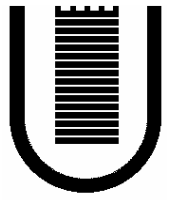
Università degli Studi di Roma "Tor Vergata"



Facoltà di Ingegneria

Corso di Laurea in Ingegneria Informatica

Query expansion mediante WordNet in un sistema di IR di tipo VSM

Docente

Prof. Roberto Basili

Candidati

Andrea Benedetti
Marco Brunamonti
Marco Dell'Olio

Anno Accademico 2005/2006

Sommario

Introduzione

Nell'ambito del corso di Basi di Dati Distribuite è stato sviluppato un progetto riguardante la valutazione delle prestazioni di un modello di espansione delle queries basato su WordNet. A tale scopo è stato necessario dapprima sviluppare un sistema di IR sul quale applicare tale modello e dal quale ricavare gli opportuni indici di prestazione.

Il sistema è stato implementato impiegando, in un framework Java, Lucene come motore di IR, il Cranfield Data Set come collezione standard di documenti e di queries e la libreria JWordNet come interfaccia tra WordNet e il sistema di IR.

Sostanzialmente il sistema di IR realizzato compie le seguenti operazioni:

- effettua il parsing della collezione di documenti;
- crea un indice e vi aggiunge i documenti;
- effettua il parsing della collezione di queries;
- sottomette le queries al sistema e memorizza i risultati;
- valuta le prestazioni del sistema interpretando i risultati ottenuti.

Nella quarta fase è possibile richiedere al sistema l'espansione delle queries mediante WordNet.

Al fine di utilizzare più comodamente le funzionalità del sistema realizzato, è stata implementata una interfaccia grafica basata sul package javax.swing.

Una volta realizzato il sistema di IR è stata compiuta la valutazione comparativa delle prestazioni dello stesso, distinguendo due casi, quello in cui esso utilizzi WordNet per espandere le queries, o meno.

Le misure ottenute, dunque, sono state raccolte in diversi grafici, e sono state opportunamente analizzate.

Il primo Capitolo tratta la realizzazione del sistema di IR; nella Sezione 1.1 viene introdotto il motore di IR Lucene, la Sezione 1.2 descrive il modo in cui viene effettuato il parsing e l'indicizzazione dei documenti della collezione Cranfield, la sezione 1.3 tratta il parsing delle queries e l'interrogazione dell'indice creato in precedenza. La Sezione 1.4 tratta il processo di analisi del contenuto dei documenti e delle queries, mentre la Sezione 1.5 illustra la fase di valutazione delle prestazioni del sistema.

Il secondo Capitolo descrive il modo in cui viene utilizzato WordNet. La Sezione 2.1 introduce le caratteristiche di questo tesauro, mentre la Sezione 2.2 espone in maniera dettagliata come è stato implementato il processo di espansione delle queries nel sistema di IR sviluppato.

Infine, il terzo Capitolo fornisce, inizialmente, una panoramica sull'utilizzazione del sistema, tramite riferimenti grafici alle varie fasi di esecuzione; inoltre, la Sezione 3.4 riporta il confronto tra i diversi risultati ottenuti; infine, la Sezione 3.5 trae le conclusioni derivanti dallo studio proposto.

Capitolo 1

Sistema di IR

1.1 Lucene

Lucene [Lucene] è una libreria di funzioni che consente di realizzare i più disparati task di Information Retrieval (IR). Esso è un progetto open-source implementato in Java, ed è un membro della popolare famiglia di progetti Apache Jakarta. Lucene è scalabile, offre elevate prestazioni, e grazie ad esso è possibile aggiungere alle varie applicazioni diverse capacità di indicizzazione e di ricerca.

Lucene fornisce una semplice ma molto potente API, la quale richiede una conoscenza minima a proposito delle operazioni di indicizzazione e di ricerca; inoltre, il fatto che sia implementato in Java fa sì che esso non faccia assunzioni a proposito degli oggetti su cui effettua tali operazioni.

Molte persone spesso confondono Lucene con una applicazione *ready-to-use*, come un programma di ricerca di files, o un motore di ricerca per il Web. Tuttavia questo è proprio ciò che Lucene *non* è; esso, come già detto, è una libreria, ed è utilizzato come strato di software sul quale vengono costruite le applicazioni che ne fanno uso.

Lucene può indicizzare ed effettuare ricerche su ogni dato che può essere convertito in forma testuale. Esso non si cura della sorgente dei dati, del loro formato o del loro linguaggio, a patto che questi possano essere trasformati in testo; ciò significa che Lucene può essere utilizzato per indicizzare e ricercare dati contenuti in pagine ospitate su Web server remoti, documenti memorizzati in file systems locali, dati contenuti in databases, e così via.

Le prossime sezioni ed i capitoli seguenti illustreranno in che modo l'API offerta da Lucene è stata utilizzata e modificata, al fine di realizzare il sistema di IR atto a produrre i risultati che questo lavoro si prefigge.

1.2 Parsing ed indicizzazione dei documenti

La prima operazione di cui ci si è preoccupati nel realizzare il sistema di IR è stata l'identificazione del formato dei documenti che esso dovrà indicizzare. In seguito si è passati alla effettiva indicizzazione degli stessi.

Nell'ambito di questo progetto si è scelto di utilizzare il Cranfield Data Set [Cran], una collezione di 1400 documenti riguardanti nozioni di aerodinamica; tali documenti vengono presentati nell'unico file *cran.all* secondo una struttura predefinita, grazie alla quale è possibile riconoscere l'id di ogni documento, il suo titolo ed il suo autore, nonché il suo contenuto ed il suo riferimento bibliografico.

Affinché il sistema di IR possa indicizzare e referenziare ogni singolo documento della collezione, è dunque necessaria una operazione preliminare di parsing della collezione stessa. I documenti prodotti da questa fase sono le unità sottoposte alla successiva fase di indicizzazione.

A tal proposito, nelle prossime sezioni verranno descritte due classi Java che compongono il sistema di IR sviluppato,

MyDocumentParser e MyIndexMaker, le quali si occupano rispettivamente di effettuare il parsing della collezione di documenti, e di aggiungere ognuno di essi all'indice; le operazioni di analisi del contenuto che vengono effettuate preliminarmente su ogni documento e su ogni query verranno, invece, trattate in dettaglio nella Sezione 1.4.1.

1.2.1 MyDocumentParser

Il file *MyDocumentParser.java*, contenuto nel package progettobdd.parser, implementa una classe il cui unico metodo statico parse() è in grado di scorrere il file *cran.all* e di estrarre uno ad uno i documenti contenuti in esso.

Ogni documento contenuto in questo file ha inizio con l'espressione ".I X", dove X è l'identificativo numerico univoco associato al documento. Ogni volta che si incontra un'espressione di questo genere viene istanziato un oggetto di tipo Document, appartenente al package org.apache.lucene.document, al quale viene aggiunto un campo di tipo Keyword contenente l'id del documento; questo tipo di campo non viene sottoposto a nessuna operazione di analisi del contenuto durante l'indicizzazione, ma viene memorizzato nell'indice stesso così come è stato creato.

In seguito, ogni documento include l'espressione ".T", seguita da alcune righe che identificano il titolo del documento stesso. Dopo aver effettuato il parsing di queste righe, viene aggiunto all'oggetto di tipo Document un nuovo campo di tipo Keyword contenente, appunto, il titolo appena letto.

Quindi si incontra l'espressione ".A", seguita dall'autore del documento; ancora una volta, viene creato un campo di tipo Keyword contenente le informazioni appena reperite, e viene aggiunto all'oggetto di tipo Document rappresentativo del documento di cui si sta effettuando il parsing. Questa informazione, come le precedenti, non

viene analizzata quando viene indicizzata; questa scelta è stata fatta dal momento che il corpo del documento, di cui si parlerà tra breve, replica il suo titolo come prima frase, ed è stato ritenuto superfluo effettuare due volte la stessa operazione.

Un'altra espressione che il metodo parse() è in grado di trattare è ".B", la quale riporta il riferimento bibliografico del documento; ancora una volta, questa informazione viene incapsulata in un campo di tipo Keyword, il quale viene aggiunto all'oggetto di tipo Document.

Infine, l'espressione ".W" è seguita dal corpo vero e proprio del documento. Dopo aver effettuato il parsing completo di questa informazione, essa viene memorizzata in un campo di tipo Text; al contrario del campo Keyword considerato finora, un campo Text fa sì che il testo contenuto in esso venga sottoposto ad alcune operazioni di analisi atte a migliorare le operazioni di ricerca che lo coinvolgono, come la rimozione di termini non utili, o la riduzione di un termine alla sua radice lessicale, e solo in seguito venga indicizzato e memorizzato.

I passi descritti sopra hanno permesso di ricavare, a partire da un testo libero, un oggetto rappresentativo di un documento, per il quale sono stati definiti tutti i campi ed i corrispondenti valori. Per indicizzare questo documento è quindi necessario eseguire la seguente operazione:

```
writer.addDocument(doc);
```

dove *doc* è il documento appena letto, e *writer* è un oggetto di tipo IndexWriter, definito in org.apache.lucene.index, che consente di creare ed effettuare operazioni su un indice; l'oggetto di tipo IndexWriter viene passato al metodo parse() dall'esterno, in modo che all'interno del metodo stesso possa essere effettuato il parsing iterativo dell'intera collezione di documenti, ed aggiungerli uno ad uno allo stesso indice.

Nella prossima Sezione verrà trattata la classe responsabile della creazione dell'oggetto di tipo IndexWriter, la quale è anche l'autrice della chiamata al metodo parse() descritto finora.

1.2.2 MyIndexMaker

Il file *MyIndexMaker.java*, contenuto nel package progettobdd.index, implementa una classe il cui metodo statico makeIndex() è il responsabile della creazione dell'oggetto di tipo IndexWriter. Il metodo makeIndex() contiene la dichiarazione:

```
IndexWriter writer = new IndexWriter(indexName, new MyAnalyzer(false), true);
```

grazie alla quale viene istanziato l'oggetto che consente di effettuare operazioni sull'indice. Il costruttore di questo oggetto accetta tre parametri:

- una stringa che identifica il nome della directory che conterrà i file relativi al processo di indicizzazione, in questo caso contenuta in *indexName*;
- un oggetto Analyzer, definito in org.apache.lucene.analysis. Un oggetto di questo tipo possiede un metodo denominato tokenStream(), che prende in input un flusso di caratteri, ed è in grado di applicarvi una serie di "filtri" che effettuano particolari operazioni sullo stesso prima di restituirlo in output. Questo metodo viene implicitamente chiamato quando si effettua la già nota operazione writer.addDocument(doc), e viene applicato a tutti e soli i campi del documento che sono di tipo Text; in questo modo vengono eseguite le operazioni di analisi del contenuto accennate nella Sezione precedente. L'oggetto di tipo MyAnalyzer presente nell'invocazione del costruttore è definito nel package progettobdd.analyzer, ed è stato realizzato estendendo la classe Analyzer fornita da Lucene, in modo da poter fare l'overriding del metodo tokenStream(), e quindi di definirlo in base alle esigenze del sistema di IR; il valore booleano presente nel costruttore è utilizzato per indicare se avvalersi o meno di WordNet, secondo le modalità descritte nel capitolo seguente. Una trattazione

dettagliata dell'implementazione della classe MyAnalyzer è invece rimandata alla Sezione 1.4.1.

- un valore booleano, il cui valore *true* indica di creare da capo l'indice ogni volta.

Una volta che è stato istanziato l'oggetto di tipo IndexWriter, ed è quindi stato definito univocamente l'indice a cui assegnare i documenti, esso viene passato al metodo parse() descritto nella Sezione 1.2.1, che ne fa uso per aggiungere un documento per volta all'indice con il metodo writer.addDocument(doc).

Quando il metodo parse() termina il suo compito, l'indice creato viene dapprima ottimizzato e compresso, ed in seguito viene fissato chiudendo l'oggetto di tipo IndexWriter associato; a questo punto sarà stata creata la directory contenente i file di cui l'indice fa uso.

1.3 Parsing delle queries e interrogazione dell'indice

Le Sezioni precedenti hanno mostrato i passi necessari per effettuare il parsing dei documenti del Cranfield Data Set e per costruire l'indice che consente di effettuare ricerche su di essi; a questo punto è disponibile una directory creata dal sistema di IR che verrà utilizzata dalle altre componenti del sistema che si stanno per trattare.

Il Cranfield Data Set contiene anche un altro file, denominato *query.text*, il quale contiene una collezione di 225 queries a proposito degli argomenti trattati nei documenti forniti contestualmente. Come nel caso di questi ultimi, ogni singola query deve essere estratta da questo unico file prima di poter essere sottomessa al sistema di IR.

Anche il file *query.text* è formattato secondo una struttura predefinita, grazie alla quale è possibile ricavare l'identificativo numerico univoco di ogni query, ed il suo contenuto. Man mano che le queries vengono estratte, esse vengono sottomesse al sistema, il quale

interroga l'indice creato in precedenza e produce i risultati in termini di documenti rilevanti per la query, memorizzandoli in un file di testo.

Le prossime Sezioni tratteranno, quindi, dapprima la fase di parsing della collezione di queries, introducendo la classe Java MyQueryParser, ed in seguito la fase di interrogazione dell'indice e di memorizzazione dei risultati derivanti da questa operazione, discutendo la classe MyIndexSearcher.

1.3.1 MyQueryParser

Il file MyQueryParser.java implementa una classe appartenente al package progettobdd.parser, la quale contiene il metodo statico parse() che è responsabile del riconoscimento e dell'estrazione delle singole query dal file *query.text*, e provvede alla chiamata del metodo search() definito nella classe MyIndexSearcher, trattata nella prossima Sezione.

Innanzitutto, affinché sia possibile effettuare ricerche sull'indice, è necessario effettuare una operazione del tipo:

```
Searcher searcher = new IndexSearcher(indexName);
```

grazie alla quale viene istanziato un oggetto di tipo Searcher, definito nel package org.apache.lucene.search, il quale utilizza i files di indicizzazione contenuti nella directory *indexName*, disponibile dai passi precedenti.

La struttura del file delle queries è più semplice rispetto a quella del file contenente la collezione di documenti; infatti, le uniche espressioni contenute al suo interno sono ".I" e ".W", le quali si riferiscono rispettivamente all'identificativo e al corpo di ogni query. Una volta estratte le informazioni a proposito di una singola query, queste vengono utilizzate per la chiamata:

```
MyIndexSearcher.search(id, query, threshold, searcher, wordnet);
```

la quale fa, appunto, riferimento al già introdotto metodo search() della classe MyIndexSearcher. I parametri presenti nella chiamata di questo metodo sono 5:

- *id*, l'identificativo numerico univoco della query che sta per essere sottomessa al sistema;
- *query*, una stringa nella quale è contenuta la vera e propria query;
- *threshold*, un valore di tipo double compreso tra 0 e 1, che indica la soglia di similarità oltre la quale un documento viene considerato rilevante per la query;
- *searcher*, l'oggetto di tipo Searcher già descritto sopra;
- *wordnet*, un valore booleano il quale indica se avvalersi dell'espansione della query tramite WordNet o meno.

Il metodo descritto sopra viene, dunque, utilizzato per ognuna delle queries presenti nel file *query.text*, ed è il responsabile dell'interrogazione dell'indice, nonché del trattamento dei risultati derivanti da questa operazione. La prossima Sezione descrive la classe MyIndexSearcher, nella quale il metodo search() è stato definito.

1.3.2 MyIndexSearcher

Il file *MyIndexSearcher.java*, definito in progettobdd.index, implementa la classe contenente il metodo search() descritto in precedenza. All'interno di questo metodo è possibile notare le seguenti operazioni:

```
Query q = QueryParser.parse(query, "body", new MyAnalyzer(wordnet));
Hits hits = searcher.search(q);
```

Inizialmente viene invocato il metodo statico parse() della classe QueryParser, offerta da Lucene nel package org.apache.lucene.queryParser; questo metodo accetta tre parametri:

- *query*, la stringa contenente la query di cui si è effettuato il parsing, che viene passata dal metodo search() della classe descritta nella Sezione precedente;
- una stringa che indica su quale campo del documento ricercare i termini contenuti nella query, settata a *"body"*;
- un oggetto di tipo Analyzer; anche in questa circostanza viene utilizzata la classe ridefinita secondo le nostre esigenze MyAnalyzer, inizializzata tramite il valore booleano *wordnet*, anch'esso passato dal metodo search() della classe MyQueryParser.

L'invocazione di questo metodo porta alla creazione di un oggetto di tipo Query, il quale rappresenta una query nel formato che Lucene è in grado di trattare. Questo oggetto viene, dunque, utilizzato nell'invocazione del metodo search() sull'oggetto di tipo Searcher, e restituisce un oggetto di tipo Hits che contiene tutti i documenti che hanno una rilevanza non nulla nei confronti della query sottomessa. Il calcolo della rilevanza di un documento viene effettuato grazie alla cosiddetta DefaultSimilarity, la cui formula viene riportata di seguito:

$\sum_{t \text{ in } q}$ tf(t in d) * idf(t) * boost(t.field in d) * lengthNorm(t.field in d)

ed il cui risultato è garantito essere compreso tra 0 ed 1. Le componenti di questa formula sono:

- *tf(t in d)*: term frequency per il termine t nel documento d;
- *idf(t)*: inverse document frequency per il termine t;
- *boost(t.field in d)*: fattore di moltiplicazione di un campo, non utilizzato;

- *lenghtNorm(t.field in d)*: valore normalizzato di un campo, dato il numero di termini al suo interno.

Una volta che l'oggetto di tipo Hits è stato inizializzato, viene utilizzato il parametro *threshold* di cui il metodo parse() della classe MyIndexSearcher dispone. I documenti contenuti nell'oggetto Hits vengono estratti uno ad uno, e viene verificato che il loro valore di similarità sia maggiore o uguale a quello specificato da threshold; in caso positivo, gli identificativi della query e del documento rilevante per essa vengono aggiunti ad un file di testo, che tornerà utile in fase di valutazione delle prestazioni del sistema; in caso negativo, non viene effettuata nessuna operazione aggiuntiva.

Alla fine di questa fase, dunque, è disponibile un file di testo contenente gli identificativi dei documenti che, per ciascuna query, sono stati giudicati essere rilevanti almeno quanto stabilito dal parametro threshold.

1.4 Analisi del contenuto dei documenti e delle queries

Nelle Sezioni 1.2.2 e 1.3.2 è stato accennato il significato degli oggetti di tipo Analyzer offerti da Lucene. Questi oggetti hanno il compito di analizzare il flusso di caratteri che potrebbe rappresentare alternativamente un documento o una query, e di applicare su di esso una serie di tecniche molto conosciute nell'ambito dell'IR, le quali hanno l'effetto di migliorare sensibilmente le prestazioni delle ricerche sui documenti indicizzati dal sistema. Come è stato possibile notare, queste operazioni hanno luogo nel momento in cui un documento viene aggiunto all'indice, e quando una query viene trasformata in un oggetto che il sistema di Ir costruito su Lucene è in grado di trattare.

Lucene mette a disposizione diversi oggetti di tipo Analyzer nel package org.apache.lucene.analysis, ognuno dei quali definisce una

serie prestabilita di operazioni sul testo; ad esempio, StandardAnalyzer identifica le parole contenute all'interno del flusso di caratteri per mezzo di una grammatica costruita con JavaCC, trasforma le lettere maiuscole in minuscole, e rimuove alcune parole considerate ininfluenti per le operazioni di ricerca, denominate "stopwords". Un altro esempio di questo tipo è SimpleAnalyzer, il quale riconosce le parole separandole alle occorrenze di alcuni caratteri considerati "non-letters", e trasforma ogni lettera maiuscola dei tokens risultanti in minuscola.

Nel realizzare il sistema di IR in esame, è stato ritenuto opportuno definire un proprio tipo di oggetto di tipo Analyzer, il quale sia in grado di effettuare tutte e sole le operazioni sul testo considerate utili ai fini dello studio proposto. Per questo motivo, nella prossima Sezione verrà discussa la classe MyAnalyzer, già incontrata a proposito delle fasi di indicizzazione dei documenti e di sottomissione delle queries al sistema, e verranno discusse in dettaglio le operazioni da essa eseguite.

1.4.1 MyAnalyzer

Il package progettobdd.analyzer contiene il file *MyAnalyzer.java*, il quale implementa la classe MyAnalyzer, già ampiamente introdotta. Questa classe estende la classe astratta Analyzer fornita da Lucene, ed effettua l'overriding del metodo tokenStream(); in questo modo, tutte le operazioni che è possibile effettuare grazie ad un oggetto Analyzer saranno ancora possibili con un oggetto di tipo MyAnalyzer, ma verranno esplicitamente eseguiti i passi definiti in quest'ultima classe.

Il metodo tokenStream(), come anticipato in precedenza, prende in input un flusso di caratteri, identificato da un oggetto di tipo Reader; la prima operazione che viene effettuata sul flusso di caratteri è la seguente:

```
TokenStream result = new LowerCaseTokenizer(reader);
```

la quale costruisce un oggetto di tipo LowerCaseTokenizer e lo assegna ad uno di tipo TokenStream; entrambe queste classi sono definite nel package org.apache.lucene.analysis, e questa assegnazione fa sì che il flusso di testo contenuto in *reader* venga separato all'occorrenza di caratteri denominati "non-letters", e venga convertito in caratteri minuscoli. Il flusso di token risultante da questa operazione viene assegnato all'oggetto *result*, il quale alla fine sarà restituito in output da metodo tokenStream().

In seguito viene eseguita la seguente operazione:

```
result = new StopFilter(result, StopAnalyzer.ENGLISH_STOP_WORDS);
```

con la quale viene istanziato un oggetto di tipo StopFilter che utilizza l'oggetto *result* derivante dall'operazione precedente, e lo assegna nuovamente allo stesso oggetto. In questo modo il flusso di token ricavato in precedenza viene ulteriormente "filtrato", ed in particolare vengono rimosse alcune parole inglesi comuni che non sarebbero utili ai fini della ricerca dei documenti, le quali sono contenute nell'array di stringhe *ENGLISH_STOP_WORDS* della classe StopAnalyzer.

Quindi, esclusivamente nel caso in cui si stia analizzando il contenuto di una query, viene eseguita una ulteriore operazione di trattamento del testo contenuto nel flusso di token *result*:

```
result = new LowerCaseTokenizer(MyQueryExpander.expand(result));
```

che consente di aggiungere alla query alcuni termini affini al suo contenuto grazie all'uso di WordNet. La spiegazione dettagliata del funzionamento di WordNet, ed in particolare del metodo expand() della classe MyQueryExpander, verrà fornita nel Capitolo 2. Ora si fa notare che questo metodo restituisce un oggetto di tipo Reader, il quale viene trasformato in un TokenStream grazie all'istanziazione di un oggetto di

tipo LowerCaseTokenizer. Inoltre, dal momento che l'operazione di espansione del testo tramite WordNet deve essere eseguita esclusivamente nel caso di una query, essa è controllata da un flag booleano che viene settato al momento della costruzione dell'oggetto MyAnalyzer; ecco quindi spiegata la differenza tra le operazioni:

```
IndexWriter writer = new IndexWriter(indexName, new MyAnalyzer(false), true);

Query q = QueryParser.parse(query, "body", new MyAnalyzer(wordnet));
```

Nel primo caso, il valore booleano passato a MyAnalyzer è sattato a *false*, dal momento che non è necessario espandere con WordNet il contenuto dei documenti; nel secondo caso, invece, il valore booleano viene passato dall'esterno, è sarà *true* nel caso si voglia espandere il contenuto di una query con WordNet, *false* altrimenti.

L'ultima operazione che viene effettuata prima di restituire l'oggetto di tipo TokenStream è la seguente:

```
result = new PorterStemFilter(result);
```

la quale applica l'algoritmo di stemming definito da Martin Porter, che riduce ogni parola alla sua radice lessicale, al fine di ridurre il numero di termini diversi di cui tenere conto. Infine, l'oggetto *result* così creato viene restituito.

1.5 Valutazione delle prestazioni del sistema

Dopo aver indicizzato i documenti della collezione, ed aver sottomesso al sistema le queries associate, devono essere ricavate le prestazioni del sistema stesso, in modo da poter confrontare i risultati ottenuti dalle ricerche effettuate con o senza l'ausilio di WordNet.

La fase di sottomissione delle queries termina con la produzione di un file di testo contenente gli identificativi dei documenti rilevanti per le queries stesse. Questo file è organizzato in una struttura identica a quella del file *qrels.text*, fornito con la collezione Cranfield, il quale a sua volta contiene una serie di giudizi di rilevanza compilati a mano da esperti del settore.

La prossima Sezione tratterà l'operazione di lettura e di confronto dei risultati contenuti nel file prodotto dal sistema di IR e dal file "oracolo" *qrels.text*. Le prestazioni del sistema verranno, infine, valutate tramite indici quali la Macro-Precision (Mprecision), la Macro-Recall (Mrecall), e la F-Measure, tutte calcolate al variare di un threshold di accettazione compreso tra 0 e 1.

1.5.1 MyPerformanceEvaluator

Il file *MyPerformanceEvaluator.java*, contenuto nel package progettobdd.performance, implementa una omonima classe che è responsabile della valutazione delle prestazioni del sistema di IR.

Questa classe definisce il solo metodo evaluatePerformance(), il quale prende in input il percorso del file contenente i risultati del sistema e quello del file contenente i giudizi di riferimento sulle queries; questo metodo costruisce al suo interno delle strutture dati atte a manipolare le informazioni contenute in questi files, e grazie alle quali è possibile calcolare i valori prestazionali di Mprecision, Mrecall e F-Measure.

Il valore di Mprecision è ricavato mediando sul numero di queries il rapporto tra documenti rilevanti ritrovati ed il numero di documenti ritrovati per ogni query; in questo calcolo si è tenuto conto del fatto che, se il sistema non ha restituito nessun documento rilevante per una data query, la sua precision è pari ad 1.

Il valore di Mrecall è stato calcolato allo stesso modo, considerando il rapporto tra numero di documenti rilevanti ritrovati e numero di documenti rilevanti in totale.

Il valore di F-Measure viene ricavato calcolando la media armonica tra i precedenti indici. Tutti questi valori vengono restituiti dal metodo evaluatePerformance() all'interno di un array di oggetti di tipo double.

Capitolo 2

Espansione della query con Wordnet

2.1 WordNet

WordNet [WordNet] è un sistema di riferimenti lessicali on-line (cioè un database lessicale che può essere letto da un computer) il cui design è ispirato dalle ultime teorie psico-linguistiche della memorizzazione lessicale umana. I nomi, i verbi e gli aggettivi sono organizzati in insiemi di sinonimi, ed ognuno di questi rappresenta un concetto lessicale fondamentale. Le differenti relazioni collegano gli insiemi di sinonimi.

WordNet è una proposta per una efficace combinazione tra le tradizionali informazioni lessicografiche e la moderna computazione ad alta velocità basata su calcolatori elettronici.

La maggiore differenza tra WordNet e un dizionario standard è che WordNet divide il lessico in quattro categorie: sostantivi, verbi, aggettivi e avverbi (il sistema e' organizzato in strutture di grafi indipendenti per ogni categoria grammaticale). Il prezzo di imporre tali categorie sintattiche è un certo ammontare di ridondanza che i dizionari convenzionali evitano, ma il vantaggio è che le differenze fondamentali

nell'organizzazione semantica di queste categorie sintattiche può essere vista con chiarezza e sfruttata sistematicamente.

Tuttavia, la caratteristica più ambiziosa di WordNet è il tentativo di organizzare le informazioni lessicali in termini di significati di parole piuttosto che tramite la forma delle stesse. Considerato ciò, WordNet è più un tesauro che un dizionario.

Gli insiemi di sinonimi in WordNet sono organizzati secondo relazioni lessicali (ad esempio sinonimia, antinomia), di tipo semantico o concettuali (iperonimia, iponimia), mentre sono descritti da glosse, che consistono in esempi d'uso.

2.2 Uso di WordNet nel sistema di IR

WordNet è stato impiegato nel sistema di IR per espandere le queries messe a disposizione dalla collezione di documenti CRAN e valutare le prestazioni (cioè calcolare i valori di Precision, Recall e F-measure) del sistema di IR.

La valutazione delle prestazioni è stata effettuata mettendo a confronto i risultati forniti dal sistema utilizzando le queries senza alcuna espansione e con espansione tramite WordNet.

Prima di procedere con l'espansione delle queries, i termini di queste ultime sono stati ridotti in caratteri minuscoli e, tramite l'ausilio di una lista di parole, sono state rimosse dalle query le cosiddette "stopwords": le parole, come ad esempio le congiunzioni e gli articoli, che non hanno impatto sulla semantica di un documento, ma che rallentano il processo di analisi della query in questo caso e in generale l'analisi di testo.

L'espansione è compiuta analizzando una query per volta: per ogni termine che compone la query il sistema ricerca i sinonimi e li aggiunge alla query espansa, che a sua volta viene utilizzata per ricercare i documenti rilevanti per tale query.

È stato deciso tuttavia di estendere la query ricercando i sinonimi delle parole che sono sostantivi, o comunque possono anche essere tali, escludendo gli altri termini come aggettivi, verbi e avverbi.

Tale decisione è stata presa in quanto sono i sostantivi ad avere il maggiore effetto caratterizzante un documento rispetto ad un altro, e si è così evitato di espandere la query eccessivamente.

È stato possibile interfacciare WordNet a Lucene grazie alle API JWordNet [JWordNet]. JWordNet è fondamentalmente un'interfaccia orientata agli oggetti, stand-alone e scritta in Java, al database delle relazioni lessicali di WordNet.

Nella prossima sezione è riportata la descrizione della classe Java che è stata scritta per impiegare il database di WordNet impiegando JWordNet.

2.2.1 MyQueryExpander

Come è stato precedentemente anticipato, tale classe ha il compito di espandere le queries della collezione presa in esame facendo uso di JWordNet per attingere al database di WordNet.

La classe MyQueryExpander presenta un unico metodo statico chiamato expand che prende in input un oggetto di tipo TokenStream, che contiene i termini della query da espandere, e restituisce un oggetto Reader in output, che rappresenta la query espansa.

La seguente istruzione:

```
DictionaryDatabase dictionary = new FileBackedDictionary("/wordnetdata");
```

istanzia un nuovo oggetto di tipo FileBackedDictionary e il costruttore accetta come parametro il path della cartella in cui si trovano i file che costituiscono il database di WordNet.

In seguito, tramite il metodo lookupIndexWord() della classe FileBackedDictionary, viene effettuata una ricerca di ogni parola nel database specificando tramite il parametro *POS* (part of speech) la categoria lessicale che, come precedentemente evidenziato, corrisponde a *NOUN*. La ricerca produce un insieme di parole che sono sinonimi alla parola data come input; questi sinonimi infine vengono concatenati in una stringa di testo che andrà a formare la query espansa.

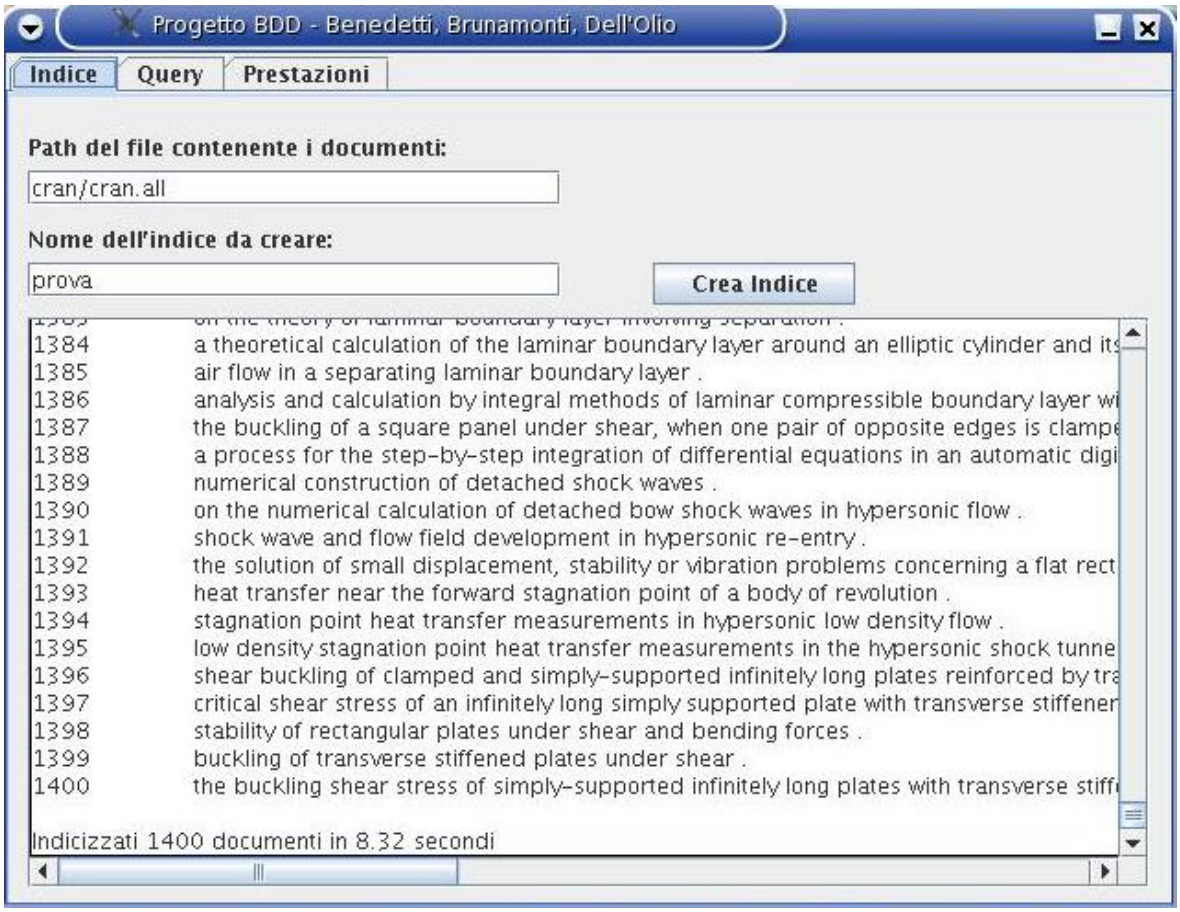
Il metodo expand(), una volta terminata l'analisi della query ottenuta in input, restituisce un oggetto Reader inizializzato con la stringa che contiene la query espansa e termina.

Capitolo 3

Esecuzione del sistema di IR e risultati

In questa Sezione verranno mostrate le fasi di esecuzione del sistema di IR realizzato; per ogni fase verrà fornito un riferimento grafico, ed una descrizione delle funzionalità utilizzate e dell'output prodotto.

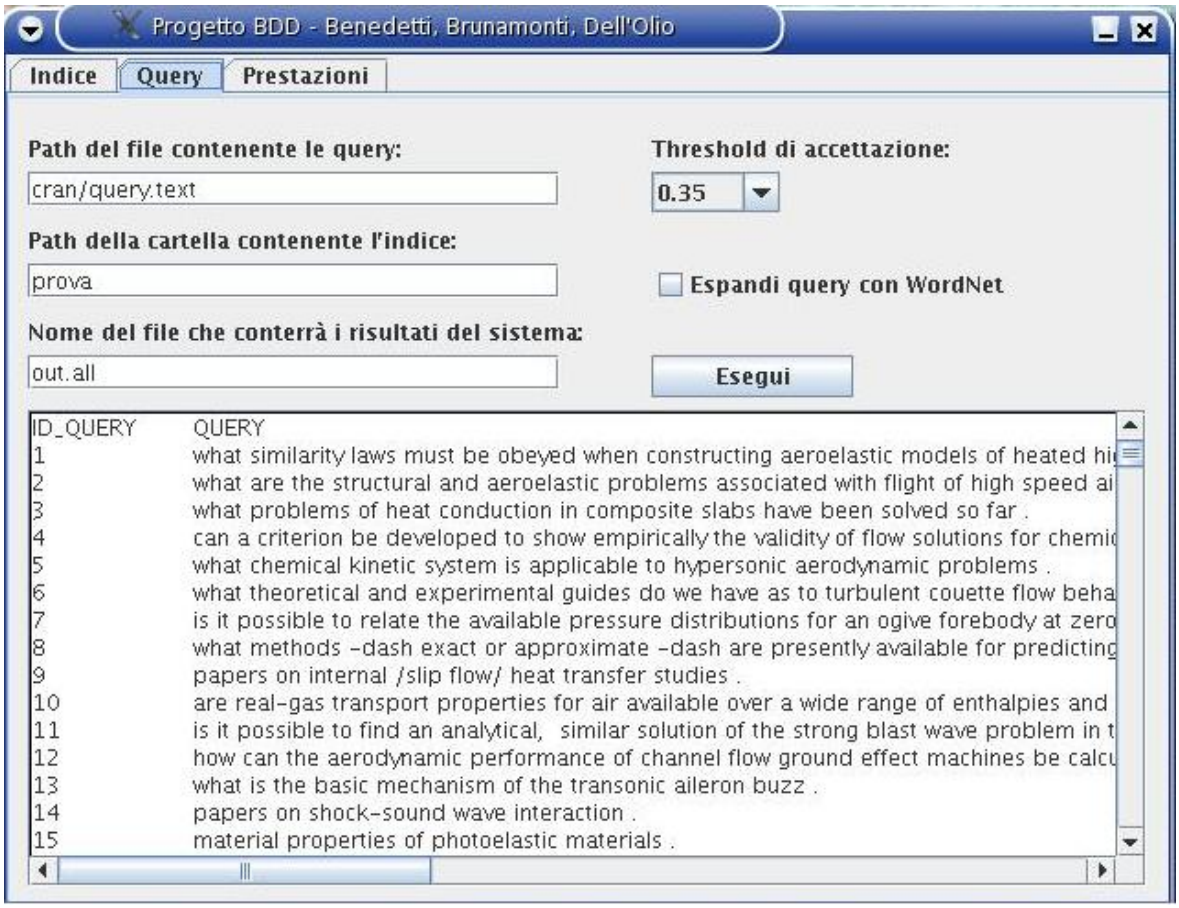
3.1 Parsing ed indicizzazione dei documenti



In questa immagine è rappresentata l'esecuzione della fase di parsing ed indicizzazione dei documenti della collezione. L'interfaccia grafica del sistema permette di specificare il percorso del file contenente i documenti ed il nome della cartella dell'indice che verrà creato.

Al termine dell'esecuzione di questa fase è possibile osservare in output la lista di documenti indicizzati ed il tempo con cui questa fase è stata portata a termine.

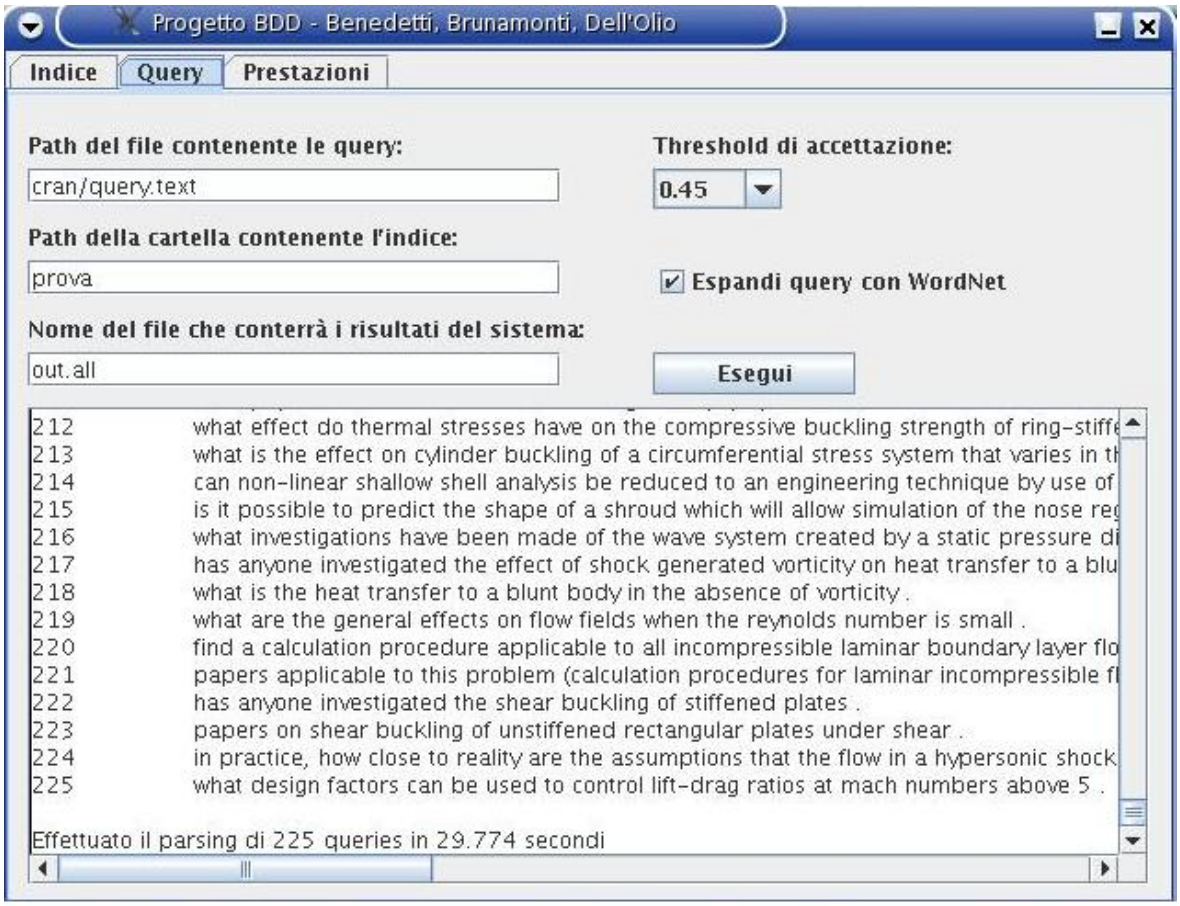
3.2 Parsing delle queries ed interrogazione dell'indice



In questa figura è mostrata l'esecuzione grafica della fase di parsing delle queries e di sottomissione di queste ultime all'indice. L'interfaccia grafica permette di specificare il percorso del file

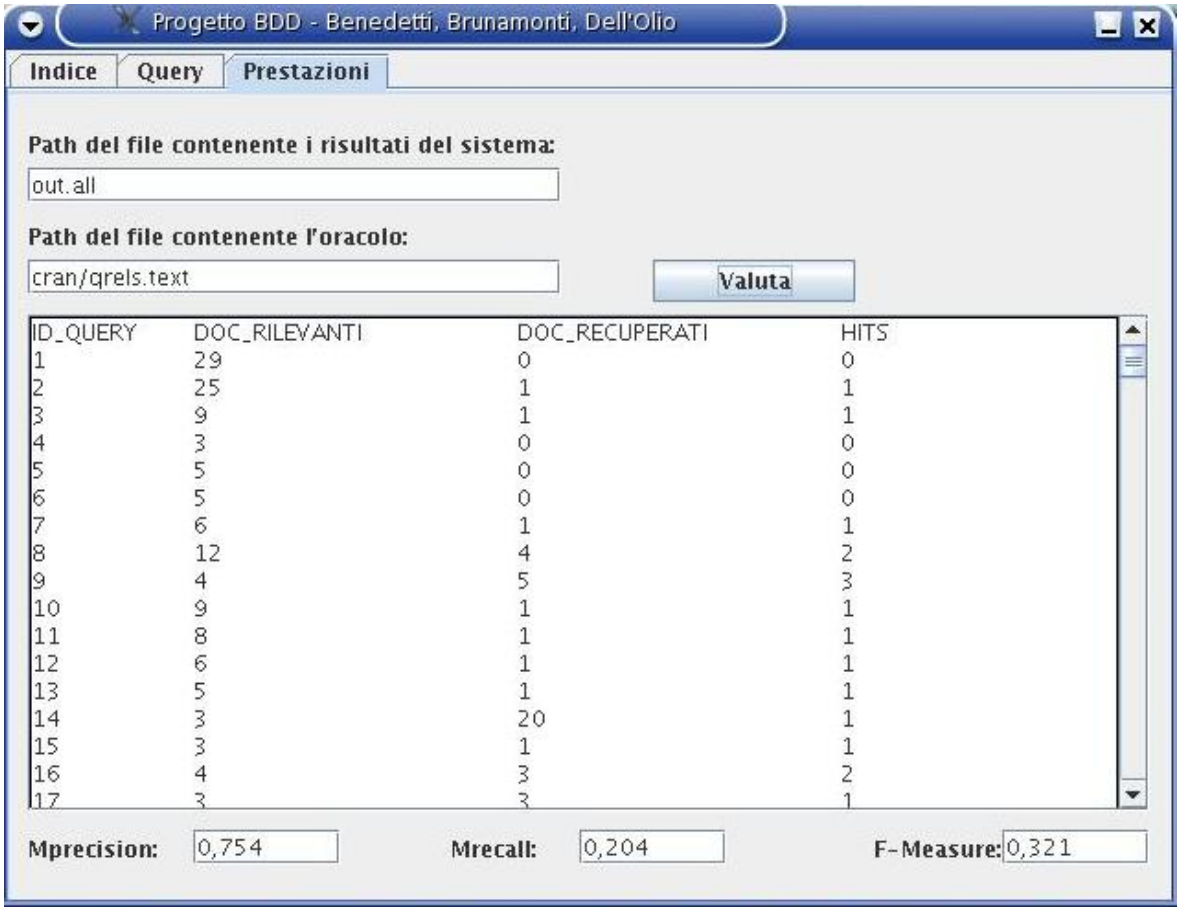
contenente le queries, il percorso della cartella che contiene l'indice ed il nome del file che conterrà i risultati del sistema.

Inoltre, può essere specificato il threshold di accettazione, compreso tra 0.05 e 0.95, e scegliere se espandere le queries per mezzo di WordNet o meno.



Anche questa immagine mostra la stessa fase, con la differenza che verrà utilizzata l'espansione delle queries tramite WordNet. In output è possibile osservare la lista delle queries lette e sottomesse al sistema, ed il tempo di esecuzione di questa fase.

3.3 Valutazione delle prestazioni del sistema



In questa ultima immagine viene mostrata la fase di valutazione delle prestazioni del sistema di IR realizzato; l'interfaccia grafica permette di specificare il percorso del file contenente i risultati del sistema, ed il percorso del file contenente i giudizi ufficiali sulle queries.

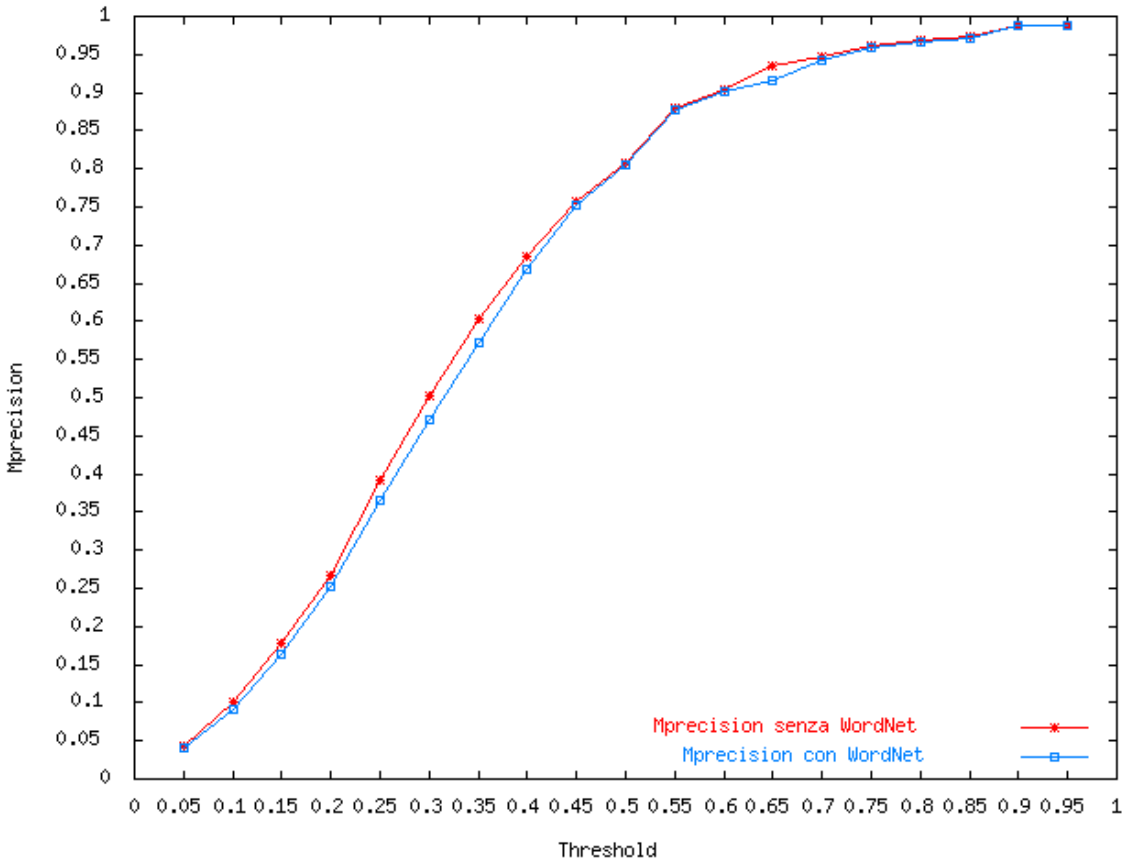
L'output ottenuto mostra, per ogni query, il numero di documenti rilevanti per essa, il numero di documenti recuperati dal sistema, ed il numero di matches tra questi. In basso, infine, vengono riportati esplicitamente i risultati degli indici di Mprecision, Mrecall, ed F-Measure.

3.4 Risultati

Le prossime Sezioni presenteranno i risultati ottenuti dal sistema sia nel caso in cui le queries vengano sottomesse così come sono reperite, sia nel caso in cui esse siano espanse con WordNet.

Per ogni indice di prestazione verrà riportato un grafico che ne esprime il valore al variare del threshold di accettazione, e verrà fornito un breve commento allo stesso.

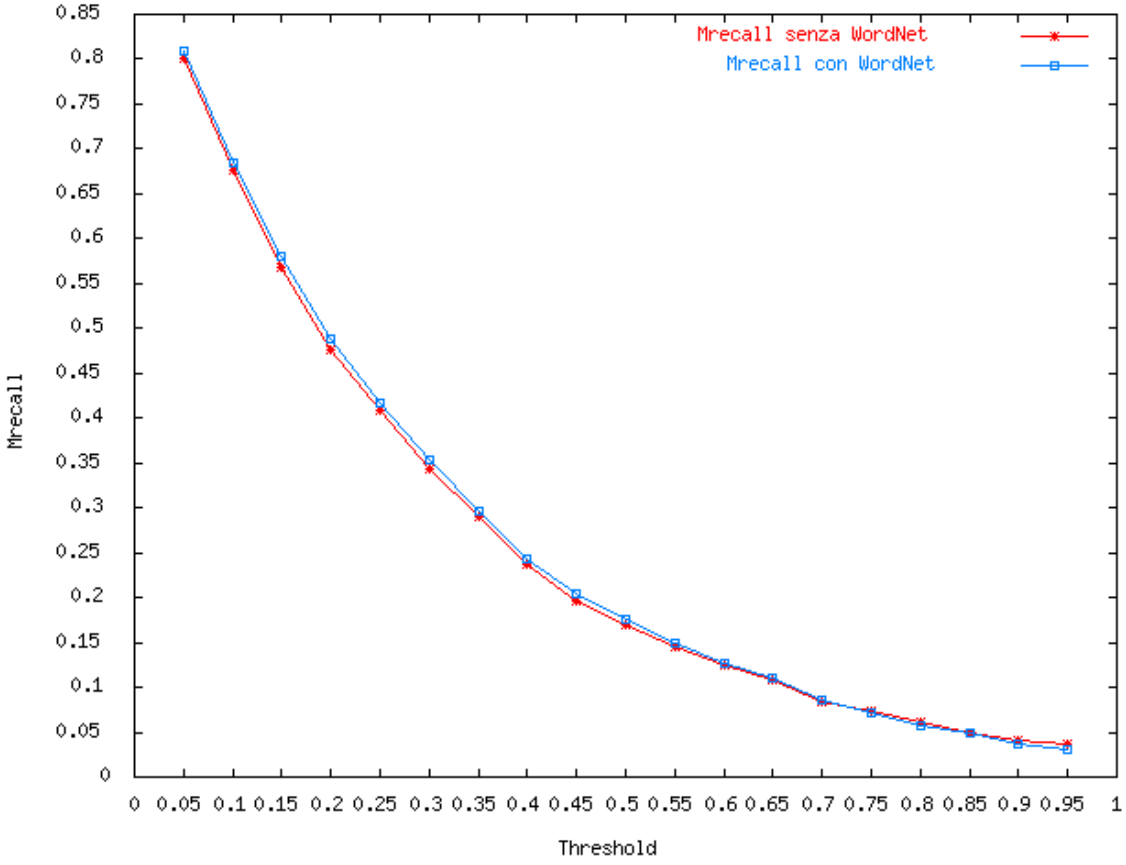
3.4.1 Mprecision



Il grafico mostrato riporta i valori di Mprecision ottenuti dal sistema senza l'uso di WordNet (rosso) e con l'uso di WordNet (blu). Si può notare che i valori ottenuti sono pressoché identici, e nel complesso

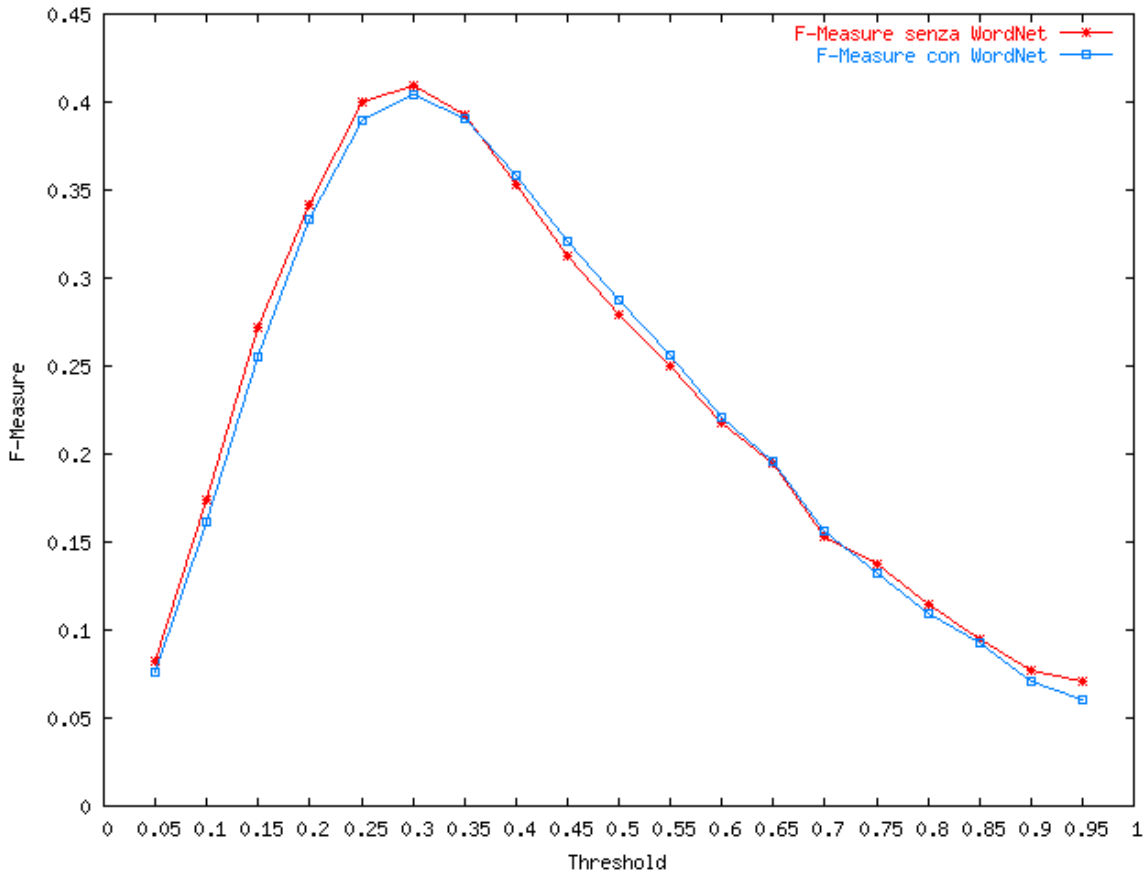
la Mprecision è inferiore utilizzando WordNet, come da letteratura, in particolare per valori di threshold fino a 0.45 e compresi tra 0.6 e 0.7.

3.4.2 Mrecall



Il grafico mostra l'andamento della Mrecall in funzione della threshold di accettazione sia utilizzando WordNet (blu) sia senza l'utilizzo di WordNet (rosso). Come è possibile osservare dal grafico, l'utilizzo di WordNet è vantaggioso scegliendo valori di threshold inferiori 0.6, è sostanzialmente indifferente per valori compresi tra 0.6 e 0.85, mentre è svantaggioso per valori più elevati. Nel complesso si può comunque dire che l'impiego di WordNet migliora, seppur di poco, la Mrecall, come è possibile riscontrare in studi affini.

3.4.2 F-Measure



Il grafico riportato mostra la F-Measure in funzione del threshold di accettazione distinguendo nei casi di utilizzo di WordNet (blu) o meno (rosso).

Dal momento che la F-Measure fornisce un valore che tiene conto sia della Mprecision che della Mrecall, il grafico suggerisce che è più conveniente utilizzare il sistema di IR che non fa uso di espansione con WordNet, ovviamente nelle ipotesi in cui è stato posto il sistema realizzato e l'espansione con WordNet proposta.

Analizzando il grafico infatti, per valori di threshold inferiori a 0.35 è più leggermente più performante il sistema senza WordNet, per valori compresi tra 0.35 e 0.65 è più performante il sistema con WordNet, mentre per valori di threshold superiori a 0.65 il sistema senza WordNet torna ad avere prestazioni migliori.

3.5 Conclusioni

In conclusione è possibile affermare che:

- complessivamente la Mprecision peggiora con l'utilizzo di WordNet;
- complessivamente la Mrecall migliora con l'utilizzo di WordNet;
- dato che la diminuzione della Mprecision non è bilanciata da almeno corrispondente miglioramento della Mrecall, la F-Measure è tendenzialmente superiore nel caso in cui il sistema non utilizzi WordNet.

Questi risultati sono da intendersi riferiti alla implementazione dello specifico sistema di IR, nonché allo specifico modello di espansione delle queries basato su WordNet sviluppato.

Bibliografia

[Cran] Cranfield Data Set
ftp://ftp.cs.cornell.edu/pub/smart/cran/

[JWordNet] JWordNet interface
http://jwn.sourceforge.net/

[Lucene] Apache Lucene
http://lucene.apache.org/

[WordNet] WordNet lexical database
http://wordnet.princeton.edu/