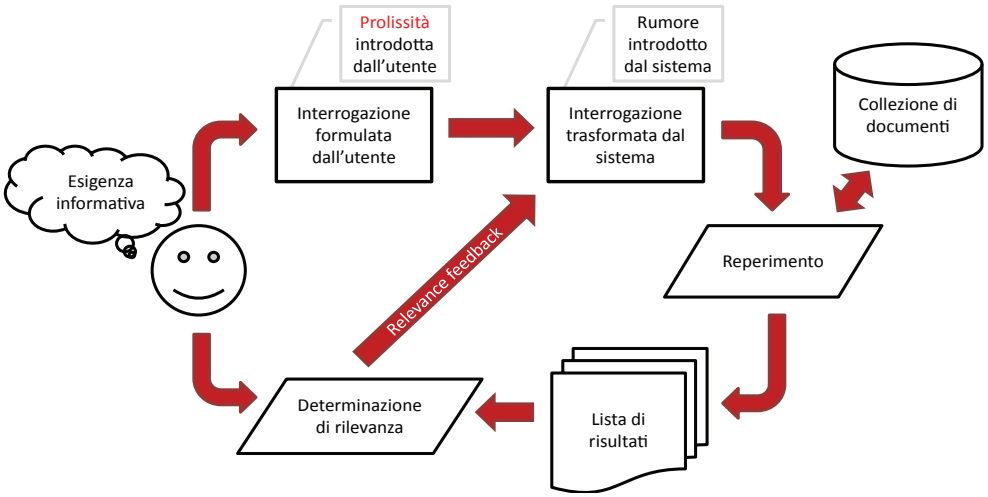


Figura 2.12: Schema di un sistema di IR.

2.4.2.1 Relevance Model

La *rilevanza* è un concetto centrale in IR e sono state fatte varie ricerche nel settore per fornire un modello formale di rilevanza. In (31), Lavrenko e Croft propongono un modello formale di rilevanza per integrare il modello probabilistico classico con tecniche più recenti legate ai *language model*.

Nel seguito si farà riferimento ai seguenti parametri:

- V: un vocabolario;
- C: una collezione composta da un gran numero di documenti;
- R: il sottoinsieme di documenti rilevanti ($R \subset C$); si indicherà invece con $overline R$ l'insieme complementare;
- $P(w \mid R)$: la probabilità che la query Q contenga la parola w conoscendo i documenti rilevanti;

Uno dei modelli di reperimento più popolari, introdotto da Robertson e Sparck Jones (51), assegna i punteggi ai documenti in base alla probabilità che essi appartengano alla classe di documenti rilevanti per una query. Il modello probabilistico rappresenta l'incertezza e il costo della decisione presa dal sistema di IR di reperire un documento, attraverso strumenti del calcolo delle probabilità e della statistica e, in particolare, della teoria statistica delle decisioni (vedi fig. 2.13). Il modello probabilistico classico si basa

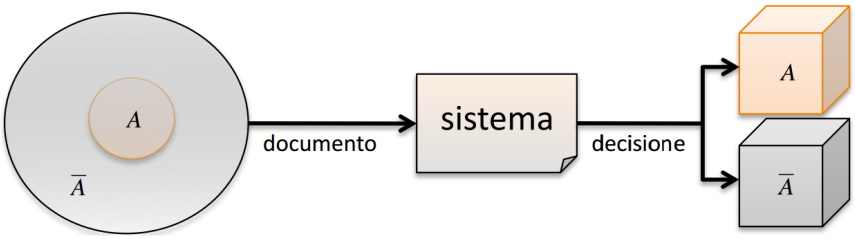


Figura 2.13: Decisione secondo il modello probabilistico.

sul **Probability Ranking Principle** (PRP) (50), che può essere espresso nel modo seguente (41):

> Se un sistema di IR risponde a ciascuna interrogazione con una lista di documenti ordinati in modo non crescente per probabilità di rilevanza all'esigenza informativa dell'utente che ha espresso l'interrogazione, e posto che le probabilità siano state stimate nel modo migliore possibile sulla base dei dati a disposizione, allora l'efficacia complessiva del sistema è la maggiore possibile sulla base di quei dati.

Il PRP (che è solo un principio, e non un teorema) si basa sulle seguenti assunzioni semplificative:

- la rilevanza è una variabile binaria;
- la rilevanza di un documento non dipende da quella degli altri documenti della lista di risultati.

I punteggi che il PRP prevede di assegnare ad ogni documento $D \in C$ vengono calcolati come valori non crescenti secondo il seguente rapporto tra probabilità (31):

$$\frac{P(D \mid R)}{P(D \mid \overline{R})}$$

Se si assume che un documento D sia una sequenza di termini indipendenti: $d_1, d_2, ..., d_n$, il *probability ranking principle* può essere espresso come:

$$\frac{P(d_1 \cdots d_n \mid R)}{P(d_1 \cdots d_n \mid \overline{R})} = \prod_{i=1}^{n} \frac{P(d_i \mid R)}{P(d_i \mid \overline{R})}$$

Il modelli di rilevanza proposti da Lavrenko e Croft combinano questo modello classico di reperimento con quelli più recenti introdotti sotto il nome di *language model*,

che si basano sulla probabilità di osservare i termini che costituiscono la query all'interno dei documenti rilevanti. Tali modelli si basano infatti sull'assunzione che chi scrive l'interrogazione sia a conoscenza di quali termini compaiono nel documento di interesse. Si introduce così il vantaggio di spostare l'attenzione sullo sviluppo di euristiche quali *tf*idf* per rappresentare l'importanza dei termini, piuttosto che concentrarsi su tecniche per la stima di modelli di documenti.

Gli approcci di tipo *language-modeling* sono stati poi estesi per la *cross-language retrieval* da Hiemstra e de Jong in (22) e da Xu et al. Anche il modello proposto da Berger e Lafferty, che si applica per la "traduzione" di un documento in una query in un ambiente monolingua, può essere facilmente adattato per un ambiente bilinguistico.

Il *ranking* dei documenti può essere basato sulla **cross-entropy**, ossia la misura di divergenza tra due *language model*. Sia $P(w \mid R)$ il language model della classe di rilevanza e $P(w \mid D)$ per ogni documento D il corrispondente *language model* del documento. La cross-entropy può essere calcolata come:

$$H(R \parallel D) = -\sum_{w \in V} P(w \mid R) log P(w \mid D)$$

Intuitivamente, documenti con una piccola cross-entropy sono "simili" a quelli rilevanti, quindi conviene ordinare i documenti secondo cross-entropy crescente (30).

Il modello proposto da Lavrenko e Croft valuta varie tecniche per stimare l'insieme delle probabilità $P(w|R)$, che possono essere suddivise in:

- stima non supervisionata;
- stima supervisionata;
- stima basata su tecniche di *cross-language retrieval*.

Si rimanda a (31) per ulteriori dettagli.

2.4.2.2 Limited Dependencies Language Model

I language model nella pratica spesso risultano efficaci. Una delle assunzioni del modello è di considerare i termini indipendenti tra loro. Le proprietà del linguaggio naturale del target di documenti possono essere però usate per trasformare e arricchire le dipendenze

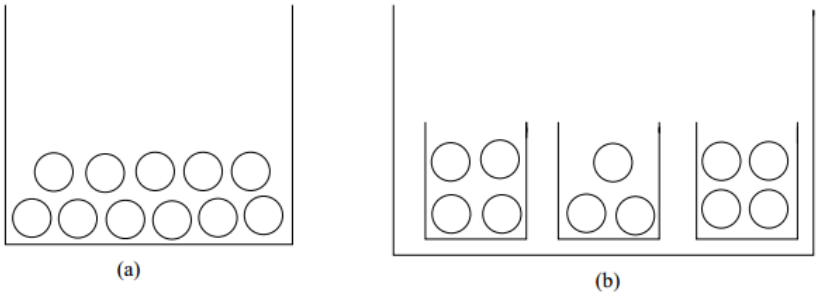


Figura 2.14: In (a) un documento come viene visto nell'Unigram Language Model e in (b) come viene visto nel Dependencies Language Model.

tra termini, al fine di ottenere delle utili statistiche. In fig. 2.14 vengono messe a confronto le visioni in cui si considerano unigrammi indipendenti e multigrammi composti da termini in relazione tra loro.

In (23) vengono sfruttate le relazioni tra termini in tre passi:

1. Le co-occorrenze dei termini delle query e dei documenti vengono rappresentate come catene di Markov[1]. Si può dimostrare che una catena di questo tipo è erogodica[2], e pertanto il suo comportamento asintotico è unico, stazionario e indipendente dallo stato iniziale.

2. Si considera la distribuzione stazionaria per modellare query e documenti, piuttosto che la distribuzione iniziale.

3. Si calcolano i vari punteggi utilizzando un paradigma basato su *language model*.

2.4.2.3 Markov Random Field Model

Riassumendo l'approccio proposto da Metzler e Croft in (42), l'utilizzo di un **Markov Random Field Model** prevede di:

- costruire un grafo che rappresenta le dipendenze tra i termini dell'interrogazione (in fig. 2.15 vengono riportati degli esempi);

[1] Si dice catena di Markov (finita) un sistema dotato di un numero finito n di stati $1, 2, ..., n$ che soddisi la seguente ipotesi: la probabilità che il sistema passi dallo stato i allo stato j al tempo $k-1$ è $p_{ij}(k)$.

[2] Una catena di Markov è ergodica se è persistente e aperiodica.

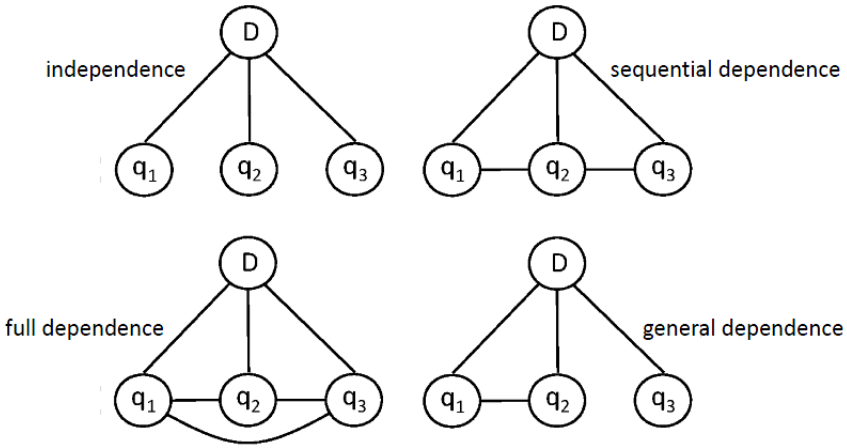


Figura 2.15: Assunzioni del MRF Model.

- definire un insieme di potenziali funzioni sulle clique[1] del grafo;
- assegnare i punteggi ai documenti;

I punteggi vengono assegnati in ordine non crescente di P_Λ, calcolata come la probabilità a posteriori:

$$P_\Lambda(D \mid Q) = \sum_{c \in C(G)} \lambda_c f(c)$$

dove $f(c)$ è una *feature function* sulla clique e λ_c è il peso dato alla particolare *feature*. L'utilizzo di tale modello è stato esteso dagli autori in (43), dove vengono generalizzati lo *pseudo-relevance feedback* e i modelli di rilevanza.

In (32) si presenta come l'utilizzo di interrogazioni prolisse permette di migliorare il modello di Metzler e Croft.

2.4.2.4 Translation Models

In *"Information retrieval as statistcal translation"* (6), Berger e Lafferty propongo un nuovo approccio al reperimento dell'informazione basato su metodi statistici di traduzioni automatica. L'ingrediente centrale di tale approccio è un modello statistico di

[1] Sia $G = (V, E)$ è un grafo non orientato; una clique C di G è un sottografo completo massimale, cioè tale per cui tutti i nodi di C sono a due a due adiacenti e che nessun altro sottografo completo di G contiene C.

come un utente dovrebbe riassumere o "tradurre" un dato documento in un'interrogazione. Per valutare la rilevanza di un documento rispetto un'interrogazione dell'utente, si stima la probabilità che tale interrogazione venga generata come traduzione di un documento. Tale modello può essere visto come una generalizzazione e giustificazione dei *language model* proposti da Ponte e Croft (48).

Questo tipo di approccio è spesso sfruttato da sistemi per il *Question Answering*. Si rimanda a ((66), (49) e (15) per ulteriori approfondimenti.

2.4.2.5 Unified Models

Altri approcci cercano di migliorare le performance del sistema di reperimento ridefinendo le interrogazioni sulla base di varie strategie quali: *stemming*, correzione ortografica, suddivisione in sottosezioni, selezione di termini, sostituzione o aggiunta di termini. Si può trovare un esempio in (19) oppure in (62).

3

Metodologia

Scopo di questo capitolo è mettere in luce i metodi che si utilizzeranno in fase di sperimentazione per valutare in che misura l'utilizzo di interrogazioni prolisse influisca sulle performance di un sistema di reperimento.

Dopo aver chiarito quali sono i fattori che possono influenzare la ricerca, soffermandosi su metodo di indicizzazione e modello di reperimento, si presenterà una strategia per elaborare le interrogazioni prolisse, con lo scopo di eliminare parte del rumore in esse contenuto. In particolare, si proporrà:

- *un algoritmo per l'estrazione dei sinonimi dei termini contenuti nella query, che verranno inseriti nell'interrogazione come clausole OR;*
- *varie versioni di un algoritmo per il topic gisting, ossia per l'estrazione di termini rilevanti al fine di descrivere il contenuto dell'interrogazione (anche non presenti nella query originale, ma ad essa semanticamente collegati), che verranno inseriti nella query con una pesatura maggiore rispetto ai termini originali.*

A tal fine si utilizzerà ConceptNet, un database di relazioni semantiche, a cui è possibile accedere attraverso un indice creato in Solr.

Nella seconda metà del capitolo, si proporrà un predittore pre-reperimento per stabilire se un'interrogazione è prolissa o meno, che verrà creato attraverso Weka.

3.1 Oggetto d'indagine ed ipotesi

Quello che si vuole indagare è in che misura il tipo di interrogazione, in combinazione al modello di reperimento, influisca sulle performance di un motore di ricerca. Poiché

in generale l'utilizzo di interrogazioni lunghe o prolisse[1] influisce negativamente sulle performance, si vuole proporre un metodo per eliminare parte del rumore presente nelle query, in modo da migliorare le prestazioni del sistema. Si vuole proporre infatti un metodo per elaborare le query in modo da incrementare l'efficacia del reprimento. A tal proposito si pensa che un buon metodo possa essere quello di sfruttare le **relazioni semantiche** tra i termini che costituiscono le interrogazioni. In particolare, si proporranno due tipi di algoritmi, il primo per estrarre i sinonimi dei termini contenuti nella query, che verranno inseriti nell'interrogazione sotto forma di *clausole OR*; il secondo per il *topic gisting*, ossia per l'estrazione del succo del discorso, costruendo un insieme di termini semanticamente collegati a quelli presenti nella query originale. Tali termini verranno inseriti nell'interrogazione assegnando ad essi un peso maggiore rispetto a quelli originali. I dettagli verranno esposti nel par. 3.3.

Diversi fattori possono influenzare gli esiti della ricerca, a parità di metodo di indicizzazione e modello di reperimento:

- Corpus di documenti: possono variare infatti le dimensioni del corpus e il numero di termini[2];
- Tipo di esigenza informativa; con riferimento alla classificazione proposta da Broder in (7) si è deciso di considerare interrogazioni che rappresentano esigenze di tipo *informational*. Oltre a questo tipo di classificazione, le esigenze informative possono venire suddivise anche in base alla complessità: alcune risultano difficili per i sistemi di reperimento, ossia si ottengono scarsi risultati dal reperimento.
- Tipo di query: corte o lunghe, prolisse o non prolisse e le 4 combinazioni possibili di queste categorie;
- Metodo utilizzato per elaborare le query;

Si terrà conto di questi fattori in fase di sperimentazione (v. cap. 4).

Poiché nelle applicazioni reali i sistemi di reperimento non dispongono di informazioni preliminari sul tipo di interrogazioni che ricevono in input, si ritiene utile proporre

[1] Non è detto che un'interrogazione prolissa sia lunga, ma quasi certamente un'interrogazione molto lunga sarà prolissa.

[2] La **legge di Heap** mette in relazione la dimensione del vocabolario e la lunghezza della collezione in numero di parole. Sia V la prima e N la seconda di queste grandezze. La crescita del vocabolario sarà allora: $V = KN^{\beta}$, dove $K \approx 10 - 100$ e $0 < \beta < 1$.

un predittore pre-reperimento per stabilire se l'interrogazione possa essere riassunta. Un'informazione aggiuntiva di questo tipo potrebbe essere sfruttata da un sistema di reperimento in vari modi, in particolare per scegliere il metodo più adatto per elaborare l'input. Nel corso di questa ricerca ci si limiterà a mettere in relazione l'informazione sul tipo di interrogazione con i risultati ottenuti in fase di reperimento.

3.2 Indicizzazione e modello di reperimento

Si è scelto di utilizzare le librerie di **Lucene** per effettuare l'indicizzazione dei documenti e implementare vari modelli di reperimento.

Per l'indicizzazione si sono sfruttate in particolare le seguenti opizioni:

- `Field.Store.YES/NO`, per salvare o meno il campo all'interno dell'indice;
- `Field.Index.ANALYZED/NO`, per stabilire se considerare singolarmente i termini del campo o se indicizzarli come un termine unico;
- `Field.TermVector.YES/NO`, per decidere se costruire il vettore dei termini.

I campi dell'indice sono riportati in tabella 3.1 con le rispettive impostazioni.

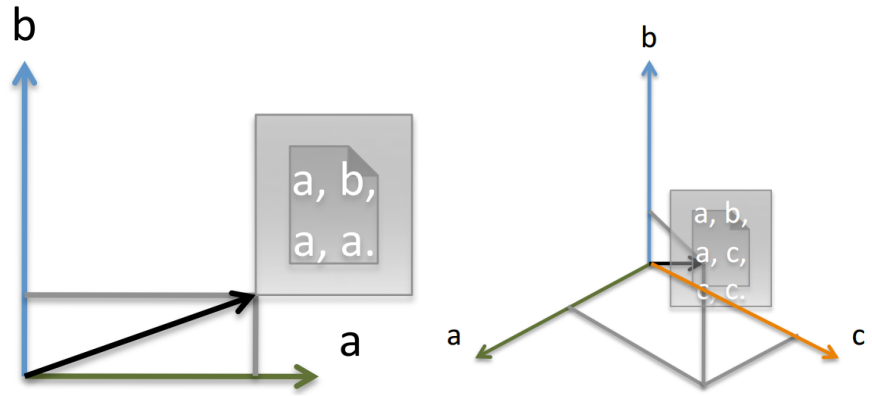
Campi	**Impostazioni**
docno	`Field.Store.YES`, `Field.Index.NO`, `Field.TermVector.NO`
text	`Field.Store.YES`, `Field.Index.YES`, `Field.TermVector.YES`

Tabella 3.1: Campi dell'indice e corrispondendi impostazioni.

Inoltre, si è utilizzata la lista di *stopword* riportata nell'allegato B (la stessa lista è stata utilizzata anche in fase di reperimento).

Il modello di reperimento messo a disposizione da Lucene (36) è una combinazione di ***Vector Space Model*** (VSM) e ***Boolean Model***. Secondo il modello vettoriale, proposto tra gli anni sessanta e gli anni settanta da Salton (54), i descrittori sono vettori di uno spazio lineare di dimensione finita e le loro combinazioni lineari sono documenti, interrogazioni o qualisasi altro oggetto che contiene informazioni come, ad esempio, un video un suono, un'immagine, il cui contenuto emantico è descritto, appunto, dai

descrittori (vedi fig. 3.1). Lucene utilizza il modello booleano per discriminare i documenti che necessitano di essere pesati sulla base dell'utilizzo della logica booleana nella specificazione della query.



(a) Spazio a due dimensioni (b) Spazio a tre dimensioni

Figura 3.1: Vettori di documenti e interrogazioni nello spazio vettoriale a due dimensioni (a) e in quello a tre dimensioni (b).

Si è deciso di implementare più di un algoritmo per il reperimento dei documenti. L'unica caratteristica che differenzia i vari modelli è il metodo utilizzato per elaborare le query in input. Nello specifico:

- vengono eliminate le *stopword* e viene fatto lo *stemming* (lo *stemming* viene effettuato utilizzando l'*English Analyzer* di Lucene);
- le query vengono modificate in vario modo utilizzando il tool **ConceptNet** (di cui si parlerà nella sezione seguente);

3.3 Elaborazione delle query attraverso ConceptNet

ConceptNet è un database di relazioni semantiche, strutturate secondo un ipergrafo, che racchiude diversi dizionari e tesauri (57). I nodi rappresentano *concetti*, sotto forma di parole o brevi frasi in linguaggio naturale, mentre gli archi rappresentano *relazioni*. Un'API REST permette di reperire i dati per un particolare nodo o lato, effettuare interrogazioni per lati con particolari proprietà, misurare la distanza semantica tra nodi. Si riporta in fig. 3.2 un esempio di rete semantica contenuta in ConceptNet.

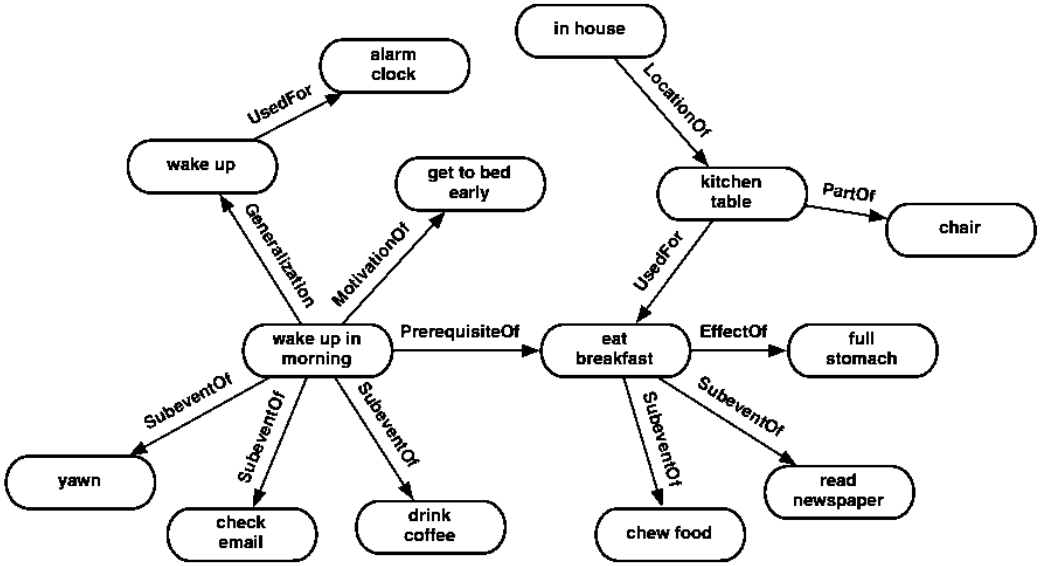


Figura 3.2: Esempio di rete semantica contenuta in ConceptNet.

È anche possibile scaricare ed utilizzare una copia personale dei dati, a cui è possibile accedere attraverso Solr. Per maggiori informazioni su ConceptNet si rimanda al sito `http://conceptnet5.media.mit.edu/` (56).

3.3.1 File `.json` importati

A causa della dimensione dei file, è sconsigliabile indicizzare interamente ConceptNet 5 (ossia l'ultima versione del tool, rilasciata nel maggio 2012) in Solr su un'unica macchina. Si è pertanto fatta una selezione delle varie fonti di informazione:

- WordNet 3 (64);
- ConceptNet 4 (8);
- DBpedia, è un progetto che consiste nell'estrazione di informazione struttura da Wikipedia. Per maggiori informazioni si rimanda a (12);
- Wiktionary, un dizionario simile a WordNet, che contiene anche sinonimi e contrari (63).

Osservando tale lista, è possibile notare come si sia deciso di non importare i file che contengono specificatamente informazioni riguardanti la lingua cinese e le traduzioni

(anche se Wiktionary contiene delle corrispondenze tra termini in varie lingue), in quanto i documenti di interesse per la sperimentazione saranno unicamente in inglese.

3.3.2 Analisi preliminari su ConcepetNet

Si riportano in tabella 3.2 alcuni dati riguardanti l'indice creato a partire dai pacchetti `.json` importati in Solr.

Campi dell'indice	context, dataset, end, endLemmas, features, id, license, nodes, rel, sources, start, startLemmas, surfaceText, text, timestamp, uri, weight
numero di nodi	3.973.647
numero di relazioni (lati)	10.439.973
tipi di relazioni	51

Tabella 3.2: Informazioni sull'indice creato in Solr a partire da alcuni pacchetti di ConceptNet.

I concetti in ConceptNet sono prevalentemente rappresentati da pochi termini. Nel grafico di figura 3.3 si riporta l'andamento del numero di termini rispetto ai nodi.

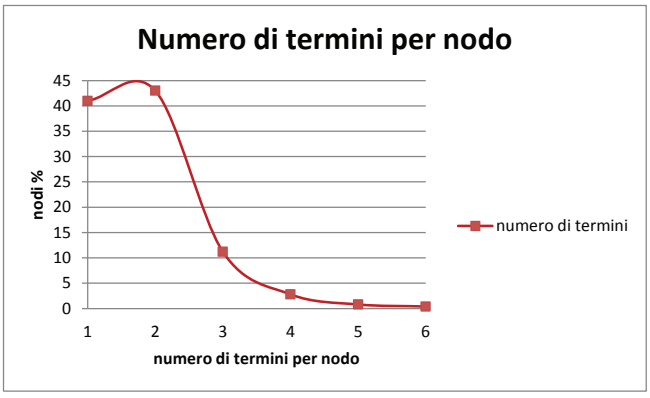


Figura 3.3: Numero di termini per nodo. Dal grafico è possibile osservare come i nodi sono prevalentemente semplici.

Utilizzando come query in Solr i termini del vocabolario inglese di Linux (contenuto nel file `/usr/share/dic/words`), si ottiene una media di 1124 risultati per parola. Si riporta in fig. 3.4 il numero di relazioni in rapporto ai termini del vocabolario.

Figura 3.4: Numero di relazioni per parola del vocabolario inglese di Linux (contenuto nel file `/usr/share/dic/words`).

3.3.3 Algoritmo per l'estrazione dei sinonimi

Si è deciso di sfruttare la relazione `/r/Synonym` per estrarre i sinonimi dei termini presenti nella query, in modo da espandere le interrogazioni.

Si è prestato attenzione al fatto che il Wiktionary contiene anche termini in lingue diverse dall'inglese, pertanto si è selezionata `/s/web/en.wiktionary.org` come fonte per i sinonimi, ossia la sezione del Wiktionary riguardante la lingua inglese.

Poiché la sintassi per le query di Lucene permette di sfruttare la logica booleana per la scrittura delle interrogazioni, si ritiene conveniente inserire i termini con sinonimi in clausole di tipo `OR`. Ad esempio, l'interrogazione corta:

```
Falkland petroleum exploration
```

diventa:

```
Falkland petroleum/oil/gasoline/petrol/petrolatum exploration
```

che tradotta nel linguaggio per le query di Lucene è:

```
text:falkland (text:petroleum text:oil text:gasoline text:petrol text:petrolatum
    ) text:exploration
```

Per maggiori dettagli sull'algoritmo si rimanda alla sezione B.4.1 in appendice.

3.3.4 Algoritmo per il *topic gisting*

L'idea che sta alla base dell'algoritmo viene riportata in fig. 3.5. Quello che si vuole fare è estrarre i concetti collegati ad ogni termine contenuto nella query. Una volta

ottenuti gli insiemi di concetti collegati, si effettua l'intersezione per ottenere i concetti in comune. Il *topic gisting*, così come viene qui proposto, è una tecnica che viene utilizzata principalmente per fornire riassunti di testi (24) o conversazioni (14). A differenza di questi lavori, quello che si vuole ottenere in questo caso, sfruttando le relazioni semantiche messe a disposizione da ConceptNet, è quello di estrarre dei termini che aggiunti all'interrogazione originale determinino un miglioramento di performance in fase di reperimento. Cambia quindi il campo di applicazione a cui si fa normalmente riferimento quando si parla di *topic gisting*.

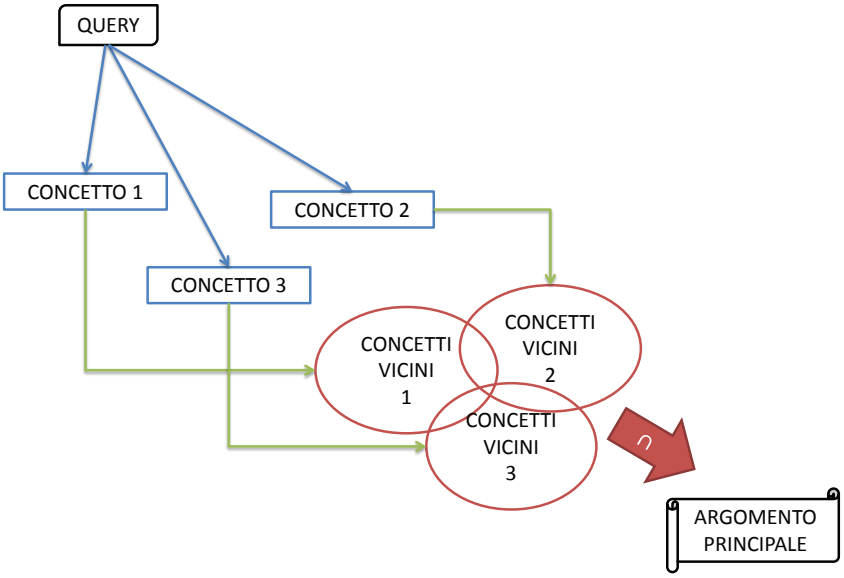


Figura 3.5: Schema riassuntivo del funzionamento dell'algoritmo per il *topic gisting*.

Poiché quello che si vuole ottenere è l'argomento principale di testi lunghi, si è preferito non effettuare l'intersezione di tutti gli insiemi di concetti collegati in un unico passaggio, perché così facendo si rischierebbe sempre di ottenere un'intersezione vuota. Si è preferito applicare un semplice algoritmo ricorsivo, che viene schematicamente riportato nel seguito:

Passo base: Se il testo è composto da un unico termine viene restituito l'insieme di concetti collegati a tale termine (nel caso in cui l'insieme sia vuoto si restituisce il termine stesso); se invece il testo è composto da due termini, per ciascuno di essi viene restituito l'insieme dei concetti collegati e si effettua l'intersezione; nel

caso in cui l'intersezione dovesse risultare vuota, viene restituito il testo dato in input.

Passo ricorsivo: Se il testo è composto da più termini, viene suddiviso in due parti e su ognuna si applica ricorsivamente l'algoritmo. Viene quindi effettuata l'intersezione dei risultati dei due sottoproblemi.

Nel passo ricorsivo si è scelto di suddividere il problema principale in due sottoproblemi sulla base degli studi di Bendersky e Croft (4), che hanno valutato la lunghezza attesa di una query per un query log a larga scala pari a 2,4 parole (vedi grafico in fig. 2.1). Intuitivamente, aumentando il numero di sottoproblemi aumenta la probabilità di considerare in ciascun sottoproblema un singolo termine, il che equivarebbe ad effettuare l'intersezione tra tutti gli insiemi di concetti collegati in un unico passaggio, riducendo l'efficacia dell'algoritmo (aumenta infatti la probabilità di ottenere un'intersezione vuota).

Si noti che così facendo, anche nel caso in cui un concetto non fa parte di ConceptNet, l'algoritmo funziona correttamente. Risulta inoltre evidente che modificando l'ordine delle parole nel testo varia anche il risultato finale delle intersezioni. A seconda dell'ordinamento scelto, si ottengono al termine dell'esecuzione dell'algoritmo insiemi composti da un numero di termini diverso. Lo scopo del presente algoritmo non è però quello di fornire in output l'insieme più piccolo possibile, bensì quello di estrarre dei termini che se aggiunti all'interrogazione originale ne aumentino le performance in fase di reperimento, determinando il reperimento di un maggior numero di documenti rilevanti e aumentando il numero di tali documenti in cima alla lista dei risultati.

Questo approccio non tiene conto però che se sono presenti "troppi" concetti, si può incappare nel problema di *topic drift*, senza contare che all'aumentare del numero di concetti aumenta anche il carico computazionale. Per questa ragione, si è preferito selezionare i concetti da dare in input all'algoritmo sulla base delle parti del discorso, che verranno individuate utilizzando lo **Stanford Parser**. Si è scelto di utilizzare due tipologie di selezione:

- la prima, che verrà indicata nel seguito con "tag1", seleziona i nomi;
- la seconda, che verrà indicata nel seguito con "tag2", seleziona nomi e verbi.

Un ulteriore accorgimento che si è preso per evitare *topic drifting* è quello di sfruttare la pesatura dei termini messa a disposizione dalla sintassi per le query di Lucene. Si è deciso di assegnare peso 2 ai concetti estratti dall'algoritmo, e di lasciare invariata la pesatura dei termini dell'interrogazione originale. Ad esempio, un'interrogazione del tipo:

```
paint art gallery artist
```

viene modificata in:

```
art
```

e la query finale diventa:

```
+(text:art)^2 +(text:paint text:art text:gallery text:artist)
```

Per maggiori dettagli sulle varie versioni dell'algoritmo per il *topic gisting* si rimanda alla sezione B.4.2 in appendice.

3.4 Predittore di prolissità

Quello che si vuole ottenere è un predittore pre-reperimento di prolissità. I predittori pre-reperimento stimano le prestazioni di un'interrogazione prima che il reperimento venga eseguito e sono quindi indipendenti dalla lista ordinata dei risultati (essenzialmente sono indipendenti dalla ricerca). La **predizione pre-reperimento** si basa su diversi fattori, quali: la distribuzione dei termini all'interno della query, statistiche sulla collezione, e altri dati, che possono essere forniti da fonti esterne come ad esempio WordNet, che mette a disposizione informazioni sulle relazioni semantiche tra termini. Siccome questo tipo di predittori si basa su dati che sono disponibili in fase di indicizzazione, generalmente possono essere calcolati in modo più efficiente, determinando un minor *overhead* per il sistema di reperimento, anche se la **predizione post-reperimento** in genere fornisce risultati più accurati. Ovviamente, poiché i predittori post-reperimento si basano sui punteggi dei risultati, sono necessarie più fasi di reperimento per sfruttare le predizioni (20).

Se la query in input è prolissa, è probabile che essa porterà a scarsi risultati in fase di reperimento (21). Il predittore di prolissità può pertanto venir sfruttato dal sistema per migliorarne le performance. Ad esempio, si potrebbe consigliare all'utente di riassumere l'interrogazione, oppure utilizzare dei metodi di riduzione automatici.

Una semplice tecnica di classificazione prevede l'utilizzo di un **albero di decisione**, in cui ciascuna foglia rappresenta una classe, mentre i nodi interni contengono condizioni di test sugli attributi, che vengono utilizzate per separare i record da classificare in base alle loro caratteristiche. Una volta che si dispone dell'albero di decisione, classificare i record è triviale: partendo dalla radice è sufficiente verificare le condizioni necessarie per spostarsi verso uno dei figli, fino ad arrivare ad un nodo foglia.

Nelle sottosezioni seguenti si presentano quali *feature* sono stati associate alle interrogazioni e quale tecnica è stata utilizzata per la costruzione dell'albero di decisione.

3.4.1 Costruzione dell'albero di decisione attraverso Weka

Weka, acronimo di Waikato Environment for Knowledge Analysis, è un software open-source per l'apprendimento automatico, implementato in Java.

3.4.1.1 Scelta delle *feature*

Le *feature* che sono state associate a ciascuna interrogazione sono le seguenti:

- numero di termini: non è detto che una query lunga sia prolissa o che una query corta non lo sia, ma è ragionevole considerare riassumibili query molto lunghe;
- numero di sinonimi: a seconda della lunghezza della query, il numero di sinonimi è un indicatore del fatto che essa sia prolissa o meno; tale valore è stato calcolato come somma del numero di sinonimi per ciascun termine distinto della query.
- punteggio calcolato sulla base del numero di concetti estratti: si è deciso di utilizzare una variante del **metodo di Lesk**[1] (61). Anziché calcolare la similarità tra termini verificando il numero e il tipo di sovrapposizioni nelle definizioni degli stessi, si calcola la prolissità del testo considerando il numero di concetti contenuti nelle intersezioni effettuate dall'algoritmo presentato in 3.3.4.
- numero di ripetizioni: la presenza di molti termini ripetuti potrebbe influenzare la classificazione, in quanto tale valore può essere collegato alla lunghezza delle

[1] Il metodo di Lesk viene solitamente utilizzato per la *Word Sense Disambiguation*, ossia per selezionare il significato di una parola in relazione al contesto dato un insieme definito di possibilità. Riassumendo, l'algoritmo discrimina tra i diversi significati dei termini sulla base del numero e del tipo di sovrapposizioni nelle definizioni dei termini stessi.

interrogazioni; tale valore è stato calcolato come somma del numero di volte che ciascun termine distinto compare nell'interrogazione, se esso è maggiore di uno.

Nell'appendice B si riporta il file `.csv` dato in input a Weka per la costruzione dell'albero di decisione.

3.4.1.2 Tecnica utilizzata: J48 e cross-validation

Weka mette a disposizione molti algoritmi per la costruzioni di alberi di decisione. Si è scelto di utilizzare l'**algoritmo J48**, ossia l'implementazione Java dell'**algoritmo C4.5**, alla cui base c'è l'**algoritmo di Hunt** (58).

L'algoritmo di Hunt consiste nella costruzione di un albero di decisione in modo ricorsivo, partizionando i record che fanno parte del training set in sottoinsieme via via più piccoli. Sia D_t l'insieme dei training record associati al nodo t e $y = y_1, y_2, ..., y_c$ siano le etichette delle classi. Quello che l'algoritmo fa è:

Step 1: Se tutti i record in D_t appartengono alla stessa classe y_t, allora t è una foglia e viene etichettata con y_t.

Step 2: Se D_t contiene record che appartengono a più di una classe, viene selezionata una **condizione di test sugli attributi** per suddividere i record in sottoinsiemi. Viene quindi creato un nodo figlio per ciascun possibile risultato del test. L'algoritmo viene poi ricorsivamente applicato a ciascun nodo figlio.

Per evitare **overfitting** si è scelto di utilizzare come opzione per l'algoritmo J48 il **pruning**. Con questa opzione l'albero di decisione non viene sviluppato interamente sulla base del training set.

Per la valutazione del modello si è scelto di utilizzare la tecnica **cross-validation leave-one-out**, in cui ciascun test set contiene esattamente un record. Questo approccio ha il vantaggio di utilizzare il maggior numero di dati possibile per il training. Inoltre, gli insiemi di test sono tra loro mutualmente esclusivi e coprono effettivamente l'intero insieme di dati. Gli svantaggi di questo approccio sono però i tempi computazionali maggiori e la varianza delle performance stimate abbastanza elevata.

4

Sperimentazione

Scopo del capitolo è presentare i passi della sperimentazione che si è effettuata e i risultati ottenuti. Si è testata l'efficacia del predittore di prolissità e dei metodi di riduzione/espansione delle interrogazioni prolisse presentati nel capitolo precedente, verificando l'impatto che ha l'utilizzo di interrogazioni prolisse o meno a parità di modello di reperimento. Si è prestata particolare attenzione alla scelta della collezione sperimentale da utilizzare per i vari test, in modo da ottenere dei risultati ripetibili. Inoltre, i risultati così ottenuti sono confrontabili con sperimentazioni effettuate da altri, in particolare con quelle effettuate nel settore da Croft.

4.1 Fasi della sperimentazione

La sperimentazione viene suddivisa nelle seguenti fasi:

- fase preliminare, in cui si costruisce la collezione sperimentale;
- analisi sull'insieme di interrogazioni, in cui si confrontano i risultati ottenuti dal predittore con quelli che si ottengono da valutazioni umane;
- sperimentazione sulla collezione, al variare del metodo di elaborazione delle interrogazioni.

4.2 Collezione sperimentale

Si è deciso di utilizzare parte della collezione TREC Robust 2004. Tale collezione è composta da 250 interrogazioni e contiene circa mezzo milione di documenti tratti dal

Financial Times, il *Federal Register*, il *LA Times* e il *FBIS*. Tale collezione è conosciuta per contenere molte interrogazioni difficili, pertanto tale *data set* costituisce una sfida per i motori di ricerca attualmente diffusi e viene spesso utilizzata per ricerche nel settore (3).

4.2.1 Scelta dei topic

Si è deciso di selezionare i topic dalla collezione Robust04 dischi 4 e 5, poiché di tali topic si dispone dell'elenco di quelli considerati *difficili* per gli attuali motori di ricerca. Dei 150 topic a disposizione (da 300 a 450), ne sono stati selezionati 30, in modo che il numero di documenti rilevanti per ciascun topic fosse vicino al numero medio di documenti rilevanti, dopo aver verificato che fosse presente almeno un documento rilevante tra i giudizi di rilevanza messi a disposizione da TREC. Questa scelta è stata fatta tenendo conto che i giudizi di rilevanza vengono assegnati in modo tale da essere giudicati *completi*, nel senso che considerando l'insieme dei giudizi si può assumere che tutti i documenti rilevanti siano stati identificati (45), ma varia considerevolmente tale numero a seconda del topic. Si voleva pertanto evitare, con una scelta casuale di 30 topic, di selezionare quelli con un alto o un basso numero di documenti rilevanti, in quanto tale scelta avrebbe potuto compromettere la sperimentazione.

L'elenco dei topic scelti viene riportato in tabella 4.1.

Collezione	**# docs**	**Identificatori dei topic**
Robust04	528,155	302, 315, 323, 326, **330**, 333, 335, 340, **341**, **346**, 349, **350**, 351, 366, 368, **375**, 376, **378**, 384, 385, **389**, **399**, 402, 418, 421, 422, 428, **435**, **443**, 449

Tabella 4.1: Dati sulla collezione TREC e topic utilizzati. I topic riportati in grassetto sono quelli considerati *difficili*.

I topic della collezione sono quasi esclusivamente di tipo *informational* (con riferimento alla classificazione proposta da Broder, (7)) e trattano argomenti vari (politica, economia, medicina, cronaca...).

4.2.2 Costruzione dell'insieme di interrogazioni

I topic TREC costituiscono una semplificazione necessaria al fine di esprimere un'esigenza informativa. Tale esigenza informativa deve poi essere tradotta in un'interrogazione. Poiché la sperimentazione in esame vuole valutare l'impatto dell'utilizzo di interrogazioni prolisse, risulta un buon approccio costruire per una stessa esigenza informativa, ossia (approssimando) per uno stesso topic TREC, la versione prolissa e non dell'interrogazione. Questo perché si vogliono mettere in relazione gli effetti dell'utilizzo di interrogazioni prolisse sul reperimento di documenti rilevanti per una certa esigenza informativa, con quelli di interrogazioni non prolisse. Per poter confrontare i risultati, ottenuti dall'esecuzione del modello di reperimento al variare della query in input, è necessario pertanto disporre della versione prolissa e non dell'esigenza informativa.

Un approccio di facile implementazione per ottenere vari tipi di query come rappresentazione di una stessa esigenza informativa sarebbe quello di, dato un topic TREC, considerare il campo *title* come query corta, quello *description* come query lunga e quello *narrative* come query prolissa. Tale approccio non tiene però in considerazione degli aspetti fondamentali, che potrebbero influire negativamente sulla validità dei risultati ottenuti dalla ricerca in esame:

- Non è detto che un'interrogazione lunga sia prolissa o viceversa: è possibile che un'interrogazione corta, come può essere quella ricavabile dal campo title sia, seppur composta da pochi termini, ridondante.
- In generale il campo narrative non è prolisso, anzi ha un alto valore informativo, seppur composto da più frasi.

Sulla base di queste osservazioni si è deciso di costruire l'insieme di interrogazioni prolisse sfruttando parti di documenti rilevanti per il topic TREC. La scelta delle porzioni di documento da scegliere per la costruzione di un'interrogazione risulta un passaggio delicato, in quanto:

- la prolissità o meno della query dipende fortemente da tale scelta. Infatti, solo estraendo parti che trattano lo stesso aspetto si ottiene una query prolissa.
- la query risultato deve essere una trasposizione dell'esigenza informativa espressa nel Topic. Nel caso in cui vengano scelte parti di documento rilevante per il Topic

che non riguardano quest'ultimo, si potrebbe modificare l'insieme dei documenti rilevanti ai fini della ricerca.

Sempre sulla base delle osservazioni sopra riportate, si è deciso di sottoporre tutte le interrogazioni a dei giudici, i quali ne hanno stabilito la prolissità o meno, secondo la **definizione di *prolisso***, che si era presentata nel cap. 1, ma che si riporta nuovamente nel seguito per maggior chiarezza: *troppo lungo e particolareggiato o che utilizza un numero di parole superiore al necessario; ridondante; riassumibile senza perdita di informazione.*

Si è tenuto presente il fatto che una query può risultare prolissa o meno a seconda dell'esigenza informativa. Pertanto, poichè tale esigenza viene riassunta nel Topic TREC (vedi fig. 4.1), per ciascuna interrogazione sottoposta a valutazione è stato fornito il Topic a cui fa riferimento.

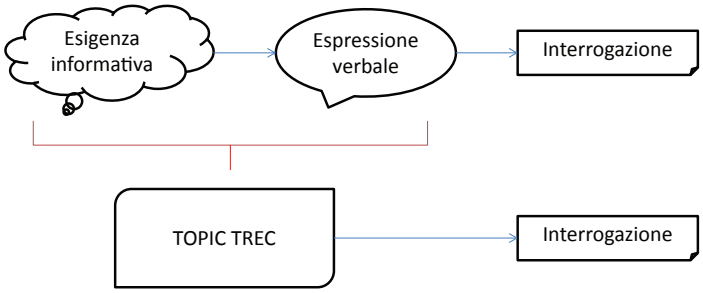


Figura 4.1: I Topic TREC cercano di descrivere le esigenza informativa.

Riassumendo, a ciascun giudice sono stati forniti:

- la definizione di prolisso, da tener presente per tutto il corso della valutazione;
- una lista di interrogazioni da valutare;
- il topic relativo a ciascuna interrogazione.

Per ovviare al problema di giudizi discordanti, si è stabilito di valutare ciascuna interrogazione 3 volte, in modo da poter sempre arrivare ad una classificazione sulla base di una valutazione a maggioranza. Per i risultati delle valutazioni si rimanda all'appendice B.

Al termine di questa fase si sono ottenuti quattro insiemi di interrogazioni:

- corte: ottenute dal campo *title*;
- lunghe: ottenute dal campo *description*;
- lunghe e prolisse: ottenute da documenti rilevanti per il Topic;
- corte e prolisse: ottenute in modo analogo.

4.2.3 Estrazione delle *feature* e albero di decisione

Nell'appendice B si riporta il file `.csv` dato in input a Weka per la costruzione dell'albero di decisione.

L'albero di decisione che si è ottenuto ed utilizzato per la classificazione è riportato in figura 4.2.

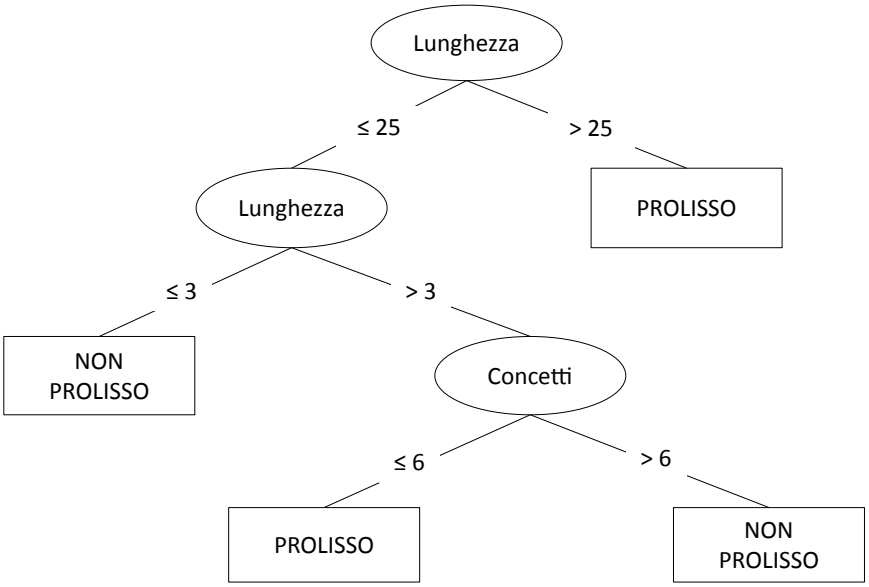


Figura 4.2: Albero di decisione realizzato con Weka utilizzando J48.

In tabella 4.2 si riportano alcuni dati sul modello, mentre in tabella 4.3 i dettagli sull'accuratezza per classe. La statistica K, che è un valore statistico calcolato sulla base della matrice di confusione che rappresenta il grado di accuratezza e affidabilità in una classificazione statistica, indica che la concordanza è discreta $(0,4 < k < 0,6)$. Visto il valore della ROC-Area il test si può considerare moderatamente accurato.

Correctly Classified Instances	96	80%
Incorrectly Classified Instances	24	20%
Kappa statistic	0.6	

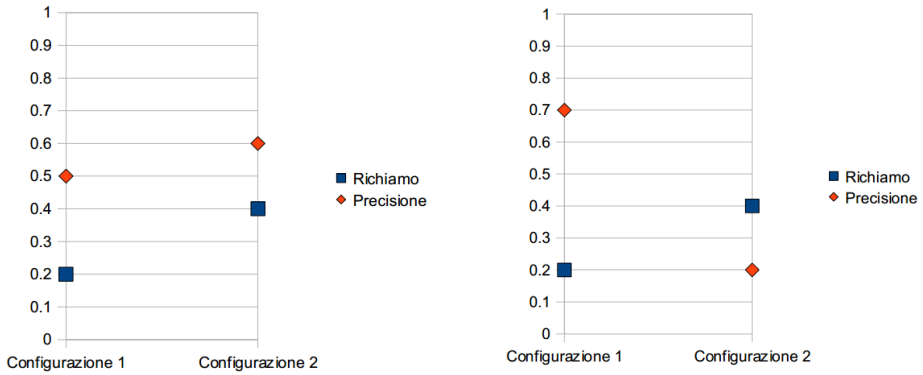
Tabella 4.2: Valutazione del classificatore

	TP Rate	FP Rate	Precision	Recall	F-Measure	ROC Area	Class
	0.817	0.217	0.79	0.817	0.803	0.801	NO
	0.783	0.183	0.81	0.783	0.797	0.801	YES
Weighted Avg.	0.8	0.2	0.8	0.8	0.8	0.801	

Tabella 4.3: Dettagli sull'accuratezza per classe. La classe NO indica le interrogazioni prolisse, la YES quelle non prolisse.

4.3 Misure di valutazione

La scelta delle misure di valutazione è stata fatta sulla base di varie osservazioni (41). Innanzitutto, richiamo e precisione ottenuti con una configurazione, non sono sempre confrontabili con quelli ottenuti con un'altra configurazione. Se l'efficacia della configurazione i è espressa con una coppia di valori (P(i), R(i)), è impossibile ordinare due configurazioni i e j quando P(i) > P(j) e R(j) < R(j) (fig. 4.3). Inoltre, nel caso di precisioni multiple associate ad un richiamo (e viceversa) e all'assenza di richiami per delle precisioni è necessario ricorrere a interpolazione e all'utilizzo di medie aritmetiche.



(a) Confrontabili

(b) Non confrontabili

Figura 4.3: Due configurazioni a confronto: misure di valutazione confrontabili e non.

Visti gli svantaggi di precisione e richiamo, e considerati anche gli inconvenienti della misura E (è difficilmente interpretabili e dipende dalla scelta di α), si è imposta di recente la media delle precisioni medie non interpolate su un insieme di interrogazioni (MAP - *Mean Average Precision*). La precisione media non interpolata (*Noninterpolated Average Precision*, AP) è la media dei valori di precisione calcolati ad ogni documento rilevante trovato nella lista.

Prendendo come riferimento anche le altre ricerche nel settore, in particolare (3), come misure di valutazione dell'efficacia si è scelto di stimare:

- **mean average precision (MAP)**, che ha una serie di proprietà che la rendono preferibile alle altre misure: innanzitutto, non dipende da soglie arbitrarie; inoltre, i documenti rilevanti trovati nelle prime posizioni contribuiscono maggiormente alla MAP dei documenti rilevanti trovati nelle posizioni successive.

- **precision at five (P@5)**, che è la frazione di documenti rilevanti tra i primi cinque documenti reperiti.

I valori delle misure di valutazione che verranno riportati e commentati nel paragrafo seguente sono stati calcolati utilizzando `ireval`, un tool messo a disposizione dal **progetto Lemur** (1).

4.4 Risultati degli esperimenti

Poiché le interrogazioni prolisse sono state costruite a partire da documenti rilevanti, si è deciso di eliminare tali documenti dalla collezione, ossia si è deciso di effettuare la sperimentazione sulla **collezione residua**.

Le valutazioni dei giudici sugli insiemi di interrogazioni hanno determinato l'esclusione di alcune query dalla sperimentazione. In particolare, sono state valutate lunghe e prolisse 27 interrogazioni, mentre corte e prolisse 23. Piuttosto che selezionare solo i topic di cui si avevano a disposizione tutti e 4 i tipi di interrogazione, si è preferito distinguere tra insieme di topic di cui si dispone dell'interrogazione lunga e prolissa e quello con interrogazione corta e prolissa, per non ridurre eccessivamente il numero di topic. Si riportano nelle tabelle 4.4, 4.5, 4.6 i risultati relativi ai topic di cui si hanno a disposizione le corrispondenti interrogazioni lunghe e prolisse. In tabella 4.4 vengono riportati i risultati degli esperimenti condotti sull'insieme completo di interrogazioni,

suddivisi per categoria, mentre nelle tabelle 4.5 e 4.6 vengono presentati i risultati riguardanti rispettivamente le interrogazioni facili e quelle difficili. Nelle tabelle 4.7, 4.8, 4.9 vengono riportati i risultati relativi ai topic di cui si hanno a disposizione le corrispondenti interrogazioni corte e prolisse, distinguendo rispettivamente tra insieme completo di interrogazioni, interrogazioni facili e interrogazioni difficili.

Si descrive brevemente a cosa si fa riferimento con i nomi che si è deciso di associare ai vari modelli di reperimento:

- **stopword e stemming**: l'unica operazione che si effettua sulle interrogazioni prima di processarle è l'eliminazione delle *stopword* e lo *stemming*;
- **aggiunta dei sinonimi**: dopo aver eliminato le *stopword* e aver fatto lo *stemming* dei termini, si aggiungono all'interrogazione i sinonimi corrispondenti a ciascun termine, utilizzando l'algoritmo presentato nel par. 3.3.3;
- **aggiunta dei concetti**: oltre all'eliminazione delle *stopword* e allo *stemming* dei termini, si aggiungono all'interrogazioni i termini ottenuti utilizzando l'algoritmo base per il *topic gisting*, presentato nel par. 3.3.4, ossia senza selezione dei termini da dare in input all'algoritmo;
- **sinonimi+concetti**: i due punti precedenti vengono combinati, aggiungendo prima i sinonimi e poi i concetti collegati (conta l'ordine);
- **concetti+sinonimi**: viene fatto lo stesso del punto precedente, invertendo l'ordine delle operazioni;
- **tag+concetti**: dopo aver eliminato le *stopword* e aver fatto lo *stemming* dei termini, viene applicato l'algoritmo per il *topic gisting* utilizzando come input solo i termini che corrispondono a nomi;
- **tag2+concetti**: dopo aver eliminato le *stopword* e aver fatto lo *stemming* dei termini, viene applicato l'algoritmo per il *topic gisting* utilizzando come input solo i termini che corrispondono a nomi e a verbi.

I risultati riportati in grassetto indicano le migliori performance per la categoria in esame. È stata verificata la significatività statistica delle differenze tra i risultati

utilizzando il **test dei segni**[1] imponendo $p < 0.05$ (i valori indicati con un asterisco hanno $p < 0.1$). Il test è stato calcolato in modo automatico sfruttando la funzione `binom.test` messa a disposizione dall'**ambiente software R**. Le differenze statisticamente significative con le interrogazioni corte e con quelle lunghe sono state contrassegnate rispettivamente con una c e una l, mentre quelle con la baseline sono indicate con una b.

modello di reperimento	**interrogazioni**					
	corte		**lunghe**		**lunghe e prolisse**	
	MAP	**P@5**	**MAP**	**P@5**	**MAP**	**P@5**
stopword e stemming	$18{,}34^{l}$	42,96	$12{,}79^{c}$	40	$13{,}97^{c}$	45,19
aggiunta dei sinonimi	$\mathbf{18{,}65}^{l}_{b}$	42,96	$13{,}01^{c}$	38,52	14	43,70
aggiunta dei concetti	$18{,}34^{l}$	$42{,}96^{l}$	$12{,}91^{c}$	$37{,}78^{c}_{b}$	$13{,}86^{c}$	$\mathbf{51{,}54}_{b}$
sinonimi + concetti	$\mathbf{18{,}65}^{l}_{b}$	42,96	$12{,}99^{c}$	37,04	$12{,}78^{c}$	48,46
concetti + sinonimi	$18{,}62^{l}$	**43,70**	$13{,}06^{c}$	38,52	$13{,}29^{c}$	50^{c}
tag + concetti	$15{,}46_{b}$	$33{,}33^{l}_{b}$	$14{,}80_{b}$	$44{,}44^{c}$	$\mathbf{16{,}65}_{b}$	45,93
tag2 + concetti	$18{,}19^{l}$	42,22	$\mathbf{15{,}06}^{c}_{b}$	**45,19**	14,75	46,67

Tabella 4.4: Valori percentuali per l'**insieme di interrogazioni rispetto ai topic di cui si ha a disposizione l'interrogazione lunga e prolissa** calcolati sulla collezione residua. Si riportano in grassetto i risultati migliori per ciascuna categoria. Le differenze statisticamente significative con le interrogazioni corte e con quelle lunghe sono state contrassegnate rispettivamente con una c e una l, mentre quelle con la baseline sono indicate con una b (si è utilizzato il test dei segni con $p < 0,05$ e sono indicati con un asterisco i casi in cui $p < 0,1$).

Confrontando le tabelle è evidente come effettivamente i topic riportati in grassetto in tabella 4.1 risultino *più difficili* rispetto agli altri, ossia che si ottengono scarsi risultati di reperimento.

[1] Il *sign test* è un test non parametrico, ossia un tipo di test impiegato qualora non si abbiano informazioni preliminari sul tipo e sulla forma della distribuzione e/o non si possano fare assunzioni di normalità. Può essere utilizzato per rigettare l'ipotesi che "non c'è differenza nella mediana" tra due distribuzioni continue X e Y di cui si possono considerare coppie di campioni. Sia $p = Pr(X > Y)$ e sia la *null hypothesis* $H_0 : p = 0,50$. Prese n coppie indipendenti di campioni $\{(x_1, y_1), (x_2, y_2), ..., (x_n, y_n)\}$ (con $x_i \neq y_i$), sia W il numero di coppie per cui si ha $y_i - x_i > 0$. Assumendo che H_0 sia vera, allora W segue una distribuzione binomiale $W \sim b(n, 0,5)$. Per calcolare la significatività statistica è quindi sufficiente applicare il test binomiale, tenendo presente che per $n > 25$ si può approssimare la distribuzione binomiale con una normale.

modello di reperimento	interrogazioni facili					
	corte		lunghe		lunghe e prolisse	
	MAP	P@5	MAP	P@5	MAP	P@5
stopword e stemming	$23{,}30^{l}$	53,68	$15{,}23^{c}$	42,11	$16{,}34^{c}$	45,26
aggiunta dei sinonimi	$\mathbf{23{,}80}^{l}$	$53{,}68_{b*}$	$15{,}62^{c}$	$38{,}95_{b*}$	$16{,}26^{c}$	45,26
aggiunta dei concetti	$23{,}30^{l}$	53,68	$15{,}33^{c}$	38,91	$15{,}10^{c}$	$\mathbf{50{,}53}_{b*}$
sinonimi + concetti	$\mathbf{23{,}80}_{b*}$	53,68	15,33	$38{,}95_{b*}$	$15{,}10^{c}_{b*}$	50,53
concetti + sinonimi	$23{,}69^{l}$	**54,74**	$15{,}58^{c}$	37,89	$14{,}09^{c}$	48,42
tag + concetti	$19{,}58_{b}$	$43{,}16_{b}$	$17{,}93_{b}$	$\mathbf{48{,}42}_{b}*$	$\mathbf{19{,}73}_{b}$	45,26
tag2 + concetti	$23{,}30^{l}$	53,68	$\mathbf{17{,}94}^{c}_{b}$	47,37	$16{,}91^{c}$	45,26

Tabella 4.5: Valori percentuali per l'**insieme di interrogazioni *facili* rispetto ai topic di cui si ha a disposizione l'interrogazione lunga e prolissa** calcolati sulla collezione residua. Si riportano in grassetto i risultati migliori per ciascuna categoria. Le differenze statisticamente significative con le interrogazioni corte e con quelle lunghe sono state contrassegnate rispettivamente con una c e una l, mentre quelle con la baseline sono indicate con una b (si è utilizzato il test dei segni con $p < 0,05$ e sono indicati con un asterisco i casi in cui $p < 0,1$).

modello di reperimento	interrogazioni difficili					
	corte		lunghe		lunghe e prolisse	
	MAP	P@5	MAP	P@5	MAP	P@5
stopword e stemming	**6,58**	**17,50**	7	35	8,33	45
aggiunta dei sinonimi	6,40	**17,50**	6,81	37,50	8,62	40
aggiunta dei concetti	**6,58**	**17,50**	$7{,}14_{b}$	35	**10,49**	**54,29**
sinonimi + concetti	6,40	**17,50**	$6{,}82_{b}$	35	9,24	48,57
concetti + sinonimi	**6,58**	**17,50**	$7{,}16_{b}$	35	9,53	51,43
tag + concetti	5,68	10^{l}	7,38	35^{c}	$9{,}32_{b}$	$47{,}50^{c}$
tag2 + concetti	$6{,}07^{l}$	15	$\mathbf{8{,}22}^{c}_{b}$	**40**	$9{,}63^{c}$	50

Tabella 4.6: Valori percentuali per l'**insieme di interrogazioni *difficili* rispetto ai topic di cui si ha a disposizione l'interrogazione lunga e prolissa** calcolati sulla collezione residua. Si riportano in grassetto i risultati migliori per ciascuna categoria. Le differenze statisticamente significative con le interrogazioni corte e con quelle lunghe sono state contrassegnate rispettivamente con una c e una l, mentre quelle con la baseline sono indicate con una b (si è utilizzato il test dei segni con $p < 0,05$ e sono indicati con un asterisco i casi in cui $p < 0,1$).

Considerando il modello di reperimento base, ossia quello in cui si applica semplicemente l'eliminazione delle *stopword* e lo *stemming*, si è osservato come le interrogazioni corte (prolisse e non) abbiano in generale performance migliori rispetto a quelle lunghe (prolisse e non), in accordo con i risultati di Bendersky e Croft. Se però si analizza la difficoltà delle esigenze informative, facendo riferimento ai risultati riportati in tabella 4.5 e 4.6, in cui vengono riassunte le misure riguardanti rispettivamente le interrogazioni facili e quelle difficili, si può notare come per l'insieme di interrogazioni difficili (che sono però in numero limitato) le query prolisse risultino più efficaci rispetto alle altre, anche se i risultati rimangono di molto inferiori alla media complessiva.

modello di reperimento	**interrogazioni**					
	corte		**lunghe**		**corte e prolisse**	
	MAP	**P@5**	**MAP**	**P@5**	**MAP**	**P@5**
stopword e stemming	$19{,}10^{l}$	45,22	$12{,}37^{c}$	40	$19{,}79^{l}$	54,78
aggiunta dei sinonimi	$\mathbf{19{,}46}^{l}$	45,22	$12{,}68^{c}$	38,26	$\mathbf{21{,}75}^{l}$	$\mathbf{60{,}87}^{c,l}$
aggiunta dei concetti	$19{,}10^{l}$	45,22	$12{,}43^{c}$	36,52	$19{,}52^{l}$	53,04
sinonimi + concetti	$\mathbf{19{,}46}^{l}$	45,22	$12{,}65^{c}_{b*}$	36,52	$21{,}08^{l}$	$55{,}65^{c*,l*}$
concetti + sinonimi	$19{,}43^{l}$	**46,09**	$12{,}65^{c}_{b*}$	37,39	$20{,}06^{l}$	$54{,}78^{l}$
tag + concetti	$15{,}49^{l}_{b}$	$34{,}78_{b}$	$14{,}39^{c}_{b}$	40,87	$17{,}29^{l*}_{b}$	$52{,}17^{c}$
tag2 + concetti	$18{,}92^{l}$	44,35	$\mathbf{15}^{c}_{b}$	**41,74**	$20{,}88^{l}_{b*}$	$56{,}52^{c,l}$

Tabella 4.7: Valori percentuali per l'**insieme di interrogazioni rispetto ai topic di cui si ha a disposizione l'interrogazione corta e prolissa** calcolati sulla collezione residua. Si riportano in grassetto i risultati migliori per ciascuna categoria. Le differenze statisticamente significative con le interrogazioni corte e con quelle lunghe sono state contrassegnate rispettivamente con una c e una l, mentre quelle con la baseline sono indicate con una b (si è utilizzato il test dei segni con $p < 0,05$ e sono indicati con un asterisco i casi in cui $p < 0,1$).

L'aggiunta dei sinonimi ha determinato un aumento di prestazioni per le interrogazioni corte (prolisse e non), mentre i risultati sono peggiorati per quelle lunghe (prolisse e non).

Come era prevedibile, l'utilizzo dell'algoritmo base per l'estrazione del *succo del discorso* non modifica le prestazioni per le interrogazioni corte, mentre porta ad un miglioramento di performance in particolare per le interrogazioni lunghe e prolisse.

Si sono valutate poi le possibili combinazioni degli algoritmi per l'estrazione dei sinonimi e dei concetti rilevanti (riportati nelle tabelle 4.4, 4.5, 4.6 come "sinonimi +

modello di reperimento	interrogazioni facili					
	corte		lunghe		corte e prolisse	
	MAP	P@5	MAP	P@5	MAP	P@5
stopword e stemming	$25{,}24^{l}$	57,33	$16{,}02^{c}$	45,33	$24{,}49^{l}$	58,67
aggiunta dei sinonimi	$\mathbf{25{,}88}^{l*}$	57,33	$16{,}59^{c*}$	42,67	$\mathbf{27{,}54}^{c*,l}_{b*}$	$\mathbf{70{,}67}^{c,l}_{b*}$
aggiunta dei concetti	$25{,}24^{l}$	57,33	$16{,}09^{c}$	41,33	$24{,}03^{l}$	56
sinonimi + concetti	$\mathbf{25{,}88}^{l}$	57,33	$16{,}54^{c}$	$41{,}33_{b*}$	$26{,}54^{l}$	$62{,}67^{l}$
concetti + sinonimi	$25{,}74^{l}$	**58,67**	$16{,}41^{c}$	42,67	$24{,}85^{l}$	60^{l}
tag + concetti	$20{,}18^{l*}_{b}$	45,33	$18{,}59^{c*}_{b*}$	46,67	$21{,}45_{b*}$	$62{,}67^{c,l*}$
tag2 + concetti	$25{,}24^{l}$	57,33	$\mathbf{19{,}68}^{c}_{b}$	**50,67**	$26{,}03^{l}_{b*}$	60

Tabella 4.8: Valori percentuali per l'**insieme di interrogazioni *facili* rispetto ai topic di cui si ha a disposizione l'interrogazione corta e prolissa** calcolati sulla collezione residua. Si riportano in grassetto i risultati migliori per ciascuna categoria. Le differenze statisticamente significative con le interrogazioni corte e con quelle lunghe sono state contrassegnate rispettivamente con una c e una l, mentre quelle con la baseline sono indicate con una b (si è utilizzato il test dei segni con $p < 0,05$ e sono indicati con un asterisco i casi in cui $p < 0,1$).

modello di reperimento	interrogazioni difficili					
	corte		lunghe		corte e prolisse	
	MAP	P@5	MAP	P@5	MAP	P@5
stopword e stemming	$\mathbf{7{,}59}^{l}$	**22,50**	$5{,}53^{c}$	**30**	$10{,}98^{l}$	$47{,}50^{c*,l*}$
aggiunta dei sinonimi	$7{,}41^{l}$	**22,50**	$5{,}34^{c}$	**30**	$10{,}89^{l}$	$42{,}50^{c*}$
aggiunta dei concetti	$\mathbf{7{,}59}^{l}$	**22,50**	$5{,}58^{c}_{b}$	27,50	$11{,}06^{l}$	$47{,}50^{c*,l*}$
sinonimi + concetti	$7{,}41^{l}$	**22,50**	$5{,}35^{c}_{b}$	27,50	$10{,}84^{l}$	$42{,}50^{c*}$
concetti + sinonimi	$\mathbf{7{,}59}^{l}$	**22,50**	$5{,}61^{c}_{b}$	27,50	$11{,}06^{l}$	$45^{c*,l*}$
tag + concetti	6,69	15^{l*}	**6,50**	$\mathbf{30}^{c*}$	9,50	32,50
tag2 + concetti	$7{,}08^{l}$	20	$6{,}21^{c}_{b}$	25	$\mathbf{11{,}22}^{l}$	$\mathbf{50}^{c,l*}$

Tabella 4.9: Valori percentuali per l'**insieme di interrogazioni *difficili* rispetto ai topic di cui si ha a disposizione l'interrogazione corta e prolissa** calcolati sulla collezione residua. Si riportano in grassetto i risultati migliori per ciascuna categoria. Le differenze statisticamente significative con le interrogazioni corte e con quelle lunghe sono state contrassegnate rispettivamente con una c e una l, mentre quelle con la baseline sono indicate con una b (si è utilizzato il test dei segni con $p < 0,05$ e sono indicati con un asterisco i casi in cui $p < 0,1$).

concetti" e "concetti + sinonimi", a seconda dell'ordine di esecuzione). Nel secondo caso le misure rilevate mostrano come la P@5 per le interrogazioni prolisse risulti maggiore di quella per le interrogazioni corte in modo statisticamente significativo.

Si è poi valutato l'algoritmo per l'estrazione del *succo del discorso* nel caso in cui venga fatta selezione dei concetti in input attraverso un *tagger*. Si è indicato con "tag" la selezione dei nomi e con "tag2" la selezione di nomi e verbi. Nel primo caso si è ottenuto un incremento della MAP di $2,39$ punti percentuali. Si riporta in fig. 4.4 un grafico a dispersione dei valori di MAP ottenuti per le interrogazioni lunghe e prolisse utilizzando l'algoritmo base e quello indicato con "tag+concetti", e in fig. 4.5 quelli ottenuti per le interrogazioni lunghe utilizzando l'algoritmo "tag2+concetti". Tali grafici (fig. 4.4 e fig. **??**) mostrano come gli algoritmi proposti risultino efficaci nel caso sia necessario elaborare interrogazioni lunghe (prolisse e non, rispettivamente).

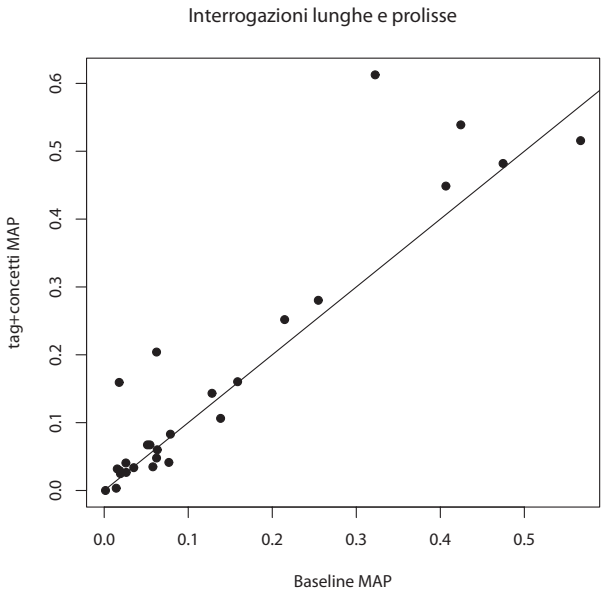


Figura 4.4: Confronto tra i valori di MAP ottenuti utilizzando l'algoritmo base e quello "tag+concetti" nel caso di interrogazioni lunghe e prolisse.

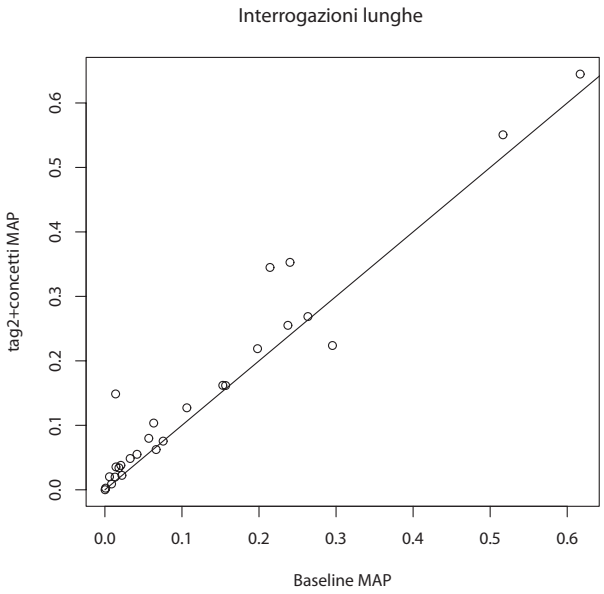


Figura 4.5: Confronto tra i valori di MAP ottenuti utilizzando l'algoritmo base e quello "tag2+concetti" nel caso di interrogazioni lunghe.

5

Conclusioni

Nel presente lavoro di tesi si è voluto indagare in che misura l'utilizzo di interrogazioni prolisse influisca sulle performance di un sistema di reperimento. A partire dalla **collezione TREC Robust 2004**, nota per contenere esigenze informative difficili per i comuni motori di reperimento, è stata costruita una collezione sperimentale contenente 4 insiemi di interrogazioni: corte, lunghe, lunghe e prolisse, corte e prolisse. Le interrogazioni corte e quelle lunghe sono state ricavate rispettivamente dal campo *title* e *description* dei topic TREC, mentre quelle prolisse sono state costruite a partire da documenti rilevanti per il topic corrispondente. Le interrogazioni sono state quindi sottoposte a dei giudici per stabilirne la categoria di appartenenza. Tale approccio per la costruzione della collezione sperimentale risulta differente da quello classico adottato per le ricerche nel settore, che si limitano a costruire le interrogazioni a partire dai campi del topic TREC, senza verificarne l'effettiva prolissità.

Poiché le interrogazioni prolisse sono state costruite a partire da documenti rilevanti, la sperimentazione è stata effettuata sulla collezione residua. Si è effettuato il **test dei segni** per verificare la significatività statistica dei risultati ottenuti in fase di sperimentazione, utilizzando il **software statistico R**. Si riportano i risultati ottenuti nei prossimi paragrafi.

Utilizzando il modello di reperimento di base, costruito utilizzando **Apache Lucene**, in cui si applica semplicemente l'eliminazione delle *stopword* e lo *stemming*, si è osservato come le interrogazioni corte (prolisse e non) abbiano in generale performance migliori rispetto a quelle lunghe (prolisse e non), in accordo con i risultati di Bendersky e Croft.

La sperimentazione ha inoltre confermato come effettivamente alcuni topic della collezione Robust 2004 determinino scarse performance nel reperimento.

Si è proposto un nuovo approccio all'elaborazione di query prolisse che sfrutta le relazioni semantiche in esse contenute. Utilizzando un indice in **Solr** delle relazioni contenute in **ConceptNet 5**, è stato possibile implementare vari algoritmi, riconducibili a due categorie:

- un algoritmo per l'estrazione dei sinonimi dei termini contenuti nelle interrogazioni, che li inserisce in quest'ultime sottoforma di *clausole OR* (sfruttando la sintassi per le query di Lucene);
- vari algoritmi per il *topic gisting*, che espandono le interrogazioni aggiungendo a quest'ultime i termini che costituiscono il succo del discorso, assegnando ad essi un peso maggiore rispetto a quelli contenuti nella query originale. Quello che differenzia le varie versioni di questi algoritmi è la strategia utilizzata per selezionare i termini da dare in input all'algoritmo. Si è proposta una versione di base in cui si selezionano tutti i termini dell'interrogazione e due versioni che selezionano solo alcune parti del discorso. L'etichettatura delle parti del discorso, necessaria per la selezione dei termini da utilizzare come input per tali algoritmi è stata effettuata utilizzando lo **Stanford Parser**.

In fase di sperimentazione si è osservato come l'utilizzo del primo algoritmo incrementa le prestazioni delle query corte (prolisse e non). In particolare, per le interrogazioni corte e prolisse si è passati da un valore di MAP del 19,79% a uno del 21,75%, mentre per quanto riguarda la misura di valutazione P@5 si è passati da 54,78% a 60,87%. La seconda tipologia di algoritmo è risultata efficace principalmente per le interrogazioni lunghe (prolisse e non). In particolare, per le interrogazioni lunghe la versione dell'algoritmo che è risultata più efficace al fine di incrementare la MAP è quella che seleziona i termini che corrispondo a nomi e verbi (tale versione dell'algoritmo è stata indicata con "tag2+concetti" nella sezione 4.4), mentre per le interrogazioni lunghe e prolisse si ha avuto il maggiore aumento di MAP per la versione "tag+concetti", ossia quella che seleziona unicamente i nomi (la MAP è passata da un valore del 13,97% a 16,65%, con un incremento statisticamente significativo). Per quanto riguarda invece la misura P@5 i maggiori miglioramenti per le interrogazioni lunghe e prolisse si sono riscontrati utilizzando la versione base dell'algoritmo, ossia selezionando tutti i

termini dell'interrogazione (tale versione è stata indicata con "aggiunta dei concetti"): si è passati da 45,19% a 51,54%, con un incremento statisticamente significativo. È interessante notare come in generale considerando l'insieme delle interrogazioni facili e quello delle interrogazioni difficili non cambia l'algoritmo che determina il maggiore aumento di prestazioni.

Si è costruito un classificatore di prolissità utilizzando **Weka**. Tale classificatore ha lo scopo di distinguere, data in input un'interrogazione, se essa appartiene alla classe "prolisso" oppure a quella complementare. Le *feature* che si sono prese in considerazione sono: numero di termini, numero di termini ripetuti, numero di sinonimi, numero di concetti collegati. In particolare, quest'ultimo valore è stato calcolato utilizzando una versione modificata del **metodo di Lesk**, tenendo conto del numero di termini contenuti nell'output della versione base dell'algoritmo per il *topic gisting*. Il classificatore è un albero di decisione costruito secondo l'algoritmo C4.5 ed è stato valutato utilizzando la **tecnica di cross-validation leave-one-out**. Sulla base dei valori della statistica K e della ROC Area, che sono pari rispettivamente a 0,6 e 0,801, tale classificatore può essere considerato un predittore di discreta affidabilità.

5.1 Sviluppi futuri

Si riportano schematicamente nel seguito alcuni spunti per sperimentazioni future:

- La presente analisi ha riguardato esigenze informative di tipo *informational*. Si ritiene che esista una correlazione tra tipo di esigenza informativa (secondo la classificazione proposta da Broder) e performance nel reperimento, nel caso si utilizzino interrogazioni prolisse o meno.

- Nella presente sperimentazione si è sfruttata la pesatura dei termini prevista dalla sintassi per le query di Lucene. Per i termini che si sono aggiunti alle interrogazioni in seguito al *topic gisting* si è utilizzato un peso pari a 2, lasciando invece invariato il peso dei termini della query originale. Risulterebbe interessante verificare in che modo la variazione dei pesi influisca sulle performance del sistema.

- Le varie versioni dell'algoritmo per il *topic gisting* sono state implementate utilizzando le relazioni semantiche messe a disposizione da ConceptNet 5. In particolare, tra i diversi tipi di relazioni messe a disposizione dal database, si è deciso di

sfruttare `/r/SimilarTo` e `/r/RelatedTo`. Ovviamente il tipo di relazione selezionata influisce sui risultati dell'algoritmo. Sarebbe quindi interessante verificare quale sia la scelta ottimale per il tipo di relazioni da selezionare per l'estrazione dei concetti collegati.

- Le varie versioni dell'algoritmo per il *topic gisting* sono costruite in modo ricorsivo, suddividendo ad ogni passo l'input in due parti, in modo da aumentare l'efficacia dell'algoritmo, a discapito dell'efficienza. Intuitivamente esiste un trade-off tra queste due misure, al variare del numero di sottoproblemi che ricorsivamente si vanno a creare.

- Studi quali quello condotto da Salton e Buckley (53) hanno dimostrato come l'utilizzo di *Relevance Feedback* risulti più efficace nel caso di interrogazioni corte. Si ritiene che l'utilizzo di una strategia di *Relevance Feedback* applicata ai termini estratti dall'algoritmo per il *topic gisting* (che in generale sono in numero limitato rispetto a quelli dell'interrogazione originale) aumenterebbe l'efficacia del reperimento.

Appendice A

Apache Solr

Solr (si pronuncia *Solar*) è un *Enterprise Search Server* open source, estensione delle librerie java di Apache Lucene, ossia una piattaforma utilizzabile come sistema di reperimento in ambito aziendale.

Viene eseguito come server di ricerca indipendente all'interno di un contenitore servlet, come ad esempio Jetty o Tomcat. I documenti, che possono essere in formato XML, JSON o binario, vengono indicizzati passandoli a Solr secondo il protocollo HTTP. Sempre tale protocollo viene utilizzato per il reperimento delle informazioni.

Il sistema di configurazione permette di adattare Solr all'applicazione d'interesse, senza la necessità di scrivere codice java. Proprio per l'ampia possibilità di configurazione e le feature che mette a disposizione, Solr viene spesso utilizzato per effettuare ricerche all'interno di siti web.

Nel seguito della trattazione si presenterà come eseguire le operazioni necessarie per lo svolgimento della sperimentazione effettuata nel corso della tesi. Per maggiori informazioni su Solr ed il suo utilizzo si rimanda a (55).

A.1 Eseguire Solr su Jetty

Il modo più semplice per configurare un'istanza di Solr è quello di prendere spunto dall'esempio proposto con la distribuzione Solr, che mostra come utilizzare il servlet Jetty fornito con quest'ultima. Sarebbe comunque possibile configurare ed eseguire Solr su un servlet a sé stante, seguendo i semplici passaggi seguenti (26):

- scaricare Jetty da `http://jetty.codehaus.org/jetty/` ed installarlo.

- copiare i file `jetty.xml` e `webdefault.xml` dalla cartella `example/etc` nella distribuzione Solr alla cartella di configurazione di jetty (in genere chiamata `/etc/jetty`). A questo punto Jetty è configurato.

- per rendere Solr eseguibile su Jetty è però necessario copiare il file `solr.war` nella cartella `webapps` di Jetty e configurare Solr modificando i file `schema.xml` e `solrconfig.xml`.

- a questo punto è possibile eseguire Solr utilizzando un comando del tipo:

  ```
  /etc/init.d/jetty start
  ```

  accedendo al server attraverso il browser all'indirizzo: `http://localhost:8983/solr`.

Ovviamente sarebbe possibile una configurazione più complessa, ad esempio impostando la porta utilizzata da Jetty o impostando la dimensione del Buffer in cui vengono memorizzate le query da elaborare.

Jetty non è l'unico server per applicazioni su cui può essere eseguito Solr. L'utilizzo di un altro contenitore di servlet può essere necessario nel caso in cui siano presenti anche altre applicazioni in esecuzione, ad esempio su Apache Tomcat.

A.2 Indicizzare i dati in formato JSON

Affinché sia possibile indicizzare dati in formato JSON è necessario assicurarsi che nel file `solrconfig.xml` sia presente la riga:

```
<requestHandler name="/update/json" class="solr.JsonUpdateRequestHandler"/>
```

Per indicizzare i dati è possibile utilizzare il comando Linux `curl`, che permette di trasferire dati utilizzando vari protocolli, nel modo seguente:

```
curl http://localhost:8983/solr/update/json --data-binary @<File.json> -H "
    Content-type:application/json"
```

Se si utilizza un sistema Windows, è possibile scaricare l'applicazione cURL da `http://curl.haxx.se/download.html` ed utilizzare quindi il comando riportato sopra.

A.2.1 Possibili problemi

Il comando `curl` potrebbe non andare a buon fine. Si consiglia di prestare attenzione a digitare tale comando non riportando l'indirizzo di destinazione tra virgolette (altrimenti si potrebbe riscontrare l'errore: *curl non supporta il protocollo http*). Inoltre, nei sistemi Windows l'argomento di `-H` deve essere riportato tra virgolette (gli apici singoli non vengono riconosciuti come caratteri validi). Infine, la dimensione dei file `.json` da importare può creare dei problemi del tipo *out of memory error*, in particolare se si utilizza Cygwin (nei sistemi Windows). Per dare un'idea delle dimensioni che possono assumere questi file per l'importazione, il file `conceptnet4.json` occupa 1,21GB, contro i 215MB di `wordnet3.json`. Si può ovviare al problema allocando uno spazio di memoria maggiore oppure splittando il file `.json`. Nel caso si preferisca utilizzare `cmd.exe` al posto di Cygwin, si avranno meno problemi legati alla dimensione dei file, ma sarà necessario riscrivere il file `import-solr-json.sh`, fornito con la distribuzione di conceptnet per l'importazione dei file, in un file `.bat`, in quanto `.sh` non è un formato riconosciuto da Windows. In alternativa si può utilizzare direttamente il comando `curl` specificando di volta in volta un singolo file da importare. Ovviamente, il tempo necessario per l'importazione varia a seconda della dimensione del file.

A.2.2 Cancellare l'indice

Per cancellare l'intero indice, è sufficiente utilizzare il browser digitando:

```
http://localhost:8983/solr/update?stream.body=<delete><query>*:*</query></delete
    >\&commit=true
```

A.3 Usare SolrJ per interrogare Solr

L'utilizzo della libreria SolrJ permette la connessione a Solr di un applicazione scritta in Java, sfruttando il protocollo HTTP. Tale libreria offre un'interfaccia java per aggiugere documenti, aggiornare o interrogare un indice Solr.

Si riporta nel seguito come effettuare alcune delle operazioni più comuni che vengono eseguite su un server Solr attraverso SolrJ. Si rimanda a (60) per una trattazione più esaustiva.

A.3.1 Creare un'istanza di un server Solr

La prima cosa da fare per interagire con un server Solr attraverso SolrJ è quella di creare un'istanza del server attraverso il codice java seguente:

```
SolrServer server = new CommonsHttpSolrServer("http://HOST:8983/solr/");
```

A.3.2 Impostare il formato dei dati

SolrJ permette di scaricare contenuti in formato XML (di default) o binario. Per modificare il formato è sufficiente utilizzare il seguente metodo:

```
server.setRequestWriter(new BinaryRequestWriter());
```

A.3.3 Leggere dati in Solr

Dopo aver creato un'istanza del server Solr e aver impostato il formato dei dati, è necessario costruire una query Solr:

```
SolrQuery query = new SolrQuery();
query.setQuery("IL TESTO DELLA MIA QUERY");
```

e quindi interrogare il server:

```
QueryResponse rsp = server.query(query);
```

ottenendo i risultati:

```
SolrDocumentList docs = rsp.getResults();
```

Di default vengono forniti i primi 10 risultati. Se si ha la necessità di più documenti è possibile utilizzare l'impostazione seguente:

```
query.setRow(NUMERO DI RISULTATI);
```

Bisogna però fare attenzione che tale numero non sia troppo elevato. Infatti, si rischia un *out of memory error*. Si consiglia pertanto di mantenere l'impostazione standard oppure ridurre il numero di campi di interesse nella risposta. La selezione dei campi può essere fatta così:

```
query.setFields("CAMPO1", "CAMPO2", "CAMPO3");
```

Tendendo presente che si può specificare un qualsiasi numero di campi, i quali però devono corrispondere effettivamente a campi dell'indice.

A.3.4 Come configurare Eclipse

Si riporta schematicamente come importare in Eclipse le librerie necessarie per utilizzare le classi messe a disposizione da SolrJ:

1. Scaricare da `http://lucene.apace.org//solr` l'ultima versione di Apache Solr ed estrarre il pacchetto ad esempio in `C:/Solr`.
2. Estrarre tutti i file .jar contenuti in `C:/solr/examples/webapps/solr.war`.
3. Aggiungere tutti i file .jar nel Java Build Path del progetto Eclipse.

Appendice B

Dati e tabelle

B.1 Lista di stopword

a	did	herself	not	the	we've
about	didn't	him	of	their	were
above	do	himself	off	theirs	weren't
after	does	his	on	them	what
again	doesn't	how	once	themselves	what's
against	doing	how's	only	then	when
all	don't	i	or	there	when's
am	down	i'd	other	there's	where
an	during	i'll	ought	these	where's
and	each	i'm	our	they	which
any	few	i've	ours	they'd	while
are	for	if	ourselves	they'll	who
aren't	from	in	out	they're	who's
as	further	into	over	they've	whom
at	had	is	own	this	why
be	hadn't	isn't	same	those	why's
because	has	it	shan't	through	with
been	hasn't	it's	she	to	won't
before	have	its	she'd	too	would
being	haven't	itself	she'll	under	wouldn't
below	having	let's	she's	until	you
between	he	me	should	up	you'd
both	he'd	more	shouldn't	very	you'll
but	he'll	most	so	was	you're
by	he's	mustn't	some	wasn't	you've
can't	her	my	such	we	your
cannot	here	myself	than	we'd	yours
could	here's	no	that	we'll	yourself
couldn't	hers	nor	that's	we're	yourselves

B.2 Risultati delle valutazioni da parte dei giudici

topic	corte			lunghe			lunghe e prolisse			corte e prolisse		
302	0	0	0	0	0	0	1	1	1	0	1	1
315	0	0	0	0	0	0	1	1	0	0	0	0
323	0	0	0	0	0	0	1	1	1	1	0	1
326	0	0	0	0	0	0	1	1	1	0	1	1
330	0	0	0	1	0	0	1	1	1	1	0	1
333	0	0	0	1	0	0	1	0	0	0	0	0
335	0	0	0	0	0	0	1	1	1	0	1	1
340	0	0	0	0	0	0	1	1	0	1	0	0
341	0	0	0	1	0	0	1	1	1	0	0	0
346	0	0	0	1	0	0	1	1	1	0	1	1
349	0	0	0	0	0	0	1	1	0	1	0	1
350	0	0	0	0	0	0	1	0	0	1	0	1
351	0	0	0	0	0	0	1	0	1	0	1	1
366	0	0	0	0	0	0	1	1	0	1	0	0
368	0	0	0	0	0	0	1	1	0	1	0	1
375	0	0	0	0	0	0	1	1	1	0	1	1
376	0	0	0	1	0	0	1	1	0	1	0	1
378	0	0	0	1	0	0	1	0	1	0	0	0
384	0	0	0	0	0	0	1	1	1	0	1	1
385	0	0	0	1	0	0	1	1	1	1	0	1
389	0	0	0	1	0	0	1	1	0	1	0	1
399	0	0	0	0	0	0	1	1	1	0	1	1
402	0	0	0	1	0	0	1	1	0	0	0	1
418	0	0	0	0	0	0	1	1	1	1	1	1
421	0	0	0	0	0	0	1	1	0	1	0	1
422	0	0	0	0	0	0	1	0	1	1	0	1
428	0	0	0	1	0	0	1	0	1	0	1	1
435	0	0	0	0	0	0	1	1	1	1	0	1
443	0	0	0	0	0	0	0	0	0	1	0	1
449	0	0	0	0	0	0	1	1	1	0	1	1

Tabella B.1: risultati delle valutazioni sulla prolissità effettuate dai giudici. Gli "1" indicano che la query è stata giudicata prolissa, gli "0" non prolissa.

I risultati delle valutazioni umane effettuate dai giudici vengono riportate in tabella B.1. Gli "0" indicano che la query è stata giudicata non prolissa, gli "1" viceversa indicano che è stata giudicata prolissa. Le interrogazioni sono state suddivise al fine della valutazione in tre sottoinsiemi costruiti in modo pseudo-casuale, in modo da

contenere tutti i tipi di interrogazione, in ordine sparso. Poiché ciascun sottoinsieme ha richiesto tre valutazioni, sono stati coinvolti complessivamente 9 giudici.

I dati di tali valutazioni vengono riassunti nella seguente matrice di confusione:

	$c \wedge \bar{p}$	$l \wedge \bar{p}$	$l \wedge p$	$c \wedge p$	
$c \wedge \bar{p}$	90	0	0	0	90
$l \wedge \bar{p}$	0	80	10	0	90
$l \wedge p$	0	20	70	0	90
$c \wedge p$	40	0	0	50	90
	130	100	80	50	360

Tabella B.2: Matrice di confusione calcolata sulla base delle valutazioni effettuate dai giudici.

Può essere riassunta in:

	p	$\bar{p}$	
c	50	130	180
l	80	100	180
	130	230	360

Tabella B.3: Matrice di confusione riassuntiva.

Sulla base di tali risultati è possibile calcolare le seguenti probabilità condizionate:

$$P(p|c) = \frac{50}{180} \approx 27,78\%$$

$$P(p|l) = \frac{80}{180} \approx 44,44\%$$

$$P(\bar{p}|c) = \frac{130}{180} \approx 72,22\%$$

$$P(\bar{p}|l) = \frac{100}{180} \approx 55,56\%$$

B.3 Dati in input per Weka

```
id, lunghezza, sinonimi, concetti,
    ripetizioni, classe
3021, 3, 1.0, 3, 0.0, NO
3022, 5, 2.0, 5, 0.0, NO
3023, 34, 0.0, 26, 13.0, YES
3024, 11, 3.0, 1, 5.0, YES
3151, 3, 0.0, 3, 0.0, NO
3152, 7, 2.0, 7, 0.0, NO
3153, 54, 1.0, 44, 18.0, YES
3154, 4, 0.0, 4, 0.0, YES
3231, 3, 0.0, 3, 0.0, NO
3232, 6, 0.0, 6, 0.0, NO
3233, 60, 2.0, 58, 4.0, YES
3234, 4, 0.0, 4, 0.0, YES
3261, 2, 0.0, 2, 0.0, NO
3262, 7, 0.0, 7, 0.0, NO
3263, 20, 2.0, 2, 6.0, YES
3264, 4, 0.0, 4, 0.0, YES
3301, 3, 0.0, 3, 0.0, NO
3302, 13, 0.0, 13, 0.0, NO
3303, 93, 5.0, 1, 28.0, YES
3304, 7, 0.0, 2, 4.0, YES
3331, 3, 0.0, 3, 0.0, NO
3332, 15, 1.0, 1, 2.0, NO
3333, 66, 3.0, 1, 28.0, YES
3334, 7, 0.0, 7, 0.0, YES
3351, 3, 0.0, 3, 0.0, NO
3352, 14, 0.0, 12, 4.0, NO
3353, 67, 1.0, 56, 18.0, YES
3354, 4, 0.0, 1, 2.0, YES
3401, 3, 0.0, 3, 0.0, NO
3402, 14, 1.0, 14, 0.0, NO
3403, 98, 4.0, 89, 17.0, YES
3404, 4, 1.0, 4, 0.0, YES
3411, 2, 0.0, 2, 0.0, NO
3412, 16, 0.0, 16, 0.0, NO
3413, 99, 4.0, 68, 48.0, YES
3414, 4, 0.0, 4, 0.0, YES
3461, 2, 0.0, 2, 0.0, NO
3462, 25, 1.0, 1, 9.0, NO
3463, 100, 2.0, 1, 32.0, YES
3464, 3, 0.0, 3, 0.0, YES
3491, 1, 0.0, 1, 0.0, NO
3492, 12, 0.0, 12, 0.0, NO
3493, 48, 2.0, 1, 12.0, YES
3494, 4, 0.0, 4, 0.0, YES
3501, 3, 0.0, 3, 0.0, NO
3502, 8, 0.0, 8, 0.0, NO
3503, 48, 1.0, 45, 6.0, YES
3504, 6, 0.0, 6, 0.0, YES
3511, 3, 1.0, 3, 0.0, NO
3512, 9, 1.0, 9, 0.0, NO
3513, 32, 0.0, 2, 4.0, YES
3514, 6, 2.0, 6, 0.0, YES
3661, 3, 0.0, 3, 0.0, NO
3662, 5, 0.0, 5, 0.0, NO
3663, 36, 7.0, 34, 4.0, YES
3664, 5, 0.0, 5, 0.0, YES
3681, 2, 0.0, 2, 0.0, NO
3682, 5, 0.0, 5, 0.0, NO
3683, 83, 11.0, 71, 20.0, YES
3684, 9, 1.0, 9, 0.0, YES
3751, 2, 0.0, 2, 0.0, NO
3752, 6, 0.0, 6, 0.0, NO
3753, 64, 2.0, 59, 8.0, YES
3754, 3, 0.0, 3, 0.0, YES
3761, 2, 0.0, 2, 0.0, NO
3762, 8, 0.0, 7, 2.0, NO
3763, 34, 1.0, 1, 8.0, YES
3764, 7, 0.0, 5, 4.0, YES
3781, 2, 0.0, 2, 0.0, NO
3782, 8, 0.0, 8, 0.0, NO
3783, 63, 5.0, 5, 18.0, YES
3784, 5, 1.0, 5, 0.0, YES
3841, 3, 1.0, 3, 0.0, NO
3842, 9, 1.0, 9, 0.0, NO
3843, 33, 2.0, 26, 11.0, YES
3844, 5, 1.0, 5, 0.0, YES
3851, 3, 0.0, 3, 0.0, NO
3852, 13, 1.0, 13, 0.0, NO
3853, 150, 10.0, 9, 76.0, YES
3854, 6, 0.0, 1, 2.0, YES
3891, 3, 0.0, 3, 0.0, NO
3892, 16, 0.0, 15, 2.0, NO
3893, 34, 1.0, 31, 6.0, YES
3894, 6, 0.0, 5, 2.0, YES
3991, 2, 0.0, 2, 0.0, NO
3992, 7, 0.0, 7, 0.0, NO
3993, 112, 1.0, 4, 22.0, YES
3994, 3, 0.0, 3, 0.0, YES
4021, 2, 0.0, 2, 0.0, NO
4022, 14, 0.0, 14, 0.0, NO
4023, 171, 11.0, 130, 62.0, YES
4024, 5, 0.0, 5, 0.0, YES
4181, 2, 0.0, 2, 0.0, NO
4182, 5, 0.0, 5, 0.0, NO
4183, 71, 7.0, 1, 6.0, YES
4184, 3, 0.0, 3, 0.0, YES
```

```
4211, 3, 0.0, 3, 0.0, NO
4212, 8, 0.0, 1, 2.0, NO
4213, 45, 2.0, 42, 6.0, YES
4214, 5, 0.0, 1, 2.0, YES
4221, 3, 0.0, 3, 0.0, NO
4222, 4, 0.0, 4, 0.0, NO
4223, 18, 0.0, 17, 2.0, YES
4224, 4, 0.0, 4, 0.0, YES
4281, 3, 0.0, 3, 0.0, NO
4282, 7, 0.0, 7, 0.0, NO
4283, 44, 4.0, 40, 7.0, YES
4284, 7, 0.0, 7, 0.0, YES
4351, 3, 0.0, 3, 0.0, NO
4352, 8, 0.0, 8, 0.0, NO
4353, 132, 7.0, 2, 60.0, YES
4354, 5, 1.0, 5, 0.0, YES
4431, 4, 0.0, 4, 0.0, NO
4432, 9, 0.0, 9, 0.0, NO
4433, 47, 9.0, 1, 11.0, YES
4434, 7, 0.0, 1, 2.0, YES
4491, 2, 0.0, 2, 0.0, NO
4492, 8, 3.0, 8, 0.0, NO
4493, 51, 1.0, 1, 31.0, YES
4494, 8, 1.0, 8, 0.0, YES
```

B.4 I principali metodi implementati in Java

B.4.1 Metodo per l'estrazione dei sinonimi

Tale metodo riceve in input:

- una stringa `word`, che contiene il termine di cui si vogliono ricercare i sinonimi;
- un oggetto `server` di tipo `CommonsHttpSolrServer`, che contiene l'istanza del server Solr;

e restituisce una stringa contenente l'elenco dei sinonimi. Per produrre tale stringa le operazioni che vengono eseguite sono:

- cercare in Solr il termine dato in input, ossia il parametro `word` (riga 2-12);
- estrarre solo i risultati che contengono la relazione `/r/Synonym` (riga 13-36);
- esaminare il campo `Surfacetext` (riga 37-51); poiché il termine dato in input è uno dei due concetti contenuti in `Surfacetext`, si salva l'altro nella stringa risultato (riga 37-64).

Si riporta nel seguito il codice Java corrispondente:

```
1  public static String getSynonym(String word, CommonsHttpSolrServer server)
       throws SolrServerException {
2          //creo la query e imposto il numero di risultati
3          SolrQuery query = new SolrQuery();
4          query.setQuery(word);
5          query.setStart(0);
6          query.setRows(50);
7          query.setFields("surfaceText", "rel", "sources");
8
9          //processo la query
10         QueryRequest qryReq = new QueryRequest(query);
11         QueryResponse response = qryReq.process(server);
12
13         //leggo i risultati
14         boolean find = false;
15         Set<String> result = new HashSet<String>();
16         result.clear();
17         int i = (int) response.getResults().getNumFound();
18         if(i>50){
19                 i = 50;
20         }
21         for(int t= 0; t<i; t++){
22                 SolrDocument doc = response.getResults().get(t);
```

```
23                  String rel = doc.getFieldValue("rel").toString();
24                  String source = doc.getFieldValue("sources").toString();
25                  if(rel.equals("/r/Synonym") && source.contains("/s/web/en.
                        wiktionary.org")){ //voglio solo i sinonimi inglesi
26                      //devo controllare che uno dei due concetti sia la mia
                            parola
27                      //(e non delle estensioni...)
28                      Object text = response.getResults().get(t).getFieldValue
                            ("surfaceText");
29                      String str = text.toString();
30
31              //ripulisco il testo
32                      str = str.replace("Synonym", "");
33                      str = str.replaceAll("[^a-zA-Z\\s-]", "");
34
35                      String[] tmpList = str.split("  ");
36
37                      if(tmpList.length==2){
38                              for(int x=0; x<tmpList.length; x++){
39                                      if(tmpList[x]!= null && !tmpList[x].
                                            equals("\\s") && tmpList[x].equals(
                                            word)){
40                                          find = true;
41                                          if(x == 0){
42                                                  result.add(tmpList[1]);
43                                          }
44                                          else{
45                                                  result.add(tmpList[0]);
46                                          }
47                                      }
48                              }
49                      }
50                  }
51          }
52          //se ho trovato dei sinonimi li salvo in una stringa
53          String synonyms = "";
54          if(find){
55                  Object[] synonymList = result.toArray();
56                  for(Object tmp : synonymList){
57                          synonyms += tmp.toString() + " ";
58                  }
59          }
60          else{
61                  //System.out.println("La parola " + word + " non ha sinonimi
                        conosciuti");
62          }
63          return synonyms.trim();
64  }
```

Listing B.1: Metodo che restituisce i sinonimi di un termine dato in input.

B.4.2 Metodi per il *topic gisting*

Il metodo ricorsivo implementato per il *topic gisting* riceve in input:

- un array di stringhe `text` che contiene i termini del testo su cui si vuole effettuare il *topic gisting*. È da notare che nel caso in cui si voglia utilizzare la versione riportata in questo documento come "tag+concetti" oppure "tag2+concetti", è necessario salvare nell'array solo i termini selezionati utilizzando un *tagger* per individuare le parti del discorso (per non appesantire la trattazione non si riporta nel seguito tale codice);
- un oggetto di tipo `CommonsHttpSolrServer` che corrisponde al server Solr in cui ricercare i dati di interesse;
- un intero `min` che contiene il primo indice d'interesse dell'array;
- un intero `max` che contiene l'ultimo indice d'interesse dell'array;

e restituisce una stringa contenente l'elenco dei termini che costituiscono il *topic gist*, ossia il succo del discorso.

Come spiegato brevemente nel par. 3.3.4, facendo riferimento al codice riportato in *Listing* B.2, le operazioni che vengono svolte da questo metodo sono:

Passo base (riga 5-28): Se il testo è composto da un unico termine viene restituito l'insieme di concetti collegati a tale termine (nel caso in cui l'insieme sia vuoto si restituisce il termine stesso); per far ciò si utilizza il metodo `getContext()` che verrà presentato in maggior dettaglio nel seguito. Se invece il testo è composto da due termini, per ciascuno di essi viene restituito l'insieme dei concetti collegati e si effettua l'intersezione, attraverso il metodo `getOverlaps()`; nel caso in cui l'intersezione dovesse risultare vuota, viene restituito il testo dato in input.

Passo ricorsivo (riga 31-74): Se il testo è composto da più termini, viene suddiviso in due parti e su ognuna si applica ricorsivamente l'algoritmo. Viene quindi effettuata l'intersezione dei risultati dei due sottoproblemi.

```
1  public static String mergeGist(String[] text, CommonsHttpSolrServer server, int
       min, int max) throws ParseException{
2          HashMap<String, Integer> intersection = null;
3          String result = "";
```

```
4
5       //caso base
6           int size = max - min;
7           if(size == 0 || text.length==1){
8                   text[min]= text[min].replaceAll("[^a-zA-Z0-9\\s-]", " ");
9                   return getContext(text[min], server, 50);
10          }
11          if(size == 1 || text.length==2){
12          String tmp = text[min] + " " + text[max];
13                  String c1 = getContext(text[min], server, 50);
14                  String c2 = getContext(text[max], server, 50);
15                  if(!(c1.isEmpty()&&c2.isEmpty())){
16                                  intersection = getOverlaps(c1, c2);
17                  }
18          //se l'intersezione è vuota restituisco il testo di partenza
19                  if(intersection.isEmpty()){
20                          result = tmp;
21                  }
22                  else{
23                          result = intersection.keySet().toString();
24                  }
25                  result = result.replaceAll("\\[", "");
26                  result = result.replaceAll("\\]", "");
27                  return result;
28          }//fine caso base
29
30          //calcolo l'indice intermedio
31          int middle = (min + max)/2;
32
33          //salvo la prima metà dei concetti di partenza
34          //che utilizzerò nel caso l'intersezione sia vuota
35          String[] part1 = new String[middle-min+1];
36          int j = 0;
37          for(int i=min; i<=middle; i++){
38                  part1[j] = text[i];
39                  j++;
40          }
41      //chiamo ricorsivamente il metodo
42          String context1 = mergeGist(text, server, min, middle);
43
44          //faccio lo stesso per la seconda metà
45          String[] part2 = new String[max-(middle+1)+1];
46          j = 0;
47          for(int i=middle+1; i<=max; i++){
48                  part2[j] = text[i];
49                  j++;
50          }
51      //chiamo ricorsivamente il metodo
52          String context2 = mergeGist(text, server, middle+1, max);
53
54      //se non sono vuoti fai l'intersezione
55          if(!(context1.isEmpty()&&context2.isEmpty())){
```

```
56                  intersection = getOverlaps(context1, context2);
57          }
58          //se l'intersezione è vuota utilizzo i concetti da cui
59          //sono partita (non l'espansione...)
60          if(intersection.isEmpty()){
61                  System.out.println("concateno i risultati :");
62                  result = context1 + " " + context2;
63                  System.out.println(result);
64          }
65          else{
66                  result = intersection.keySet().toString();
67          }
68
69          //faccio un po' di pulizia...
70          result = result.replaceFirst("\\[", "");
71          result = result.replaceFirst("\\]", "");
72
73          return result;
74  }
```

Listing B.2: Metodo ricorsivo per il *topic gisting*.

Il metodo `getContext()`, che restituisce l'insieme di concetti collegati, riceve in input i seguenti parametri:

- una stringa `s` che contiene il termine di cui si vogliono conoscere i concetti collegati;
- un oggetto di tipo `CommonsHttpSolrServer` che corrisponde al server Solr in cui verranno effettuate le ricerche;
- un intero `i` che corrisponde alla massima cardinalità dell'insieme di concetti collegati;

e restituisce un elenco di concetti collegati in formato di stringa, con un concetto per riga.

Si riporta in *Listing* B.3 il codice del metodo che restituisce i concetti collegati di un termine dato in input. Le operazioni che vengono svolte da tale metodo sono:

- creare una query in Solr corrispondente al termine dato in input e processarla (riga 2-13);
- leggere il contenuto della risposta, selezionando solo le relazioni di tipo `/r/IsA` e `/r/HasProperty` (riga 16-38);

- salvare i concetti estratti in una stringa risultato (riga 39-45);

```
1  public static String getContext(String s, CommonsHttpSolrServer server, int i){
2          try{
3          //creo la query e imposto il numero di risultati
4          SolrQuery query = new SolrQuery();
5          query.setQuery(s);
6          query.setStart(0);
7          query.setRows(i);
8          //imposto i campi di interesse
9          query.setFields("text", "rel");
10
11         //processo la query
12         QueryRequest qryReq = new QueryRequest(query);
13         QueryResponse response = qryReq.process(server);
14
15         //salvo i risultati in una stringa d'output, eliminando i doppioni
16         String r = "";
17         Set<String> set = new LinkedHashSet<String>();
18         set.clear();
19         long maxNum = response.getResults().getNumFound();
20         if(maxNum < i){
21                 i = (int) maxNum;
22                 System.out.println("Sono presenti meno concetti collegati...");
23         }
24         for(int t= 0; t<i; t++){
25                 String rel = response.getResults().get(t).getFieldValue("rel").
                       toString();
26                 if(rel.equals("/r/IsA") || rel.equals("/r/HasProperty")){
27                         Object text = response.getResults().get(t).getFieldValue
                               ("text");
28                         String str = text.toString();
29                         String delims = "[\\[,\\]]+";
30                         String[] list = str.split(delims);
31                         for(int x=0; x<list.length; x++){
32                                 list[x] = list[x].trim(); //rimuovo gli spazi
                                       all'inizio delle parole
33                                 System.out.println(">"+list[x]);
34                         }
35                         List<String> l = Arrays.asList(list);
36                         set.addAll(l); //rimuovo i doppioni mantenendo l'ordine
37                 }
38         }
39         String[] result = new String[set.size()];
40         set.toArray(result);
41         for(String tmp : result){
42                 r += tmp + "\n";
43         }
44         return r;
45     }
46         catch(Error e){
47                 return "";
```

```
48          }
49          catch(Exception SolrServerException){
50                  System.out.println("Impossibile trovare il concetto");
51                  return "";
52          }
53  }
```

Listing B.3: Metodo per l'estrazione dei concetti collegati.

Il metodo `getOverlaps()`, che effettua l'intersezione tra stringhe, restituendo le parole in comune, riceve in input:

- due stringhe: `string0` e `string1`;

e restituisce una struttura dati di tipo `HashMap<String, Integer>` che contiene le parole in comune tra le due stringhe e il numero di volte che ciascuna parola compare in entrambe le stringhe.

Tale metodo è una reimplementazione in Java di una funzione omonima scritta in Perl proposta da Jason Michelizzi, Ted Pedersen, Siddharth Patwardhan, Satanjeev Banerjee e Ying Liu (44). Nel codice riportato in *Listing* B.4 si sono mantenuti i commenti originali (in inglese).

```
1   public static HashMap<String, Integer> getOverlaps(String string0, String
        string1) {
2
3       // Results are a HashMap with keys that are the overlapping strings and
4       // values that are the frequency counts of those overlaps
5       HashMap<String, Integer> overlapsHash = new HashMap<String, Integer>();
6       int matchStartIndex = 0;
7       int currIndex = -1;
8
9       // Strings are compared based on word overlaps, so create arrays of
10      // words
11      String[] S = null, W = null;
12
13      // assign the shortest string to variable S
14      if (string0.length() > string1.length()) {
15              W = string0.trim().split("\\s+");
16              S = string1.trim().split("\\s+");
17      }
18      else {
19              S = string0.trim().split("\\s+");
20              W = string1.trim().split("\\s+");
21      }
22      int[] overlapsLengths = new int[W.length+S.length];
23
24      while (currIndex < S.length - 1) {
```

```
25              currIndex++;
26              if (contains(W, Arrays.copyOfRange(S, matchStartIndex, currIndex +
                    1), false)) {
27              continue;
28              }
29          else {
30              overlapsLengths[matchStartIndex] = currIndex - matchStartIndex;
31                  if (overlapsLengths[matchStartIndex] > 0){
32                  currIndex--;
33              }
34                      matchStartIndex++;
35              }
36          }
37          for (int i = matchStartIndex; i <= currIndex; i++) {
38          overlapsLengths[i] = currIndex - i + 1;
39          }
40      int longestOverlap = 0;
41      for (int i = 0; i < overlapsLengths.length; i++) {
42          if (overlapsLengths[i] > longestOverlap){
43              longestOverlap = overlapsLengths[i];
44          }
45      }
46
47      while (longestOverlap > 0) {
48          for (int i = 0; i <= overlapsLengths.length - 1; i++) {
49                  if (overlapsLengths[i] < longestOverlap){
50                     continue;
51              }
52                  int stringEnd = i + longestOverlap;
53                  if (contains(W, Arrays.copyOfRange(S, i, stringEnd), true)) {
54                    String temp = new String(S[i]);
55                    for (int j = i + 1; j < stringEnd; j++) {
56                      temp += " " + S[j];
57                    }
58
59                    if (overlapsHash.containsKey(temp)) {
60                      overlapsHash.put(temp.toString(), overlapsHash.get(temp) +
                            1);
61                    }
62                else {
63                      overlapsHash.put(temp, 1);
64                    }
65
66                    // adjust overlap lengths forward
67                    for (int j = i; j < i + longestOverlap; j++) {
68                      overlapsLengths[j] = 0;
69                    }
70
71                    // adjust overlap lengths backward
72                    for (int j = i - 1; j >= 0; j--) {
73                      if (overlapsLengths[j] <= i - j)
74                        break;
```

```
75                      overlapsLengths[j] = i - j;
76                    }
77                  }
78                else {
79                    int k = longestOverlap - 1;
80                    while (k > 0) {
81                      stringEnd = i + k - 1;
82                      if (contains(W, Arrays.copyOfRange(S, i, stringEnd), false))
83                        break;
84                      k--;
85                    }
86                    overlapsLengths[i] = k;
87                  }
88                }
89                longestOverlap = Collections.max(Arrays.asList(longestOverlap)) -
                      1;
90          }
91      return overlapsHash;
92  }
```

Listing B.4: Metodo per effettuare l'intersezione tra le parole di due stringhe.

Bibliografia

[1] Lemur project, 2012. `www.lemurproject.org/`. 49

[2] Banerjee, S., and Pedersen, T. Extended gloss overlaps as a measure of semantic relatedness. In *Proceedings of the 18th international joint conference on Artificial intelligence* (San Francisco, CA, USA, 2003), IJCAI'03, Morgan Kaufmann Publishers Inc., pp. 805–810. 3

[3] Bendersky, M., and Croft, W. B. Discovering key concepts in verbose queries. In *Proceedings of the 31st annual international ACM SIGIR conference on Research and development in information retrieval* (New York, NY, USA, 2008), SIGIR '08, ACM, pp. 491–498. 1, 3, 8, 11, 44, 49

[4] Bendersky, M., and Croft, W. B. Analysis of long queries in a large scale search log. In *Proceedings of the 2009 workshop on Web Search Click Data* (New York, NY, USA, 2009), WSCD '09, ACM, pp. 8–14. `http://videolectures.net/wscd09_bendersky_alqlssl/`. 9, 39

[5] Bendersky, M., Metzler, D., and Croft, W. B. Parameterized concept weighting in verbose queries. In *Proceedings of the 34th international ACM SIGIR conference on Research and development in Information Retrieval* (New York, NY, USA, 2011), SIGIR '11, ACM, pp. 605–614. 24

[6] Berger, A., and Lafferty, J. Information retrieval as statistical translation. In *Proceedings of the 22nd annual international ACM SIGIR conference on Research and development in information retrieval* (New York, NY, USA, 1999), SIGIR '99, ACM, pp. 222–229. 17, 29

[7] Broder, A. A taxonomy of web search. *SIGIR Forum 36*, 2 (Sept. 2002), 3–10. vii, 2, 32, 44

[8] Conceptnet 4, 2012. `http://csc.media.mit.edu/docs/conceptnet/webapi_client.html`. 35

[9] Croft, B., Metzler, D., and Strohman, T. *Search Engines: Information Retrieval in Practice*, 1st ed. Addison-Wesley Publishing Company, USA, 2009. 14, 16, 17

[10] Croft, W. B. Unsolved problems in search: (and how we approach them). In *Proceedings of the 17th ACM conference on Information and knowledge management* (New York, NY, USA, 2008), CIKM '08, ACM, pp. 1001–1001. `http://videolectures.net/cikm08_croft_upis/`. 17

[11] Croft, W. B. Query evolution. In *Proceedings of the 31th European Conference on IR Research on Advances in Information Retrieval* (Berlin, Heidelberg, 2009), ECIR '09, Springer-Verlag, pp. 1–1. 8, 9

[12] Dbpedia, 2012. `dpedia.org/About`. 35

[13] de Marneffe, M.-C., and Manning, C. D. The stanford typed dependencies representation. In *Coling 2008: Proceedings of the workshop on Cross-Framework and Cross-Domain Parser Evaluation* (Stroudsburg, PA, USA, 2008), CrossParser '08, Association for Computational Linguistics, pp. 1–8. 21

[14] Eagle, N., Singh, P., and Pentland, A. Common sense conversations: understanding casual conversation using a common sense database. In *Proceedings of the Artificial Intelligence, Information Access, and Mobile Computing Workshop* (2003). 38

[15] Echihabi, A., and Marcu, D. A noisy-channel approach to question answering. In *Proceedings of the 41st Annual Meeting on Association for Computational Linguistics - Volume 1* (Stroudsburg, PA, USA, 2003), ACL '03, Association for Computational Linguistics, pp. 16–23. 30

[16] Edmundson, H. P. New methods in automatic extracting. *J. ACM 16*, 2 (Apr. 1969), 264–285. 19

[17] Fundel, K., Küffner, R., and Zimmer, R. Relex—relation extraction using dependency parse trees. *Bioinformatics 23*, 3 (Jan. 2007), 365–371. 20

[18] Guess, A. Bing launches adaptive search, Settembre 2011. `http://semanticweb.com/bing-launches-adaptive-search_b23175`. 11

[19] Guo, J., Xu, G., Li, H., and Cheng, X. A unified and discriminative model for query refinement. In *In SIGIR '08* (2008), pp. 379–386. 30

[20] Hauff, C., Hiemstra, D., and de Jong, F. A survey of pre-retrieval query performance predictors. *CIKM '08, Napa Valley, California, USA* (2008). 40

[21] He, B., and Ounis, J. Inferring query performance using pre-retrieval predictors. *The Eleventh Symposium on String Processing and Infroamtion Retrieval* (2004). 40

[22] Hiemstra, D., and de Jong, F. Disambiguation strategies for cross-language information retrieval. In *Proceedings of the Third European Conference on Research and Advanced Technology for Digital Libraries, ECDL '99* (1999). 27

[23] Hoenkamp, E., Bruza, P., Song, D., and Huang, Q. An effective approach to verbose queries using a limited dependencies language model. In *Proceedings of the 2nd International Conference on Theory of Information Retrieval: Advances in Information Retrieval Theory* (Berlin, Heidelberg, 2009), ICTIR '09, Springer-Verlag, pp. 116–127. 3, 28

[24] Hovy, E., and Lin, C.-Y. Automated text summarization and the summarist system. In *Proceedings of a workshop on held at Baltimore, Maryland: October 13-15, 1998* (Stroudsburg, PA, USA, 1998), TIPSTER '98, Association for Computational Linguistics, pp. 197–214. 38

[25] Jurafsky, D., and Martin, J. H. *Speech and Language Processing*. Prentice Hall, 2000. 19

[26] Kuc, R. *Apache Solr 3.1 Cookbook*. Packt Publishing, 2011. 61

[27] Kumaran, G., and Carvalho, V. R. Reducing long queries using query quality predictors. In *Proceedings of the 32nd international ACM SIGIR conference on*

Research and development in information retrieval (New York, NY, USA, 2009), SIGIR '09, ACM, pp. 564–571. 1, 11

[28] Lafferty, J. Conditional random fields: Probabilistic models for segmenting and labeling sequence data. Morgan Kaufmann, pp. 282–289. 23

[29] Lau, T., and Horvitz, E. Patterns of search: Analyzing and modeling web query refinement. In *Proceedings of the Seventh International Conference on User Modeling, Banff, Canada* (1998), Springer Wien, pp. 119–128. 12

[30] Lavrenko, V. *A Generative Theory of Relevance*. Springer Publishing Company, Incorporated, 2010. 27

[31] Lavrenko, V., and Croft, W. B. *Relevance Models in Information Retrieval*. Kluwer Academic Publishers, 2003, ch. 2. 25, 26, 27

[32] Lease, M. Improved markov random field model for supporting verbose queries. *SIGIR* (2009). 29

[33] Lehnert, W. G. Computational models of natural language processing. Elsevier North-Holland, Inc., New York, NY, USA, 1984, ch. Narrative complexity based on summarization algorithms, pp. 247–259. 19

[34] Lesk, M. Automatic sense disambiguation using machine readable dictionaries: how to tell a pine cone from an ice cream cone. In *Proceedings of the 5th annual international conference on Systems documentation* (New York, NY, USA, 1986), SIGDOC '86, ACM, pp. 24–26. 3

[35] Littman, M., Dumais, S., and Landauer, T. Automatic cross-language retrieval using latent semantic indexing. In *Cross-Language Retrieval* (1998). 15

[36] Apache lucene. `http://lucene.apache.org/`. 33

[37] Luhn, H. P. The automatic creation of literature abstracts. *IBM Journal of Research and Development 2*, 2 (Apr. 1959), 159–165. 19

[38] Markoff, J. A software secretary that takes charge, Dicembre 2008. `http://www.nytimes.com/2008/12/14/business/14stream.html`. 12

[39] Mauldin, M. L. Retreival performance in ferret: A conceptual information retrieval system. In *Proceedings of the 14th International Conference on Research and Development in Information Retrieval, Chicago* (1991). 19

[40] McKeown, K. R., and Radev, D. R. Generating summaries of multiple news articles. In *Proceedings,18th Annual International ACM SIGIR Conference on Research and Development in Information Retrieval* (1995). 19

[41] Melucci, M. *Dispensa: Lezioncine sui Motori di Ricerca.* Teoria e architetture per Information Retrieval e Machine Learning, 2011. 1, 24, 26, 48

[42] Metzler, D., and Croft, W. B. A markov random field model for term dependencies. In *Proceedings of the 28th annual international ACM SIGIR conference on Research and development in information retrieval* (New York, NY, USA, 2005), SIGIR '05, ACM, pp. 472–479. 28

[43] Metzler, D., and Croft, W. B. Latent concept expansion using markov random fields. In *Proceedings of the 30th annual international ACM SIGIR conference on Research and development in information retrieval* (New York, NY, USA, 2007), SIGIR '07, ACM, pp. 311–318. 29

[44] Michelizzi, J., Pedersen, T., Patwardhan, S., Banerjee, S., and Liu, Y. Overlapfinder - find overlapping words in strings, 2010. `http://cpan.uwinnipeg.ca/htdocs/Text-Similarity/Text/OverlapFinder.pm.html`. 78

[45] NIST, T. Text retrieval conference (trec) data - english relevance judgements, 2006. `http://trec.nist.gov/data/reljudge_eng.html`. 44

[46] Park, J. H., and Croft, W. B. Query term ranking based on dependency parsing of verbose queries. *Proceeding of the 33rd international ACM SICIR conference on Research and development in information retrieval* (2010), 829–830. 3, 19, 20

[47] Phan, N., Bailey, P., and Wilkinson, R. Understanding the relationship of information need specificity to search query length. In *Proceedings of the 30th annual international ACM SIGIR conference on Research and development in information retrieval* (New York, NY, USA, 2007), SIGIR '07, ACM, pp. 709–710. 12

[48] Ponte, J. M., and Croft, W. B. A language modeling approach to information retrieval. In *Proceedings of the 21st annual international ACM SIGIR conference on Research and development in information retrieval* (New York, NY, USA, 1998), SIGIR '98, ACM, pp. 275–281. 24, 30

[49] Riezler, S., Vasserman, A., Tsochantaridis, I., Mittal, V., and Liu, Y. Statistical machine translation for query expansion in answer retrieval. In *Proceedings of the 45th Annual Meeting of the Association for Computational Linguistics (ACL'07)* (Prague, Czech Republic, 2007). 30

[50] Robertson. The probability ranking principle in ir. *Journal of Documentation* (1977), 33:294–304. Reprinted in (Sparck Jones and Willett, 1997). 26

[51] Robertson, and Jones, S. Relevance weighting of search terms. *Journal of the American Society for Information Science* (1976), 27:129–146. Reprinted in (Willett, 1988). 25

[52] Robertson, S. E., and Walker, S. Some simple effective approximations to the 2-poisson model for probabilistic weighted retrieval. *Proc. of SIGIR* (1994), 232–241. 24

[53] Salton, G., and Buckley, C. Improving retrieval performance by relevance feedback. Tech. rep., Ithaca, NY, USA, 1988. 60

[54] Salton, G., Wong, A., and Yang, C. S. A vector space model for automatic indexing. Tech. rep., Ithaca, NY, USA, 1974. 33

[55] Solr tutorial. `http://lucene.apache.org/solr/tutorial.html`. 61

[56] Speer, R., and Havasi, C. Conceptnet 5, 2012. `http://conceptnet5.media.mit.edu/`. 35

[57] Speer, R., and Havasi, C. Representing general relational knowledge in conceptnet 5. In *Proceedings of LREC* (2012). 34

[58] Tan, P.-N., Steinbach, M., and Kumar, V. *Introduction to Data Mining.* Pearson International Edition, 2006. 42

[59] Tomlinson, S. Lexical and algorithmic stemming compared for 9 european languages with hummingbird searchserver at clef 2003. In *Cross-Language Evaluation Forum* (2003). 15

[60] Solrj. `http://wiki.apache.org/solr/Solrj`. 63

[61] Vasilescu, F., Langlais, P., and Lapalme, G. Evaluating variants of the lesk approach for disambiguating words. In *Proceedings of Language Resources and Evaluation (LREC 2004)* (Lisbonne, Portugal, 2004), pp. 633–636. 41

[62] Wang, X., and Zhai, C. Mining term association patterns from search logs for effective query reformulation. In *Proceedings of the 17th ACM conference on Information and knowledge management* (New York, NY, USA, 2008), CIKM '08, ACM, pp. 479–488. 30

[63] Wiktionary. `http://www.wiktionary.org/`. 35

[64] About wordnet, 2012. `http://wordnet.princeton.edu/`. 20, 35

[65] Xue, X., Huston, S., and Croft, W. B. Improving verbose queries using subset distribution. In *Proceedings of the 19th ACM international conference on Information and knowledge management* (New York, NY, USA, 2010), CIKM '10, ACM, pp. 1059–1068. 23

[66] Xue, X., Jeon, J., and Croft, W. B. Retrieval models for question and answer archives. In *Proceedings of the 31st annual international ACM SIGIR conference on Research and development in information retrieval* (New York, NY, USA, 2008), SIGIR '08, ACM, pp. 475–482. 11, 30

Ringraziamenti

Vorrei ringraziare il Prof. Massimo Melucci, relatore di questa tesi, per la costante cortesia dimostratami e le preziose indicazioni che mi sono state date durante questi mesi. Desidero, inoltre, ringraziare il Dr. Emanuele di Buccio per la sua disponibilità e gli utili suggerimenti. Intendo poi ringraziare il NIST per i documenti della collezione TREC Robust 2004.

Hanno collaborato in qualità di giudici a questa tesi: Daniele Barilaro, Alessando Di Pieri, Andrea Marcato, Giulia Moro, Lorena Moro, Luca Pellegrini, Rossella Petrucci, Lucia Petterle; a loro va un grazie particolare per il tempo che hanno dedicato a questo incarico e all'attenzione che vi hanno riposto.

Infine, ringrazio con affetto la mia famiglia, che mi ha sempre sostenuta e incoraggiata.