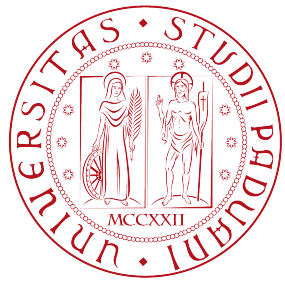
UNIVERSITÀ DEGLI STUDI DI PADOVA

Dipartimento di Ingegneria dell'Informazione

Corso di Laurea Magistrale in Ingegneria Informatica



PROGETTAZIONE E REALIZZAZIONE DI UN SISTEMA DI ENTERPRISE SEARCH

Relatore:
Prof. Maristella Agosti

Correlatore:
Dott. Daniele Turato

Laureando:
Daniel Zilio
Matricola: 1081854

Anno Accademico 2015-2016

Ringraziamenti

Desidero ringraziare l'*IMS Research Group* del Dipartimento di Ingegneria dell'Informazione dell'Università di Padova, in particolare l'Ing. Gianmaria Silvello per i suggerimenti e le discussioni utili alla realizzazione della tesi.
Ringrazio l'azienda *Siav Spa* per la collaborazione.
Ovviamente ringrazio infine la mia famiglia per il sostegno profuso in questi anni di studi universitari.

Indice

Elenco delle tabelle

Elenco delle figure

Codice

Capitolo 1

Introduzione

Questa tesi nasce con l'intento di trasferire le conoscenze acquisite durante la formazione universitaria nel campo del reperimento dell'informazione, nel contesto di una realtà aziendale. Spesso il significativo potenziale informativo che le organizzazioni possiedono resta inutilizzato in archivi digitali, e viene considerato più un peso che una risorsa. Uno dei motivi principali di questo problema è che il recupero dei contenuti informativi, di cui i documenti sono portatori, non è di immediata risoluzione e spesso è visto dalle aziende come un investimento troppo importante da sostenere. Ordinativi, rendiconti o semplici comunicazioni rimangono dimenticati nelle memorie sotto forma di file. Vi è anche una innaturale e diffusa diffidenza, se non a volte scarsa qualità di analisi del problema, nel non considerare la grande importanza delle informazioni contenute nel flusso documentale costantemente generato. Tale attenzione ai contenuti è invece necessaria nelle organizzazioni che intendono farsi portatrici di metodologie di eccellenza, come quelle inerenti al *Total Quality Management*.
Partendo da queste premesse si è condotto il lavoro di tesi, con l'intento di favorire il processo di recupero e gestione dei documenti testuali aziendali mediante l'applicazione di metodologie del reperimento dell'informazione. Il lavoro è stato svolto in collaborazione con l'azienda *Siav Spa*[1]. Uno dei principali prodotti di questa azienda è Archiflow, un software che permette la gestione e il mantenimento dei documenti che vengono generati da

[1] https://www.siav.com/it/

un'organizzazione. L'obiettivo finale della tesi è stato quello di realizzare un prototipo che permetta il reperimento di informazioni di interesse aziendale, a partire da una collezione di documenti gestiti da Archiflow anche integrando nella ricerca elementi informativi provenienti da altre sorgenti di dati.

La tesi è composta da sei capitoli, ed è strutturata in modo da documentare i risultati raggiunti nelle principali fasi del progetto di realizzazione del prototipo. Dopo il presente capitolo introduttivo, nel capitolo 2 si descrivono gli aspetti salienti della disciplina del reperimento dell'informazione, specificando gli elementi principali, considerati poi come riferimento nel proseguo del lavoro di progettazione.
Nel capitolo 3 si riportano i risultati del confronto di due motori di ricerca, Solr ed Elasticsearch, ritenuti i principali candidati per la realizzazione del prototipo. Il confronto è stato effettuato analizzando diverse funzionalità chiave e grazie al supporto della documentazione tecnica. Il capitolo fornisce quindi una visione d'insieme del funzionamento di entrambi i sistemi e termina con una loro comparazione diretta.
Nel capitolo 4 si è descritto il lavoro di progettazione e realizzazione del prototipo, realizzato mediante Apache Solr. Tale lavoro è stato caratterizzato dall'analisi dei requisiti eseguita in stretta collaborazione con l'azienda Siav, dove sono emersi i fattori chiave da considerare per raggiungere l'obiettivo fissato. I risultati del lavoro vengono presentati in due parti: nella prima viene presentata la metodologia utilizzata per l'estrazione del testo dai documenti considerati, nella seconda vengono proposte diverse possibili configurazioni per il motore di ricerca utilizzato.
Nel 5 viene illustrata la metodologia di valutazione adottata per valutare la qualità del prototipo realizzato. In particolare si affronta la complessa problematica della realizzazione di una collezione di test sperimentale.
Nel capitolo finale 6 si delineano le conclusioni a seguito del lavoro svolto e si forniscono delle indicazioni per un eventuale sviluppo futuro del lavoro.
Si precisa che per quanto riguarda la sitografia consultabile alla fine della tesi, l'integrità dei collegamenti ipertestuali presenti è stata validata fino a luglio 2016, data di pubblicazione della tesi.

Strumenti utilizzati

Gli strumenti che sono stati utilizzati per la realizzazione di questo progetto sono elencati qui di seguito:

- Computer con un processore Intel Xeon(R) CPU X5560, 2.80 GHz e 8 GB di RAM ; il sistema operativo installato è Windows Server 2012 R2.
- **Apache Solr 6.0.0**. Motore di ricerca utilizzato per lo sviluppo del prototipo.
- **Microsoft Visual Studio Enterprise 2015**. Ambiente di sviluppo utilizzato per l'estrazione del testo dai documenti forniti e per il loro successivo utilizzo da parte di Solr.
- **Eclipse Mars**. Ambiente di sviluppo utilizzato per la lettura dei documenti in formato `.XES` e il loro successivo utilizzo da parte di Solr.
- **MATLAB**, in particolare la libreria **MATTERS**.
- **Notepad++** e **Sublime Text**. Editor di testo utilizzati come supporto nella realizzazione del progetto.
- **Cygwin**. Console che simula una shell di Linux, utilizzata per le operazioni manuali di tipo cURL verso Solr.
- La tesi è stata scritta in **LaTeX** attraverso l'editor online **ShareLaTex**[2].

[2] https://it.sharelatex.com/

Capitolo 2

Il reperimento dell'informazione

Il *reperimento dell'informazione* (in inglese *Information Retrieval*, abbreviato in IR) è la disciplina che si occupa di rappresentare, memorizzare, ricercare e recuperare oggetti contenenti informazioni quali documenti di testo, pagine web, messaggi di posta elettronica, e documenti multimediali, quindi in generale risorse informative che siano rappresentabili digitalmente. Con il crescente numero di documenti digitali grazie all'aumentata informatizzazione della documentazione amministrativa, al continuo incremento di contenuti presenti nel web fino alla nuova fonte che sono le tecnologie IoT(*Internet of Things*), la necessità di avere dei sistemi che permettano di recuperare velocemente ed efficacemente risorse informative digitali sta diventando sempre più importante e decisiva per sfruttare al meglio il potenziale informativo di cui esse sono portatrici.
L'IR è una disciplina multidisciplinare che va oltre l'informatica, abbracciando campi quali la linguistica, la statistica e la psicologia. Questo perché alla progettazione di modelli matematici, si affiancano ad esempio, analisi dei comportamenti dell'utente finale e lo studio di come è strutturato un testo in una determinata lingua.

Sebbene molti ritengano che si tratti di una disciplina nata con la diffusione dell'uso della rete Internet e delle sue applicazioni, nel suo uso più noto che è nella ricerca di pagine web, il reperimento dell'informazione ha origini più antiche, che combaciano con quelle risalenti alla costruzione dei

primi calcolatori elettronici. Il primo brevetto di una macchina adibita alla ricerca di documenti risale infatti al 1927, con la cosiddetta "Statistical Machine" di E. Goldberg[1], mentre importanti studi furono condotti a partire dalla fine degli anni 1950, tra tutti quelli di H.P. Luhn sull'importanza della frequenza delle singole parole in un testo, fino ai due esperimenti di C. Cleverdon, che gettarono le basi per quella che è la valutazione nel campo dell'*Information Retrieval*[2]. Ad oggi la disciplina è in continua evoluzione e numerosi studi vengono annualmente pubblicati, aprendo nuovi filoni di ricerca.

Un sistema di reperimento dell'informazione, abbreviato IRS, rappresenta un insieme di operazioni che vengono realizzate con lo scopo di restituire a chi lo utilizza un insieme, ordinato o meno, di documenti atti a soddisfare una sua specifica esigenza informativa. Di seguito sono descritti, in modo sintetico, i componenti principali che lo compongono. I concetti discussi in questa tesi sono presentati facendo riferimento a questo esempio di architettura che quindi fungerà da guida per la lettura. L'ultima sezione del capitolo è dedicata ad una introduzione al campo più specifico dell'*enterprise search*, oggetto di studio di questa tesi.

2.1 Sistema di reperimento dell'informazione

In figura 2.1 il rettangolo interno rappresenta un sistema di reperimento dell'informazione e i suoi componenti principali. In ingresso a tale sistema vi sono i documenti e la query, che è l'espressione dell'esigenza informativa dell'utente, ovvero ciò che egli vuole ricercare all'interno della collezione. Questo aspetto è centrale nell'ambito dell'IR, in quanto la sfida principale in tale disciplina è la comprensione di ciò che effettivamente serve al fruitore del sistema per colmare una sua lacuna informativa. In uscita si hanno invece il documento o i documenti, ordinati o meno che, auspicabilmente,

[1] Per maggiori dettagli riguardo tale invenzione si veda http://people.ischool.berkeley.edu/~buckland/statistical.html

[2] Per un approfondimento sulla storia del reperimento dell'informazione si consiglia l'articolo *The History of Information Retrieval Research* di M. Sanderson e W. Bruce Croft.

soddisfino l'esigenza informativa fornita in ingresso. Un documento che adempi a tale scopo viene definito *rilevante*.

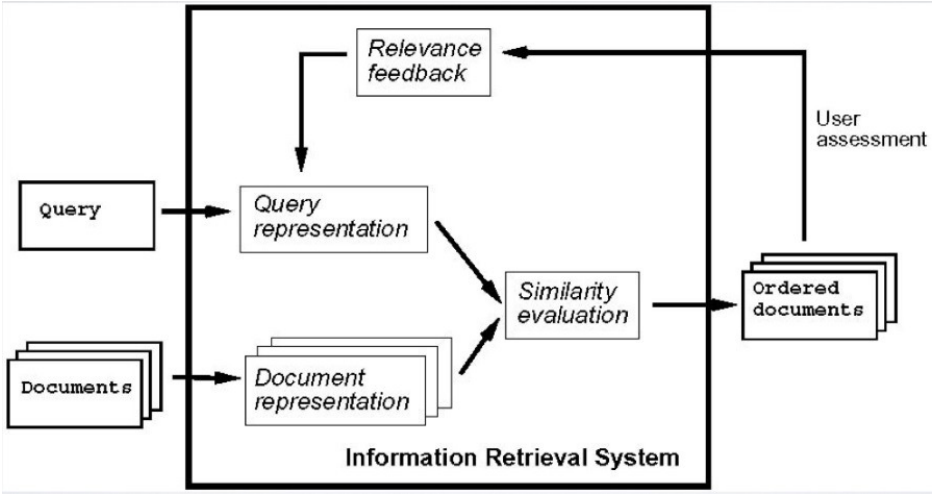


Figura 2.1. Schema di un sistema di reperimento dell'informazione

All'interno del sistema si trovano due elementi, *Query representation* e *Document representation*, che costituiscono la parte detta di *indicizzazione* che è l'operazione che servirà a rappresentare il contenuto informativo fornito in input in una forma che sia utilizzabile da un software apposito. La *Similarity evaluation* rappresenta invece la scelta di un modello che permetta di ottenere, dall'input elaborato con l'indicizzazione, una risposta all'esigenza informativa, ovvero un insieme di documenti estratti tra quelli inseriti nel sistema, insieme che potrebbe essere, a seconda delle richieste, ordinato o meno. Il *Relevance feedback* è quel componente che, in base a come verrà giudicata la risposta del sistema da parte dell'utente, cercherà di migliorarne l'efficacia, mediante l'applicazione di tecniche che permettano di affinare la comprensione e l'espressione dell'esigenza informativa.
Le fasi del reperimento possono quindi essere riassunte con acquisizione, indicizzazione, risposta. Un ulteriore aspetto da considerare è la valutazione del sistema, i cui risultati possono perfezionare tutti i componenti del sistema. Si sono sviluppate delle metriche che danno diverse indicazioni dell'efficacia del sistema e permettono quindi di quantificare la bontà del

lavoro di progettazione e realizzazione svolto. Si vuole sottolineare come la progettazione del sistema non vada vista come un insieme di passi successivi indipendenti ma vada eseguita considerando l'intero contesto del problema.

In questa tesi, l'implementazione di un sistema di IR viene denominata *motore di ricerca* (in inglese *search engine*). Va precisato che questa denominazione è utilizzata spesso per indicare la specifica applicazione di IRS che si ha nel web, che nel seguito verrà denominata *web search engine*.

Di seguito verranno descritte, le fasi per la progettazione di un sistema di reperimento dell'informazione sopra introdotte, partendo da come viene gestito l'input del sistema fino alla finale elaborazione dell'output.

2.2 Rappresentazione del contenuto informativo

La prima operazione da effettuare è quella di rappresentare il contenuto informativo degli oggetti digitali, estraendone le informazioni caratteristiche. I documenti di interesse trattati in questa tesi sono di tipo testuale e necessitano di una particolare fase di analisi tale da permetterne l'interpretazione da parte dello strumento di reperimento utilizzato, analisi che deve essere affidabile, consistente ed eseguita in modo efficiente, dato che in generale la si effettua su moli consistenti di dati[3]. L'operazione che realizza tale procedura è detta *indicizzazione*. Quando si sarà conclusa la fase di indicizzazione, si sarà preparato un elenco o indice di termini da utilizzare poi per il reperimento dei documenti.

Le fasi principali dell'indicizzazione automatica sono: analisi lessicale, rimozione delle *stop word*, *stemming* e composizione dei termini.

Analisi lessicale

Questa operazione consiste nell'analisi del documento al fine di estrarne i *token*, ovvero i termini presi singolarmente, in generale considerando come caratteri separatori gli spazi bianchi e gli elementi di punteggiatura.

[3]Uno studio del 2005 stimava solo per il Web la presenza di 11.5 miliardi di pagine. Fonte: *The indexable web is more than 11.5 billion pages* , A. Gulli, A. Signorini

Rimozione stop-word

Si procede con l'eliminazione dei *token* appartenenti alla categoria delle cosiddette *parole funzionali*[4], riferite nell'ambito dell'IR *stop word*. Per procedere con questa fase è necessario specificare un dizionario di parole da eliminare, tenendo conto che, più vocaboli non si considerano, minore sarà la dimensione dell'indice ma si incorrerà nel rischio di perdere contenuto informativo. Questo dizionario lo si può ricavare in maniera automatica analizzando la collezione di documenti in esame oppure si può utilizzare un dizionario, per la lingua dei documenti che si intendono gestire, resa disponibile a seguito di studi già effettuati.

Stemming

L'obiettivo di questa operazione è riportare diverse parole alla stessa radice linguistica (*stem*), in modo da diminuire la dimensione dell'indice, aumentando di fatto le prestazioni del sistema in termini di efficienza e spazio occupato, e favorendo l'espansione automatica delle interrogazioni. Vi sono due approcci generali a questa procedura, quello algoritmico e quello basato sul dizionario ed è possibile utilizzare entrambi in forme combinate.

Composizione dei termini

Questa fase richiede un'analisi specifica ed è particolarmente dipendente dal modello di IR che verrà utilizzato in seguito. Consiste nel comporre strutture composte da più *token* o *stem*, cercando di avvicinarsi a possibili input forniti dall'utente nell'interrogazione. Questo comporta però un'analisi abbastanza onerosa dal punto di vista del calcolo e incrementa la dimensione dell'indice.

[4] Parole che appartengono a classi chiuse (articoli, pronomi, preposizioni, congiunzioni) ma che non possono essere identificate con i morfemi grammaticali. Fonte: materiale del corso di Morfologia Linguistica e Glottologia, Jacopo Garzonio, 2012, Università degli Studi di Ferrara.

Creazione dell'indice

Si procede quindi con la creazione dell'indice trasposto o *inverted index*, chiamato così perché, nella sua forma più semplice, è una tabella costituita da record dove ad ogni *token* sono associati i documenti dove esso è presente. Altre informazioni si possono aggiungere, come la frequenza della parola o token nei singoli documenti, in modo da "pesarne" il contenuto informativo, ciò che viene riferito con il termine *Term Frequency* TF, ma molto dipende dal modello che si decide di utilizzare per il reperimento, come si vedrà nella sezione successiva.

Va precisato che le fasi di rimozione delle *stop word*, *stemming* e composizione dei termini possono essere effettuate o meno, ma in generale si segue l'ordine con cui sono state presentate. La decisione sull'eseguirle o meno va bilanciata con l'effettivo vantaggio che possono dare al prezzo del costo di esecuzione. È importante sottolineare come tutte le fasi appena elencate di estrazione del contenuto informativo siano fortemente dipendenti dalla lingua del documento, per esempio la lista delle stop-word cambia, o il concetto di radice linguistica può differire notevolmente.

Anche l'interrogazione sarà sottoposta ad un corrispondente processo di indicizzazione corrispondente a quello dei documenti e questo per avere la possibilità di eseguire un confronto.

2.3 Modelli di reperimento dell'informazione

Un modello di reperimento dell'informazione è un insieme di costrutti che sono formalizzati allo scopo di rendere possibile:

- La rappresentazione del contenuto dei documenti
- La rappresentazione delle interrogazioni
- L'ideazione, la progettazione e la realizzazione dell'algoritmo o degli algoritmi di reperimento dei documenti in risposta ad un'interrogazione

Il modello, in quanto tale, non fornisce i dettagli implementativi. Utilizzando la rappresentazione dei documenti e delle interrogazioni che si è

effettuata, la scelta del modello da utilizzare influisce sulle fasi già viste nella precedente sezione.

Nel tempo sono stati proposti diversi modelli, alcuni di essi sono:

- **Booleano**: si basa sull'uso degli operatori della logica booleana AND, OR, NOT in proposizioni logiche dove gli elementi sono detti *descrittori*[5]. Il reperimento è di tipo *exact-match* e non si ha un ordinamento dei documenti, almeno nella forma semplice del modello.
- **Probabilistico**: il modello stima l'incertezza e il costo della decisione del sistema nel reperire un dato documento. La funzione di reperimento è data dal rapporto tra la verosimiglianza dell'ipotesi che il documento sia rilevante e quella che non lo sia a seconda dei termini presenti nell'interrogazione.
- **Vettoriale**: questo modello considera i documenti e le interrogazioni come dei vettori costituiti dai pesi dei termini presenti nel documento e nella query. I documenti vengono recuperati calcolando la distanza tra il vettore della query e quello di ogni documento, utilizzando una misura di similarità (la più nota è la *cosine correlation*).

Molti altri modelli sono stati sviluppati nel tempo, sia con nuove concezioni sia come evoluzione di quelli appena elencati. Nel capitolo 4 sono descritti quelli utilizzati per il caso di studio di questa tesi.

2.4 La valutazione nel reperimento dell'informazione

La valutazione nel reperimento dell'informazione è un fattore molto importante ed è una misura di *efficacia*, ovvero serve a stabilire quanto il sistema risponde alle esigenze dell'utente fornendo i documenti che sono realmente di interesse in merito alla sue esigenza informativa, espressa attraverso la query. Per effettuarla è prassi consolidata, testare il sistema progettato

[5] Un descrittore t è l'insieme di tutti e soli i documenti in cui è presente il concetto espresso da t.

usando collezioni di test costruite per un dato *task*[6] e applicare poi delle metriche per avere un riscontro della bontà del sistema progettato. Tra queste le due più usate sono:

- *Recall*: rapporto tra il numero di documenti rilevanti recuperati e il totale dei documenti rilevanti della collezione.
- *Precision*: rapporto tra il numero di documenti rilevanti recuperati e il totale dei documenti recuperati.

Sono entrambe metriche normalizzate, con valori che vanno da 0 nel caso pessimo a 1 nel caso ottimale. Esistono poi molte altre metriche, di cui alcune prendono in esame la posizione dei documenti rilevanti restituiti, altre invece considerano diversi gradi di rilevanza, non solo quindi il caso binario[7].

2.5 Relevance feedback

Alla base del *relevance feedback* vi è l'assunto che non è semplice per l'utente esprimere la sua esigenza informativa ma egli è sicuramente in grado di riconoscere quando essa è soddisfatta o meno. Da qui si può sfruttare il suo comportamento per migliorare le prestazioni del sistema, al fine di incrementarne l'efficacia complessiva. Si parla di *Explicit Relevance Feedback* quando le informazioni da raccogliere vengono ricavate direttamente dall'utente mentre nel caso in cui si utilizzino modalità automatiche che analizzano il comportamento dello stesso in forma indiretta (ad esempio attraverso i click del mouse) si parla di *Implicit Relevance feedback*.

2.6 Enterprise search

Con il termine *enterprise search* si intende l'applicazione delle conoscenze e dei metodi del reperimento dell'informazione nel contesto della ricerca di

[6] Per task si indica sostanzialmente il compito della ricerca.

[7] Per una panoramica sulle principali metriche utilizzate nel settore del reperimento dell'informazione si consiglia la tesi magistrale *Una metrica di valutazione per l'Information Retrieval: Analisi Critica e sperimentazioni*, G. Demartini, 2005, Università degli studi di Udine.

informazioni all'interno di una organizzazione aziendale. La caratteristica peculiare è che la base documentale è costituita non da un'unica sorgente, ma da diverse sorgenti, quali ad esempio il web (spesso specificando delle *repository* scelte), l'intranet interna, e ogni altra fonte di testi digitale dell'organizzazione (messaggi di posta elettronica, database, log di eventi, ecc). Di conseguenza la base documentale può essere costituita da dati presenti sia in formati strutturati che non strutturati[8]. Da questi aspetti nascono diverse problematiche, le principali sono:

- poter ottenere in risposta all'utente una singola lista ordinata di documenti i quali però provengono da indici diversi e vengono recuperati anche mediante diversi metodi;
- l'aspettazione dell'utente è alta, in quanto deriva, in generale, dall'utilizzo di *web search engine*, che hanno raggiunto livelli accurati;
- non vi è un unico task di ricerca ma possono essercene molti diversi, ad esempio si può avere la necessità di recuperare un determinato documento ma anche di reperire dell'informazioni in merito ad un dipendente;
- l'accesso ai documenti non è indiscriminato ma è dipendente dalle autorizzazioni di chi utilizza il sistema all'interno dell'azienda;
- la valutazione del sistema è difficoltosa, in quanto le basi documentali sono private e differiscono notevolmente tra di loro.

Di contro l'organizzazione può ottenere diversi benefici dall'uso di un tale sistema, in quanto diversi studi [9] hanno evidenziato i costi derivanti dal tempo che si spreca per cercare *informazione* all'interno di una organizzazione.

[8] Per dati strutturati si intendono quelli che vengono conservati in database, seguendo regole e schemi prefissati, mentre i non strutturati non hanno uno schema e forma di conservazione ben definita.

[9] In particolare si fa riferimento a *The hight cost of not finding information*, S. Feldman, C. Sherma e *The Information Advantage: Information Accessin Tomorrow's Enterprise*, 2003, 2009, entrambi di S. Feldman, C. Sherman

Capitolo 3

Analisi di due search engine open source

Come già indicato nel capitolo 1 la scelta su quali *search engine* analizzare per la realizzazione del prototipo d'uso è ricaduta su Solr ed Elasticsearch. Ci sono diverse motivazioni sul perché questi due IRS sono stati presi in considerazione. Innanzitutto si è scelto di adottare prodotti già esistenti e di non progettare ex novo una nuova piattaforma, in quanto è pratica consolidata utilizzare sistemi con una solida base frutto di anni di lavori e miglioramenti. Tra questi in prima istanza si erano valutati due tra i più noti motori di ricerca ovvero Terrier[1] e Lucene[2]. Il primo è stato sviluppato all'Università di Glasgow ed è utilizzato principalmente in ambito accademico, il secondo invece fa parte dei progetti di *Apache Software Foundation*[3], ed è utilizzato in diversi sistemi commerciali. Essi però non presentano un elemento considerato strategico ai fini della realizzazione del prototipo ovvero l'applicazione dei principi delle architetture `REST`[4], che permettono

[1] `http://terrier.org/`

[2] `https://lucene.apache.org/core/`

[3] `https://www.apache.org/`

[4] L'acronimo sta per *Representational State Transfer* ed è stato coniato in *Architectural Styles and the Design of Network-based Software Architectures*, 2000, Fielding, R. Thomas.

l'interazione tramite semplici procedure di tipo `cURL`[5], tra il motore di ricerca e l'eventuale software già esistente.
Da una ricerca effettuata è emerso che i due motori di ricerca con queste caratteristiche più utilizzati al momento della stesura di questa tesi sono risultati essere Solr ed Elasticsearch[6]. Si è visto inoltre che entrambi si basano sulle librerie di Lucene per l'indicizzazione e la ricerca, hanno alle spalle una attiva comunità di sviluppatori e sono *open source*.

In questa tesi si è scelto di prendere in esame questi due sistemi e di confrontarli per capire quale potesse essere quello più indicato per la realizzazione del successivo prototipo. Si sono analizzati entrambi basandosi su due testi facenti parte della stessa casa editrice[7] e sulla documentazione ufficiale di ognuno. In aggiunta, per un confronto diretto, si sono analizzate diverse recensioni effettuate in ambiti diversi e reperite nel web. Si è scelto di presentarli separatamente, considerando, per entrambi, gli elementi che sono risultati essere i più importanti per l'obiettivo finale, fino all'ultima sezione dove si trarranno le conclusioni. Si è deciso inoltre di inserire, ove necessario per una migliore comprensione, degli elementi di codice, ma si precisa che questa tesi ha lo scopo di fornire un'idea di come essi siano strutturati e non è una prettamente una guida all'uso e alla configurazione, per questo scopo si rimanda alla citata documentazione[8].

3.1 Descrizione degli elementi considerati

Di seguito vengono descritti in modo sintetico gli elementi considerati importanti ai fini della scelta finale. Si sono analizzate le funzionalità utili per implementare un sistema di reperimento dell'informazione nella sua forma

[5] `https://curl.haxx.se/`

[6] Il sito `http://db-engines.com` si occupa di stilare delle classifiche in merito ai sistemi più utilizzati nell'ambito dei *database management system* (DMBS), con una traccia relativa anche al crescente settore dei sistemi NoSQL. Per la classifica inerente al caso in esame si veda la sezione relativa ai *search engine*.

[7] Precisamente i volumi *Solr in Action* e *Elasticsearch in Action*, si veda la Bibliografia per i dettagli.

[8] Si veda la precedente nota.

base descritta nel capitolo 2, in particolare come viene rappresentato il contenuto informativo dei documenti forniti in ingresso e i diversi modelli disponibili, considerando anche caratteristiche specifiche per il caso d'uso, ricavate dopo una fase di analisi dei requisiti con il committente, che hanno tenuto conto in particolar modo di come il risultato possa essere reso disponibile per una successiva presentazione all'utente finale.

Formato del documento ed eventuali crawler

Le collezioni di testi da analizzare sono composte da documenti con una struttura complessa, ad esempio PDF, Office Word, EPUB, ecc. Si è verificato come vengano gestiti i diversi casi e qual'è il formato che ogni motore richiede in ingresso per essere elaborato. Come si vedrà in seguito, una delle fonti di documenti è data dal web, si è quindi controllata la presenza o meno di *crawler* specifici inseriti all'interno, in particolar modo per le fonti *web*.

Indicizzazione

Vengono analizzate le modalità per eseguire la creazione dell'indice trasposto, le eventuali configurazioni possibili e personalizzazioni.

Modello di reperimento

Modelli di reperimento disponibili, tenendo conto anche di quello impostato di default all'atto dell'installazione.

Interrogazione

Si analizzano i formati disponibili per l'inserimento della query, i diversi *parser*[9] selezionabili ed ulteriori elementi di personalizzazione. Alcuni di questi elementi, più specifici, vengono presentati separatamente in seguito.

[9] Per *parser* si intende, in questa tesi, i diversi modi con cui avviene analizzata l'interrogazione testuale fornita in ingresso.

Re-ranking dell'interrogazione

Come anticipato, una fase importate in un sistema di reperimento dell'informazione è il relevance feedback. Si esplorano gli eventuali meccanismi inseriti all'interno delle due piattaforme che implementano, almeno in forma elementare, questa fase.

Le funzionalità che vengono elencate qui di seguito sono aggiuntive rispetto al sistema di reperimento dell'informazione base e vengono analizzate perché forniscono funzionalità di interesse.

Ricerca a faccette

Molto utilizzata nei siti web, in particolare in quelli commerciali, è un tipo di ricerca che combina la ricerca testuale sulla parte non strutturata del documento con un'analisi particolare sui metadati, rappresentanti una proprietà statica e descrittiva dell'oggetto che lo compongono, permettendo così un raffinamento dei risultati della ricerca[10].

Raggruppamento dei risultati

Concettualmente simile alla ricerca a faccette, in questo caso però non vi è una possibile riorganizzazione dei risultati della query che vengono invece raggruppati per categorie scelte o attraverso una categorizzazione automatica

Hit Highlighting

È una tecnica che permette di evidenziare la parte del documento recuperato dopo l'interrogazione. Questa funzionalità è molto utilizzata da molti *web search engine*.

Spell-checking

Strumento utile per dare un suggerimento all'utente dopo che ha inserito la query, cercando di dare un'interpretazione suggerendo eventuali correzioni

[10] Per maggiori informazioni si consiglia la tesi triennale *La Ricerca a Faccette. Aspetti teorici e pratici*, 2010, A. Marcato, Università degli Studi di Padova.

se vi sono presenti errori di scrittura delle parole e/o sintattici. Nel gergo della ricerca sul web questa funzionalità si ritrova spesso sotto la forma *Did you mean...?*

Query suggestion

Altra tecnica molto diffusa, aiuta l'utente nella ricerca dei documenti consigliando i termini durante l'inserimento dell'interrogazione, in modo da anticiparne la battitura.

Ricerca di tipo geospaziale

Si verifica se è supportato l'inserimento di campi nei documenti che siano portatori di informazione di tipo spaziale, quali ad esempio delle coordinate geografiche, o che siano direttamente rappresentabili geometricamente con linee e forme, tali da permettere interrogazioni con raggruppamenti, intersezioni, ecc.

Supporto al multilinguismo

L'analisi di collezioni di documenti aventi lingue differenti è un problema di non facile risoluzione, qui si è verificata la presenza di funzionalità che permettano la creazione di indici trasposti diversi per lingue diverse, e se vi siano presenti sistemi per l'auto riconoscimento dell'idioma.

Distribuzione dell'indice e delle interrogazioni

Si è valutata la possibilità di distribuire l'indice, e di conseguenza la ricerca, su più calcolatori, in modo da suddividere il carico di lavoro.

Monitoraggio

Tenere sotto controllo le attività che si effettuano progressivamente può essere un utile strumento per analizzare lo stato del sistema, rilevando eventuali problemi o malfunzionamenti.

3.2 Solr

Solr nasce nel 2004 da Yonik Seeley, nel 2006 diviene pubblico e completamente open source sotto licenza *Apache Software Foundation* mentre nel 2010 si fonde col progetto Lucene; la release qui considerata è la 6.0.0, rilasciata ad aprile 2016.

Figura 3.1. Logo Solr

3.2.1 Formato del documento ed eventuali crawler

L'unità base nel sistema è il *documento*, il quale è visto come un insieme di dati, ognuno portatore di una specifica informazione.

I documenti in Solr sono composti da metadati (*field*) i quali rappresentano il tipo di informazione ad essi associati, grazie alla definizione dei *field type*, dove è possibile dichiararne le caratteristiche, e come esse andranno poi utilizzate ed interpretate dal sistema, in merito alla loro indicizzazione, ricerca ed analisi[11]. Il file a cui fa capo Solr per la definizione di tali elementi è `schema.xml`[12].

Un *field type* è costituito dai seguenti elementi:

- un nome indicativo
- il nome della classe rappresentata

[11] Solr permette una maggiore flessibilità grazie alla dichiarazione di *Dynamic Field*, ovvero elementi che non sono stati dichiarati in maniera esplicita nello schema.

[12] Nella versione considerata di Solr tale file non è direttamente presente nelle cartelle di riferimento dell'indice. Per recuperarlo si consiglia di utilizzare le procedure indicate su https://cwiki.apache.org/confluence/display/solr/Schema+Factory+Definition+in+SolrConfig

- proprietà caratteristiche dipendenti dal tipo di classe e dalle scelte di intervento in fase di indicizzazione ed interrogazione in merito al tipo di dato

Solr supporta i formati XML, JSON, CSV. Per convertire un documento presente in altri formati, si appoggia al framework *Solr Cell* costruito su *Apache Tika*[13], che permette di convertire file strutturati complessi quali PDF, Office Word, EPUB, ecc[14] , in un *documento* con le caratteristiche indicate nel modo descritto nella sezione precedente.
Non vi è un *crawler web* specifico ma vengono suggeriti diversi progetti a cui Solr si appoggia[15].
Per l'importazione di dati strutturati provenienti da basi di dati è presente un apposito meccanismo denominato *Data Import Handler*.

Schemaless

Come scritto Solr si basa sulla definizione di campi `<fyeldType>` per la mappatura dei documenti da indicizzare. E' possibile delegare questa procedura in modo da far sì che il sistema gestisca in maniera autonoma il riconoscimento e le scelte per l'indicizzazione

3.2.2 Indicizzazione

La creazione dell'*inverted index* può essere schematizzata attraverso le seguenti tre fasi:

- Conversione del documento dal suo formato nativo ad uno supportato da Solr, nello specifico XML, JSON, CSV.

- Configurazione delle impostazioni per l'analisi dei documenti che verranno inviati al fine di creare l'indice.

[13] https://tika.apache.org/

[14] Per la lista dei formati supportati si veda https://tika.apache.org/1.12/formats.html#Supported_Document_Formats

[15] Si veda la sezione *Crawlers And Connectors* all'indirizzo https://wiki.apache.org/solr/SolrEcosystem

- Aggiunta del documento convertito al sistema mediante una delle interfacce definite.

La creazione dell'indice consta delle quattro macro operazioni che vengono eseguite in sequenza: analisi, rimozione stop-word, stemming, composizione dei termini. Queste operazioni possono essere realizzate o no per ogni `fieldType`.

Nel caso specifico di esempio si considera un `fieldType` denominato *nametext*. All'interno vi sono specificati diversi filtri, mediante il tag `<filter>`, che rappresentano molteplici interventi che è possibile applicare sequenzialmente.

Per l'operazione di recupero dei *token*, lo standard presente, definito da `solr.StandardTokenizerFactory`, utilizza come separatori gli spazi bianchi e la punteggiatura ma molti altri ne possono essere specificati, tenendo conto del modello di reperimento che verrà utilizzato per il successivo recupero dei documenti.

Per la rimozione delle stop word il filtro è specificato dalla classe `solr.StopFilterFactory`, ed il file utilizzato è `stopword.txt`, che va modificato a seconda della lingua utilizzata (di default quello presente è relativo alla lingua inglese[16]). Da notare come sia presente un filtro meno aggressivo detto *Suggest Stop Filter* che va a rimuovere i *token* solamente se non sono seguiti da un *token* separatore.

Per l'operazione di stemming sono presenti due diversi *stemmer* per la l'italiano, il primo è definito da `solr.SnowballPorterFilterFactory`[17] mentre il secondo su `solr.ItalianLightStemFilterFactory`.

```
<fieldType name="nametext" class="solr.TextField">
  <analyzer>
    <tokenizer class="solr.StandardTokenizerFactory"/>
    <filter class="solr.StopFilterFactory"
```

[16] https://wiki.apache.org/solr/AnalyzersTokenizersTokenFilters#solr.StopFilterFactory

[17] Per maggiori indicazioni sul funzionamento: http://snowball.tartarus.org/algorithms/italian/stemmer.html

```
      words="stopwords.txt" />
      <filter class="solr.SnowballPorterFilterFactory"
      language="Italian" />
      ...
    </analyzer>
</fieldType>
```

Listing 3.1. Esempio di configurazione del metadato `nametext` in Solr

L'indice trasposto può essere così creato, essendo nella sua forma basilare composto da record aventi per ogni *token* i relativi documenti dove sono presenti. E' possibile specificare altre informazioni, quali ad esempio la frequenza del *token* in ogni singolo documento, la sua posizione.

3.2.3 Scelta del modello di reperimento

Per la scelta del modello di reperimento da utilizzare si agisce similmente a quanto visto per le operazioni di indicizzazione specificando nei vari `fieldType` il modello scelto attraverso la definizione del tag `similarity`.

```
<similarity class="solr.ClassicSimilarityFactory" />
```

Listing 3.2. Configurazione per la scelta del modello vettoriale in Solr

Essendo Solr e Lucene integrati nello stesso progetto, si possono utilizzare tutti i modelli di reperimento definiti su quest'ultimo[18]. Dalla versione considerata il modello impostato automaticamente è quello probabilistico, nella sua specifica BM25.

3.2.4 Interrogazione

Per eseguire l'interrogazione si deve inviare una richiesta al *Request handlers*, un servizio dotato di numerose funzionalità, tra cui il *SearchHandler*, che è quello di default adibito all'esecuzione delle interrogazioni. Quest'ultimo permette diverse scelte per l'interpretazione della query attraverso il

[18] https://lucene.apache.org/core/6_0_0/core/org/apache/lucene/search/similarities/package-tree.html

Query parser, strumento utilizzato per stabilire la modalità con cui la ricerca sarà effettuata.
Tra i diversi parser disponibili quello di default è il *Lucene query parser*, ma sono disponibili anche i più flessibili *DisMax Query Parser*, *eDisMax Query Parser* e molti altri, ed è possibile crearne di personalizzati.

```
http://localhost:8983/solr/Index/select?q=*:*&wt=json
```

Listing 3.3. Esempio di semplice interrogazione in Solr

La risposta, nella sua forma base, sarà fornita nel formato richiesto tra quelli disponibili (XML, JSON, PHP, CSV, Ruby, Phyton), e con i *field* richiesti, con i risultati ordinati secondo il ranking fornito dal modello impostato.

E' possibile inoltre utilizzare l'interfaccia utente definita da *Velocity Response Writer* (conosciuta anche come *Solritas*), che permette di eseguire interrogazioni direttamente da una barra di testo attraverso un browser.

3.2.5 Ricerca a faccette

In Solr questa tecnica è presente ed è possibile sfruttarne numerose funzionalità avanzate. Per impostarne l'utilizzo sarà necessario inserire, nella dichiarazione dell'interrogazione, la specifica `facet=true`, specificando poi quali campi si vogliono avere in risposta con tale organizzazione, attraverso la definizione del campo `faced.field`. La risposta di default conterrà la lista dei termini recuperati con la loro frequenza nell'indice. Tra le specifiche si segnala `facet.sort`, che permette di scegliere tra l'ordinamento lessicografico e quello per frequenza, e `facet.prefix` che fornisce in risposta solamente quei termini aventi un prefisso impostato. Un esempio di interrogazione:

```
http://localhost:8983/solr/query?q=bar&facet=true
&facet.field=notturno
```

Listing 3.4. Esempio di query in Solr con specifiche per l'utilizzo della ricerca a faccette

3.2.6 Raggruppamento dei risultati

La funzione di raggruppamento riunisce i documenti con un elemento comune in gruppi, restituendo poi per ogni gruppo quelli più rilevanti. Per quanto possa sembrare similare alla ricerca a faccette in questo caso non c'è un raffinamento della ricerca a seconda della categoria. Tra i principali parametri che possono essere impostati vi è l'opzione di attivazione attraverso `group`, e due metodologie per l'intervento: `group.field`, ove si specifica un campo, `group.query`, dove si specificano dei parametri per un dato campo.

Per gli algoritmi di clustering, all'atto della stesura di questa tesi, per la versione *offline*[19] è stata definita un'interfaccia non ancora implementata mentre quella *online* si appoggia al progetto *Carrot*[20].

3.2.7 Hit Highlighting

Ci sono tre implementazioni disponibili:

- *Standard*: molto personalizzabile, non richiede delle strutture specifiche per l'indice ed è utilizzata in casi di situazioni che non presentano condizioni particolari per cui sia più utile utilizzare uno degli altri disponibili.

- *Fast Vector*: tecnica ottimizzata su Term Vector, in quanto richiede che siano specificati, per ogni campo che si intende utilizzare per l'evidenziazione, i paramenti `termVectors`, `termPositions`, e `termOffsets`. Questa implementazione è consigliata per documenti molto grandi.

- *Posting*: è il meno flessibile ma utilizza uno spazio di memoria inferiore rispetto ai precedenti in quanto non richiede aggiunta di elementi all'indice, quindi risulta essere il più efficiente.

[19] Per *offline* si intende che l'algoritmo viene applicato in fase di indicizzazione mentre *online* dopo che è stata eseguita l'interrogazione.

[20] `http://project.carrot2.org/`

3.2.8 Spell-checking

In Solr il componente adibito alla gestione di questa funzionalità è lo *SpellCheck*. Ci sono tre diversi approcci utilizzabili:

- *IndexBasedSpellChecker*, utilizza un indice parallelo a quello di ricerca.
- *DirectSolrSpellChecker*, impostato come predefinito, non utilizza alcun indice parallelo ma sfrutta quello utilizzato per la ricerca standard.
- *FileBasedSpellChecker*, utilizza un file esterno come dizionario per il controllo ortografico.

Esiste un ulteriore approccio, denominato *WordBreakSolrSpellChecker*, che non utilizza alcun indice o file, ma fornisce il suggerimento combinando l'adiacenza dei vocaboli dell'interrogazione ed eventualmente la separazione dei termini in più parole.

Tra i parametri configurabili nell'inserimento della query si trova `spellcheck` che va selezionato al valore `true` per utilizzarlo, `spellcheck.count` che specifica quanti suggerimenti fornire e `spellcheck.dictionary` ove si sceglie quale dizionario utilizzare.

```
http://localhost:8983/solr/techproducts/spell?df=text&
spellcheck.q=dell+ultra+sharp&spellcheck=true&
spellcheck.collateParam.q.op=AND
```

Listing 3.5. Esempio di query con SpellCheck in Solr

3.2.9 Query Suggestion

SuggestComponent è lo strumento apposito per questa funzionalità. Le principali funzionalità sono:

- L'implementazione da utilizzare, specificando il parametro `lookupImpl`. Di default si utilizza il `JaspellLookupFactory`[21]
- Il dizionario utilizzabile, specificando il campo `dictionaryImpl`.

[21] Per maggiori dettagli sul funzionamento si veda http://jaspell.sourceforge.net/

Un esempio di configurazione è il seguente

```
<searchComponent name="suggest"
class="solr.SuggestComponent">
<lst name="suggester">
<str name="name">mySuggester</str>
<str name="lookupImpl">FuzzyLookupFactory</str>
<str name="dictionaryImpl">
DocumentDictionaryFactory</str>
<str name="field">cat</str>
<str name="weightField">price</str>
<str name="suggestAnalyzerFieldType">string</str>
<str name="buildOnStartup">false</str>
</lst>
</searchComponent>
```

Listing 3.6. Esempio di configurazione per la *query suggestion* in Solr

3.2.10 Re-ranking dell'interrogazione

Solr applica la seguente metodologia: i primi top N risultati di una interrogazione (A) vengono utilizzati come collezione per una ulteriore query più complessa (B), i cui risultati vanno ad influire nei risultati della query di partenza (A).

Il ricalcolo dei risultati di un'interrogazione può essere richiesto utilizzando il parametro `rq`, specificando per parser `rerank` i seguenti tre parametri:

- `reRankQuery`: la stringa utilizzata per il ricalcolo.
- `reRankDocs`: il numero N dei documenti ordinati su cui applicare il ricalcolo.
- `reRankWeight`: un fattore moltiplicativo che sarà applicato ai documenti restituiti dalla sotto interrogazione con la stringa impostata per il ricalcolo.

```
q=greetings&rq={!rerank reRankQuery=$rqq
reRankDocs=1000 reRankWeight=3}&rqq=(hi+hello+hey)
```

Listing 3.7. Esempio di query con re-ranking in Solr

3.2.11 Ricerca di tipo geospaziale

Solr supporta questo tipo di ricerca, nello specifico permette di:

- indicizzare punti e altre forme
- filtrare i risultati di una interrogazione secondo contorni geometrici
- ordinare i risultati seguendo schemi geometrici
- generare una griglia visuale con i risultati

Ci sono tre *type* principali disponibili per le ricerche spaziali:

- `LatLonType` e `PointType`, indicati quando è richiesto in particolare l'ordinamento efficiente secondo la distanza.
- `SpatialRecursivePrefixTreeFieldType`, abbreviato con `RPT`, fornisce maggiori funzionalità rispetto al precedente, indicato per ricerche più specifiche.
- `BBoxField`, per l'indicizzazione e la ricerca per forme geometriche proprie.

3.2.12 Ricerche multilingua

Come precedentemente specificato la ricerca su documenti di lingue differenti è un argomento molto complesso. Solr mette a disposizione diversi *stemmer* specifici per lingua, un filtro apposito per l'analisi dei sinonimi (`solr.SynonymFilterFactory`), e due metodi per l'individuazione automatica della lingua[22]:

- *Tika's language detection*
- *LangDetect language detection*

[22] E' possibile vedere una comparazione tra i due su http://blog.mikemccandless.com/2011/10/accuracy-and-performance-of-googles.html

3.2.13 Distribuzione dell'indice e delle interrogazioni

Per quanto riguarda l'allocazione in memoria dell'indice Solr mantiene la visione a *segmenti* definita in Lucene quindi, quando un nuovo insieme di documenti è aggiunto, viene scritto in un nuovo segmento, con la conseguenza che l'indice sarà ripartito tra tutti i segmenti creati. È possibile intervenire sulla gestione di questa politica, forzando l'unione di più segmenti o altre operazioni che vanno ad influire direttamente sulla gestione della memoria allocata.

Per migliorare l'efficienza del sistema e per aumentare la tolleranza ai guasti si ricorre a tecniche di calcolo distribuito e in questo interviene *SolrCloud*. L'indice, definito *collezione*, viene partizionato e suddiviso in elementi, detti *shard*, e allocato in diversi server detti *nodi*, replicandolo per prevenire eventuali perdite di informazioni dovute a malfunzionamenti e per migliorare la scalabilità del sistema. La gestione dei nodi in merito a bilanciamento e malfunzionamenti si appoggia al servizio *Apache Zookeeper*[23]. *SolrCloud* si occupa anche di distribuire la ricerca, in particolare nella fase di generazione dei risultati ottenuti combinando le risposte provenienti dai vari nodi.

3.2.14 Monitoraggio

L'interfaccia web presente di default garantisce un buon livello di monitoraggio, per avere un quadro più dettagliato vi è la versione commerciale Solr *Monitoring* di Semantext[24].

3.3 Elasticsearch

Elasticsearch nasce nel Febbraio 2010 da Shay Banon, creatore del progetto Compass[25], che decise di creare ex novo un sistema di ricerca scalabile e facilmente interfacciabile con i principali linguaggi di programmazione.

[23] http://zookeeper.apache.org/

[24] https://sematext.com/spm/integrations/solr-monitoring/

[25] Per maggiori informazioni sul progetto: https://en.wikipedia.org/wiki/Compass_Project.

E' distribuito sotto licenza *Apache Software Foundation*. La release qui considerata è la 2.3, rilasciata ad Aprile 2016.

Figura 3.2. Logo Elasticsearch

3.3.1 Formato del documento ed eventuali crawler

Similmente a Solr anche in Elasticsearch l'unità base dell'informazione è il *documento*, rappresentato in JSON, attraverso la suddivisione in campi (*field*). Per i formati strutturati complessi non ha una meccanismo specifico ma si appoggia anch'esso ad *Apache Tika*[26]. Non è presente un file di configurazione corrispettivo a `schema.xml` ma è possibile specificare diverse proprietà per un campo attraverso la definizione di elementi su `"mappings"`:

```
PUT my_index
{
  "mappings": {
    "user": {
      "_all":       { "enabled": false  },
      "properties": {
        "title":    { "type": "string"  },
        "name":     { "type": "string"  },
        "age":      { "type": "integer" }
      }
    }
```

[26] Si vedano le note 13 e 14.

```
}
```

Listing 3.8. Esempio di *mapping* per il campo *user* su Elasticsearch

Non è presente un *web crawler* predefinito ma si consiglia il plugin *Elasticsearch River Web*[27], anche per l'importazione di dati da un DMBS ci si deve appoggiare a plugin appositi.

3.3.2 Indicizzazione

La procedura base di indicizzazione è molto semplice e può essere fatta dinamicamente, ovvero con l'inserimento del documento. Dal punto di vista dei contenuti l'indice è sempre similare a quello di Lucene, mentre strutturalmente è distribuito, in seguito si descriverà più dettagliatamente come viene realizzato.

L'operazione di analisi recupera i *token* attraverso l'*Unicode Text Segmentation*[28] secondo l'impostazione di default, ma permette di specificare diverse altre tecniche, usualmente tenendo conto del sistema di reperimento che verrà utilizzato.

La rimozione delle *stop-word* richiede l'impostazione della lingua inerente al documento (ve ne sono una lista di disponibili, tra cui l'italiano) o, in alternativa, è possibile inserirne di proprie direttamente nelle specifiche, senza dover scrivere un file apposito.

Per lo stemming vi sono a disposizione tre diversi algoritmi: *snowball filter*, *porter stem filter* e *kstem filter*, quest'ultimo viene consigliato per la lingua inglese. È possibile altresì utilizzare lo *Hunspell Token Filter*, combinato con un dizionario per la lingua scelta.

L'indice creato è di tipo *inverted index* e può essere arricchito con ulteriori informazioni, similmente a Solr.

```
% curl -XPUT 'localhost:9200/get-together/_mapping/
   new-events' -d '{
    "new-events" : {
```

[27] https://github.com/codelibs/elasticsearch-river-web.

[28] http://unicode.org/reports/tr29/

```
        "properties" : {
           "host": {
              "type" : "string"
            }
          }
        }
    }'
```

Listing 3.9. Mappatura di un campo su Elasticsearch

3.3.3 Scelta del modello di reperimento

Il modello utilizzato di default per il calcolo del ranking è una versione del vettoriale basato su TF/IDF. Per modificarlo si deve intervenire sul parametro `mappings` dell'indice alla voce `similarity`, nell'esempio viene impostato il BM25:

```
{
  "mappings": {
    "get-together": {
      "properties": {
        "title": {
          "type": "string",
          "similarity": "BM25"
        }
      }
    }
  }
}
```

Listing 3.10. Scelta del modello BM25 sull'indice *get-together* per *title* su Elasticsearch

Si può intervenire sui parametri del modello scelto ed inserire altri filtri per personalizzare il reperimento. I modelli a disposizione sono:

- Okapi BM25
- Divergence from randomness, or DFR similarity

- Information based, or IB similarity
- LM Dirichlet similarity
- LM Jelinek Mercer similarity

3.3.4 Interrogazione

Per l'invio al sistema della query si utilizzerà una chiamata `REST URI` e la risposta sarà in JSON. In merito alla sintassi vi sono diversi parametri che possono essere specificati (dal tipo di analizzatore che si vuole applicare alla stringa in ingresso alla taglia della risposta), comunque il principale è `q="<testo>"` ove si immette la stringa da ricercare.

```
curl 'localhost:9200/bank/_search?q=*&hello world'
```

Listing 3.11. Esempio di ricerca della stringa "hello word" su Elasticsearch

È possibile utilizzare l'interfaccia web *Kopf*[29], dove è presente una sezione per l'inserimento di comandi `REST`, considerando che non è immediatamente disponibile ma va scaricato l'apposito plugin.

3.3.5 Ricerca a faccette

In Elasticsearch non è prevista la ricerca a faccette[30] come su Solr ma viene definita una nuova metodologia detta *Aggregazione*. La differenza principale è che permette l'annidamento dei risultati, quindi fornisce un livello maggiore di profondità e dettaglio da ciò che viene fornito dalla ricerca. Esistono tre categorie principali:

- *Bucketing*, il principio è proprio quello di valutare quale sia il *bucket* più rilevante, secondo un criterio stabilito, in cui inserire un documento reperito. Sono possibili diverse strategie per la scelta del criterio per l'inserimento.

[29] https://github.com/lmenezes/elasticsearch-kopf

[30] Era prevista fino alle versioni 1.x

- Metriche, permettono di calcolare delle statistiche su un insieme di documenti, specificatamente ad un dato campo numerico presente e selezionato.

- *Pipeline*, lavorano specificatamente sull'output prodotto da una precedente aggregazione e non direttamente sui documenti recuperati. Questa tipologia è, alla data della tesi, in via sperimentale.

3.3.6 Raggruppamento dei risultati

Per ottenere un raggruppamento dei risultati si deve utilizzare lo stesso il meccanismo delle aggregazioni visto nella precedente sezione. Per far questo si deve specificare il campo `top_hits`, inserendo il filtro per il raggruppamento. Per gli algoritmi di *clustering* ci si basa sul plugin apposito del progetto Carrot[31].

3.3.7 Hit Highlighting

Le implementazioni per utilizzare tale funzionalità sono:

- *Lucene Plain highlighter*, impostato di *default*, valido nella gran parte dei casi.

- *Fast vector highlighter*, consigliato nel caso si imposti nella mappatura l'aggiunta di informazioni inerenti a `term vector` impostando in particolare `with positions offsets`, e in casi in cui i campi considerati siano di dimensioni superiori a 1 MB.

- *Postings highlighter*, il più veloce, non richiede l'aggiunta di alcun elemento nell'indice come il precedente metodo anche se si applica se `index options` e impostato come `offset` nella mappatura.

3.3.8 Spell-checking

Ci sono due diverse macro implementazioni:

[31] https://github.com/carrot2/elasticsearch-carrot2

- *Term suggester*, più veloce e basilare, utile per ricerche basate su parole chiave, in particolare su testi brevi. Si basa sulla distanza della parola editata con quelle presenti nell'indice
- *Phrase suggester*, più vicino al linguaggio naturale dell'utente, richiede però una maggiore complessità di calcolo. È anch'esso basato sullla valutazione di una distanza considerando però la frase intera, basandosi su un modello impostato sugli *n-gram*.

Entrambi i metodi si appoggiano al modulo `SpellChecker` di Lucene.

3.3.9 Query suggestion

La funzionalità di auto-completamento, denominato *Prefix suggestions* si basa sul modulo *Lucene Suggest*. E' presente un'estensione, il *Contest Suggester*, che permette di filtrare la ricerca precisando una determinata categoria (termine) o locazione geografica, inseriti in fase di indicizzazione.

3.3.10 Re-ranking dell'interrogazione

E' presente una funzionalità, chiamata *Rescoring*, che permette il ricalcolo del ranking sui primi N documenti ottenuti in risposta, basandosi su una query impostata. E' possibile specificare una funzione di ricalcolo, intervendo in maniera più specifica e personalizzata, attraverso delle `function_score` , ove è possibile definire ulteriori parametri per un raffinamento e personalizzazione dell'ordinamento ottenuto in risposta ad un'interrogazione.

3.3.11 Ricerche di tipo geospaziale

Elasticsearch è adibito alla ricerca geospaziale, integrando le principali funzionalità descritte nella lista introduttiva. I *type* presenti sono `geo_point` e `geo_shape`. Nel Listato 3.12 un esempio di interrogazione dove il risultato viene filtrato secondo una distanza geografica

```
% curl 'localhost:9200/get-together/event/_search?
   pretty' -d '{
    "query": {
      "filtered": {
```

```
        "filter": {
            "geo_distance": {
                "distance": "50km",
                "location.geolocation": "40.0,-105.
                    0"
                }
            }
        }
    }
}'
```

Listing 3.12. Esempio di ricerca con filtro geospaziale in Elasticsearch

3.3.12 Ricerche multilingua

Sono presenti vari analizzatori di testo per molte lingue, italiano incluso, che permettono un'analisi specifica in fase di indicizzazione. Nel caso di documenti aventi idiomi differenti il consiglio fornito è quello di mantenere indici separati e in fase di interrogazione restituire i documenti per lingua richiesta. Per l'identificazione automatica della lingua non vi sono funzionalità interne ma il sistema si appoggia a librerie esterne, in particolar modo viene consigliata la `chromium-compact-language-detector`[32], che utilizza *Compact Language Detector*[33].

3.3.13 Distribuzione dell'indice

L'allocazione in memoria segue l'impostazione di Lucene ma distribuisce l'indice in un modo peculiare. Innanzitutto quando si avvia un'istanza di Elasticsearch si crea un *nodo*. Insiemi di nodi vengono raggruppati in *cluster*. L'indice viene suddiviso in pezzi detti *shard*; vi è il *primary shard* e tutta una serie di copie, dette *replica shard*, distribuite in nodi diversi.

[32] https://github.com/mikemccand/chromium-compact-language-detector

[33] https://github.com/CLD2Owners/cld2

Questa implementazione favorisce la scalabilità orizzontale[34] e la suddivisione distribuita e parallela delle operazioni, aumentando così l'efficienza e la tolleranza ai guasti. Per quanto è possibile intervenire nelle scelte che regolano tale meccanismo, tutto può essere reso completamente trasparente. Le ricerche vengono distribuite tra i vari nodi, al fine di minimizzare la velocità di risposta.

3.3.14 Monitoraggio

Ci sono diversi plugin adibiti al monitoraggio in Elasticsearch, tra i più noti si evidenzia *Marvel*[35], che permette una profonda analisi della distribuzione dei cluster, l'andamento storico di ciò che è stato eseguito e un profondo controllo sulla memoria allocata ed utilizzata. Si precisa che è un prodotto commerciale, con licenza libera per un periodo di prova. Altri plugin sono il già citato *Kompf*, *Sematext SPM* e *ElasticHQ*.

3.4 Comparazione

Da quanto analizzato e studiato si può dichiarare che Solr ed Elasticsearch sono due prodotti molto simili, almeno per quanto riguarda ciò che è risultato utile ai fini della comparazione. Gli elementi che sono emersi dallo studio sono:

- Per quanto riguarda gli aspetti più prettamente legati al reperimento dell'informazione, essendo entrambi basati su Lucene, le operazioni basilari di indicizzazione, scelta del modello e ricerca sono garantite nei modi richiesti e non presentano differenze sostanziali. Eventualmente si può avere una preferenza per Solr in quanto recepisce prima le novità che possono emergere dagli sviluppi fatti su Lucene, essendo di fatto lo stesso progetto.

[34] Si ricorda che il numero massimo di documenti che possono essere contenuti in un indice di Lucene è pari a 2.147.483.519 [si veda https://issues.apache.org/jira/browse/LUCENE-5843].

[35] https://www.elastic.co/products/marvel

- Entrambi dispongono di funzionalità *schemaless* che permettono un utilizzo semplificato, anche se è comunque consigliabile un'analisi dei documenti da indicizzare e una preconfigurazione per un utilizzo maggiormente consapevole.
- Per quanto riguarda la gestione dell'indice in termini di memoria ed allocazione di quest'ultima, Elasticsearch presenta un vantaggio dovuto alla sua natura, che di fatto lo rende predisposto ad una ottimizzazione da questo punto di vista. Però Solr, col sistema *SolrCloud*, sembra aver colmato la differenza con il software Elasticsearch, proponendo una soluzione adeguata per questo tipo di problema.
- Elasticsearch punta molto sulla funzionalità delle *Aggregazioni* cercando di fornire una evoluzione rispetto alla classica ricerca a faccette, per quanto sia una caratteristica utile solo in determinate circostanze.
- Entrambi i sistemi forniscono la possibilità di inserire facilmente dei plugin realizzati da utenti per aggiungere nuove funzionalità.
- Elasticsearch è distribuito su licenza *open source* ma per quanto riguarda la possibilità di contribuire al rilascio di nuove versioni questo è riservato solamente a chi fa parte alla compagnia associata[36]. Solr invece è aperto per tutta la comunità di sviluppatori e chiunque può contribuire allo sviluppo delle nuove versioni.

Per quanto riguarda questa tesi, viste le considerazioni fatte, si è scelto di utilizzare Solr, anche a fronte di un precedente e soddisfacente utilizzo da parte dell'azienda committente.

[36]La compagnia in question è Elastic https://www.elastic.co/

Capitolo 4

Realizzazione del prototipo

In questo capitolo viene presentato il prototipo di sistema realizzato per soddisfare la richiesta da parte dell'azienda committente. La struttura seguita è coerente con lo schema di un sistema di reperimento dell'informazione presentato nel capitolo 2; vengono fornite inoltre delle informazioni aggiuntive per presentare la specificità del caso in esame. Infatti, dopo una prima sezione dove si descrive il problema, le successive sono strutturate riprendendo gli elementi fondamentali descritti in precedenza: il formato dei documenti, la modalità di indicizzazione, i modelli di reperimento considerati e le interrogazioni. Il processo di creazione della collezione di test, atta a realizzare una valutazione dell'efficacia di quanto realizzato, è presentata nel capitolo 5. Dopo l'analisi condotta nel capitolo 3, si è utilizzato, come *search engine* Solr, e similmente a quanto fatto nel medesimo capitolo, si sono inseriti solo alcuni elementi di codice ove ritenuto necessario per una maggiore chiarezza di quanto esposto. Essendo alcune informazioni di natura sensibile, si è scelto di presentare certi elementi in maniera generale, senza esporre informazioni riservate di proprietà dell'azienda.

4.1 Descrizione del problema

Come già accennato, questo lavoro di tesi è effettuato in collaborazione con l'azienda Siav e si inserisce all'interno del lavoro di ricerca e sviluppo che

ruota attorno ad Archiflow[1], la soluzione integrata di Siav per l'*Enterprise Document Management*. Un *documento* in Archiflow è costituito dall'unità *scheda documento*, che raccoglie un insieme di *metadati* e alla quale si aggiungono il *documento principale* ed eventuali *documenti allegati*.
L'obiettivo in questa tesi è quello di realizzare un prototipo che permetta il reperimento efficace a partire da questi elementi, costituiti sia dai metadati che dal contenuto testuale, con la possibilità di integrare anche altri argomenti provenienti da sorgenti differenti, in modo da fornire all'utente finale una panoramica di informazioni più ampia. La motivazione di quest'ultima scelta sta nel fatto di voler non solo soddisfare l'esigenza informativa che l'utente esprime, ma di fornirgli ulteriori informazioni che possono essergli utili ai fini di una ricerca in un contesto aziendale. Un esempio, che chiarisce questa finalità, è quello di una ricerca che richiede dei documenti relativi ad un determinato partner: oltre a fornire tali documenti, sempre mantenendo come metrica l'efficacia, si forniscono ulteriori informazioni inerenti a quel collaboratore, sia provenienti dalla base documentale, ma anche da notizie dal web, ad esempio dalla pagina web ufficiale proprio di quel cliente, o da altre sorgenti. Questo permette di anticipare eventuali esigenze informative, ottenendo così un vantaggio in termini di tempo, e soprattutto un profilo conoscitivo potenziato in un ottica di *business intelligence*.
Tutte le scelte in seguito fatte sono conseguenza di una fase di analisi dei requisiti realizzata in collaborazione con l'azienda committente.

Nel caso in esame si è stabilito che l'utente di riferimento sarà proprio l'azienda, che ha fornito un sottoinsieme della propria base documentale gestita dal software citato, oltre un insieme di dati descritto in seguito. Va inoltre precisato che, come sarà dettagliatamente descritto nel capitolo conclusivo, alcune scelte sono state fatte tenendo conto di eventuali futuri sviluppi del sistema, e si è tenuto presente che la soluzione dovrà adattarsi alle diverse esigenze delle organizzazioni che utilizzano al loro interno il prodotto Archiflow.

[1] https://www.siav.com/it/software-di-gestione-documentale/archiflow/

4.2 Formato dei documenti

Le diverse sorgenti di documenti che si sono utilizzate sono:

- Documenti provenienti da Archiflow. I metadati relativi alle scheda documento sono salvati in un file `CSV` mentre i documenti e gli allegati trattati sono presenti in diversi formati, nello specifico: `DOC`, `DOCX`, `TXT`, `PDF`, `TIF`, `MSG`, `EML`, `RTF`, `XSL`, `XML`. Nel proseguo della tesi si considera quindi come unità base dell'informazione per questa categoria il documento avente i metadati della scheda con, in aggiunta, il testo estratto da file ad essa associati.

- Un particolare tipo di documenti al quale ci si riferirà col termine *Eventi*. Siav gestisce i ticket riguardanti i problemi tecnici segnalati dai clienti tramite un sistema di *Customer Relationship Manager*, CRM, che permette di coordinare il lavoro tra helpdesk, tecnici e sviluppatori. In questo modo tutte le azioni effettuate dal personale coinvolto vengono registrate nel sistema. È quindi possibile collegarsi alla base di dati di appoggio ed estrarre tali informazioni, sotto forma di eventi, caratterizzati da una serie di metadati descrittivi (es. gravità del problema, tecnico assegnato, prodotto relativo, ecc.)[2].

- Pagine Web. La ricerca non prende in esame tutto il web ma si effettua una operazione di recupero delle pagine web di interesse utilizzando un *web crawler*[3] a partire da una lista di siti di interesse per il cliente.

Per tutte e tre le categorie considerate la lingua del testo di riferimento è l'italiano.

[2] In Siav vi è un particolare interesse su questo tipo di documenti in quanto sono alla base per l'applicazione delle tecniche di *Process Mining* utilizzate dai loro software. Per maggiori dettagli http://padova.processmining.it/public/publications/2011-manifesto-it.pdf

[3] Un *web crawler* è un programma che analizza automaticamente le pagine web recuperandone i contenuti, tra cui i collegamenti ipertestuali, dai quali esegue ricorsivamente nuove operazioni di recupero.

4.3 Indicizzazione dei documenti

L'indicizzazione in Solr prevede che i documenti in ingresso abbiano una delle estensioni compatibili (`XML`, `JSON`, `CSV`), in un formato apposito, da cui il sistema possa riconoscerne i vari `FieldType`, la cui mappatura è impostata nel file `schema.xml`[4].
Di seguito la descrizione di come è stata realizzata la conversione dei documenti, a seconda delle diverse sorgenti di documenti trattate, in modo da permetterne l'indicizzazione.

4.3.1 Documenti aziendali

Sono state fornite tre categorie diverse di documenti provenienti da Archiflow, nello specifico *Offerte*, *Comunicazioni*, *Documentazione Progetto*, corredate da un file `CSV`, ove ogni riga contiene i metadati di una singola scheda e il riferimento al nome del documento e della cartella con gli eventuali allegati associati. Essendo questi ultimi due elementi presenti nei formati nativi precedentemente indicati, è stato necessario realizzare una *utility di ingestion* che generi dei documenti utilizzabili da Solr e che provveda ad inviarli al motore per l'indicizzazione.
La prima parte del lavoro di sviluppo è quella che, per l'appunto, a partire da un file in formato nativo, effettua la lettura dei contenuti, che verranno salvati in un file testuale, in modo tale da permettere il successivo caricamento in Solr e la verifica della bontà delle procedure di estrazione del testo utilizzate. Per far questo si è ricorso alla stessa tecnica utilizzata nella desktop search di Microsoft, che fa uso del plugin *IFilter*[5]; questa funzionalità estrae il testo dal file, fornendo uno degli oggetti della ricerca. Si sono inoltre utilizzate, per i file con estensione `RTF` ed `EML`, delle librerie esterne, alcune provenienti dal progetto *Apache Tika*[6] e altre direttamente fornite dall'azienda committente. Il codice è stato scritto con il linguaggio

[4] Si ricorda che Solr permette una gestione maggiormente flessibile grazie all'auto riconoscimento dei campi a partire da un documento fornito in ingresso. Tuttavia in questa tesi si è scelto di definire in maniera personalizzata la gestione di ogni singolo campo di un documento di input per il sistema.

[5] https://msdn.microsoft.com/en-us/library/ms691105%28v=vs.85%29.aspx

[6] Nello specifico si veda https://github.com/KevM/tikaondotnet.

C# utilizzando come editor Microsoft Visual Studio Enterprise 2015.
La tabella 4.1 contiene informazioni sui documenti analizzati, precisamente la numerosità per categoria dei documenti elaborati dai quali il contenuto è stato salvato in un file .TXT per facilitarne l'elaborazione. La dimensione su disco dei documenti in tale formato, e la stima dello spazio occupato dei file nativi[7], è risultata essere di 2.13 Gigabyte a fronte di più di 60 Gigabyte di file originali.

Tipologia	Numero documenti elaborati
Comunicazioni	11974
Offerte	59763
Documentazione Progetto	4962

Tabella 4.1. Tabella con le informazioni relative ai documenti provenienti da Archiflow

Una volta estratto il testo di ogni file si è provveduto a creare un documento in un formato valido per Solr, contenente i metadadi relativi alla scheda documento a cui il file è associato più il testo estratto, ottenendo così un livello più di granularità rispetto a quello utilizzato in Archiflow, dove non vi è distinzione tra il documento principale e gli eventuali allegati della scheda a cui fanno riferimento. Il numero totale finale risultante di documenti indicizzati è risultato essere pari a 76699. Per l'interfacciamento con il motore di ricerca si è utilizzato il pacchetto di librerie apposite SolrNet[8], realizzate per il linguaggio C#.

4.3.2 Eventi aziendali

Un *Evento* è un documento testuale avente la struttura di tipo XML come da listato 4.1. Il contenuto presente è suddiviso in metadati con in aggiunta un

[7] Si fornisce una stima, e non il dato preciso, in quanto una parte dei documenti e degli allegati era in un formato non idoneo per effettuare l'indicizzazione (es .ZIP) o erano file protetti su cui non era possibile effettuarne l'elaborazione descritta. Si precisa che la modalità di gestione dei file salvati in Archiflow non è stata soggetta di analisi diretta in questa tesi.

[8] https://github.com/mausch/SolrNet

campo contenente una descrizione descrittiva; l'indicizzazione e il successivo reperimento hanno considerato entrambi gli elementi. Per l'analisi è stato fornito un file in formato `.XES`[9] con all'interno gli Eventi registrati fino al 2014. Su precisa richiesta è stato considerato come singolo documento tutto ciò che è contenuto all'interno dei tag `<EVENT>`, ottenendo così un totale di 70475 documenti, per uno spazio in memoria complessivo di 275 MB. Per l'inserimento è stato utilizzato uno script in linguaggio *JAVA*, e in questo caso non erano richieste particolari librerie per l'estrazione del testo ma semplicemente l'utilizzo di apposite librerie per la lettura di file presenti nel formato indicato.

```
<TRACE>
    <EVENT>
        ...
        <! -- Metadati >
        <DESCRIPTION> </DESCRIPTION>
    </EVENT>
    <EVENT>
        ...
        <! -- Metadati >
        <DESCRIPTION> </DESCRIPTION>
        ...
    </EVENT>
    ...
</TRACE>
```

Listing 4.1. Esempio di un Evento

4.3.3 Pagine web

La terza sorgente di informazioni considerata è stata il Web. La scelta è stata dettata dal fatto che spesso alcune notizie o particolari informazioni in merito ad un determinato cliente o servizio possono integrare i risultati di una determinata ricerca, anticipando o aumentando così l'informazione fornita. Per fornire all'utente un'informazione aggiuntiva che sia comunque

[9] http://www.xes-standard.org/

mirata e non troppo generica, si è utilizzato il *web crawler Apache Nutch*[10], configurato per fare in modo che indicizzi solo un piccolo sottoinsieme delle pagine web a partire da una lista di sorgenti di partenza, dette *seed*, e considerando una profondità di ricerca limitata. La versione utilizzata è stata la 1.4 la quale, con un'opportuna configurazione, prevede al suo interno l'auto invio a Solr delle pagine web recuperate, il quale provvederà ad indicizzarle in un indice specifico. Per questa specifica sorgente si è deciso che l'obiettivo sia quello di limitarsi al verificare la fattibilità di integrazione del web come sorgente informativa e di testare il funzionamento del *web crawler* indicato.

4.3.4 Analisi sull'estrazione dei diversi contenuti testuali

Uno dei principali colli di bottiglia riscontrati sono stati i tempi richiesti per l'estrazione del testo dai documenti provenienti da Archiflow[11]. Si è deciso quindi di generare dei file `.TXT` intermedi, allo scopo di evitare l'esportazione diretta in Solr. Questa scelta si è rivelata ottimale in quanto ha permesso di verificare in tempi ragionevoli la bontà del metodo di conversione utilizzato, e di fare diverse importazioni su indici differenti[12], descritte nelle successive sezioni. I risultati ottenuti hanno rivelato che la scelta di utilizzare gli *IFilter* si è rivelata valida, anche se è altamente consigliabile eseguire una operazione di controllo sui risultati per rilevare la presenza di eventuali elementi di rumore nel testo. L'integrazione del *web crawler* ha portato diversi problemi, in particolare di compatibilità con il sistema operativo installato nella macchina adibita al lavoro. Dopo varie prove si è trovata un versione compatibile e il sistema, configurati dei siti web di test come *seed* si è dimostrato essere valido.

---

[10] `http://nutch.apache.org/`

[11] Una stima, fatta utilizzando per il calcolo dei tempi le funzioni della libreria `System.Diagnostics.Stopwatch`:
`https://msdn.microsoft.com/en-us/library/system.diagnostics.stopwatch(v=vs.110).aspx`, basata su un campione di 12 documenti di diversa natura, ha rivelato che per l'estrazione del testo, da tali documenti, si impiegano circa 8 secondi.

[12] L'esportazione su Solr di tutti i 76699 elementi in formato `.TXT` della collezione dei documenti provenienti da Archiflow è stata stimata in 2112 secondi, di cui l'82% del tempo impiegato ad inviare i documenti mentre il rimanente utilizzato per il caricamento dei file.

4.3.5 Suddivisione degli indici

Uno dei problemi considerati è stato quello derivante dalla scelta se utilizzare un indice unico per tutte le fonti considerate o se utilizzarne di singoli specifici per ogni categoria (documenti aziendali, eventi, pagine web). L'utilizzo finale dell'applicazione ci ha suggerito di utilizzare la seconda opzione, in quanto l'interfaccia di ricerca ripartirà i risultati delle query in differenti liste ordinate per ciascuna sorgente. Questa scelta risulta opportuna in quanto si prevede come sviluppo futuro l'inserimento di ulteriori sorgenti di dati personalizzate per i diversi clienti: utilizzando un indice unico si renderebbe la collezione sperimentale che si è creata inutilizzabile. Inoltre avere un indice dedicato per ciascuna nuova sorgente rende l'operazione di indicizzazione più facilmente gestibile.

4.3.6 Indici testati

Sono state considerate quattro diverse tipologie per la creazione dei singoli indici trasposti, dai quali verranno in seguito reperiti i documenti applicando due diversi modelli. La motivazione è che si è voluto analizzare diverse soluzioni, tutte candidabili ad essere utilizzate come riferimento per lo sviluppo. Come precedentemente indicato queste soluzioni saranno in seguito valutate a partire da dei topic creati appositamente per quanto riguarderà le collezioni di documenti provenienti da Archiflow e di Eventi.
Di seguito l'elenco degli indici creati, considerando il `FieldType` di riferimento configurato per i documenti che vengono indicizzati. A seguire la descrizione sulle scelte effettuate.

- Indice 1
  - Tokenizer: `solr.StandardTokenizerFactory`
  - Rimozione *stop-word*: -
  - Stemming: -
  - Eventuali aggiunte: -
- Indice 2
  - Tokenizer: `solr.StandardTokenizerFactory`
  - Rimozione *stop-word*: `solr.StopFilterFactory`

  - Stemming: `solr.ItalianLightStemFilterFactory`
  - Eventuali aggiunte: -

- Indice 3
  - Tokenizer: `solr.StandardTokenizerFactory`
  - Rimozione *stop-word*: `solr.StopFilterFactory`
  - Stemming: `solr.SnowballPorterFilterFactory`
  - Eventuali aggiunte: -

- Indice 4
  - Tokenizer: `solr.StandardTokenizerFactory`
  - Rimozione *stop-word*: `solr.StopFilterFactory`
  - Stemming: `solr.ItalianLightStemFilterFactory`
  - Eventuali aggiunte: `solr.SynonymFilterFactory`

Analisi lessicale

Come descritto in precedenza il tokenizer implementato nella classe `solr.StandardTokenizerFactory` recupera i token dal testo considerando come delimitatori gli spazi bianchi e i segni di punteggiatura, dato che i documenti analizzati presentano l'ordinaria separazione dei termini mediante spazi bianchi e la normale punteggiatura.

Rimozione *stop word*

Questa procedura non è stata applicata nel primo indice mentre invece si è deciso di utilizzarla nelle altre tre configurazioni. La *stop-list* utilizzata è stata quella per la lingua italiana disponibile nel pacchetto di Solr. La motivazione per queste scelte sta nel voler verificare l'impatto nel processo di reperimento di questa operazione, dato che la non indicizzazione di questi termini comporta un risparmio di memoria su disco.

Stemming

Per questa operazione si sono effettuate tre diverse sperimentazioni. Nel primo caso si è scelto di non applicarla, nel secondo si è utilizzato il filtro `solr.ItalianLightStemFilterFactory`, caratterizzato dal fatto di elidere l'ultima o le ultime due lettere di una parola. Lo scopo di tale soluzione è quello di favorire una maggiore efficacia di reperimento nel caso di termini presenti al plurale o al singolare o, per certi casi, al maschile e al femminile[13]; lo stesso filtro è stato applicato nella quarta configurazione descritta, dopo l'espansione dei termini. Nel terzo caso studiato invece si è applicato il filtro più stringente `solr.SnowballPorterFilterFactory`, nella modalità per la lingua italiana[14].

Espansione dei termini

Nell'ultimo indice si è utilizzato il filtro che prevede l'espansione dei termini recuperandone i sinonimi[15] dal file `synonyms.txt`, grazie all'applicazione del filtro `solr.SynonymFilterFactory`. Questa scelta è frutto di una precisa richiesta del committente, con lo scopo di rendere il sistema più semplice per l'utente medio, permettendogli di ricercare documenti a partire da termini non direttamente inseriti nell'interrogazione ma in un certo qual modo connessi.
Il problema principale riscontrato è che, per quanto il sistema sia predisposto per la lettura del file indicato, quest'ultimo non contiene alcun dizionario adibito per l'espansione e richiede che venga creato ex-novo da chi intende utilizzare tale funzionalità. A tale scopo si è utilizzata la base di dati mista per le lingue italiano e inglese del progetto MultiWordNet[16]. Tale base di dati è stata fornita in input ad uno script di conversione per creare il

[13] Ad esempio le parole *informazioni* ed *informazione* si riducono alla stessa radice linguista *informazion*.

[14] Un esempio di applicazione di tale filtro si trova a `http://snowball.tartarus.org/algorithms/italian/stemmer.html`

[15] Il termine utilizzato in questi casi sarebbe *synset*, che sta ad indicare un gruppo di elementi considerati semanticamente equivalenti; si è tuttavia mantenuto nel testo il termine più comune `sinonimi`.

[16] `http://multiwordnet.fbk.eu/english/home.php`

dizionario dei sinonimi nel formato richiesto per essere elaborato dal filtro di Solr[17]. Poi si è scelto di applicare una seconda volta il filtro che rimuove le *stop word*, in quanto si è visto sperimentalmente che alla lettura dal file preposto viene applicato un recupero dei token, con una conseguente introduzione di termini che originariamente sono stati non considerati nel processo di indicizzazione.
In figura 4.1 è possibile vedere un esempio di applicazione delle procedure appena descritte nel caso di un indice della tipologia 4. Tale esempio illustra i differenti passi dell'analisi che Solr effettua applicando i filtri definiti. L'immagine, presa direttamente dall'interfaccia web di Solr, presenta l'elaborazione della frase *Test del taccuino in convivenza*. Ogni riga rappresenta, in ordine di applicazione, i filtri specifici utilizzati. Come si può notare i termini *del* e *in* vengono direttamente esclusi, mentre *taccuino* e *confidenza* vengono espansi. L'ultima riga rappresenta ciò che effettivamente viene inserito nell'indice trasposto.

Il listato 4.2 visualizza invece la chiamata cURL necessaria per impostare la configurazione del `Type` in modo da essere conforme a quanto definito per gli indici di tipo 4.

```
curl -X POST -H 'Content-type:application/json' --
    data-binary '{
     "add-field-type" : {
        "name":"SYNONYM",
        "class":"solr.TextField",
        "positionIncrementGap":"100",
        "analyzer" : {
           "tokenizer":{
              "class":"solr.StandardTokenizerFactory
                 " },
           "filters":[
                   {
                     "class":"solr.StopFilterFactory
                        ",
                     "format": "snowball",
```

[17] Precisamente per ogni termine i suoi sinonimi vanno riportati seguiti da una virgola, ad esempio *orribile,infame,orrendo,pauroso* .

```
                    "words":"lang/stopwords_it.txt"
                        ,
                    "ignoreCase":"true"
                },
                {
                 "class":"solr.
                    SynonymFilterFactory",
                 "synonyms":"synonyms.txt",
                 "expand":"true"
                },
                {
                    "class":"solr.StopFilterFactory
                        ",
                    "format": "snowball",
                    "words":"lang/stopwords_it.txt"
                        ,
                    "ignoreCase":"true"
                },
                {
                 "class":"solr.
                    ItalianLightStemFilterFactory
                    "
                }
            ]
        }
    }
}' http://localhost:8983/solr/Indice4/schema
```

Listing 4.2. Configurazione di un `Type` su Solr

4.3.7 Formato del documento inviato a Solr

La struttura del documento inviato a Solr prevede due campi:

- `FULL_TEXT`, che contiene il testo estratto e tutti i metadati, aventi come `FieldType` quello specificato dall'indice relativo.

- `REFERENCE`, che contiene un riferimento all'indirizzo di memoria del documento originale, utile in seguito per la valutazione. Questo campo è definito dal tipo standard `String`, non è stato oggetto dell'operazione di indicizzazione ma semplicemente memorizzato, inoltre non viene considerato nelle interrogazioni fatte in seguito.

4.3.8 Interrogazione

Prima di passare alla descrizione dei modelli utilizzati, si precisa che le stesse scelte fatte per la creazione dei diversi indici trasposti sono state applicate anche alle interrogazioni inserite, in linea con quanto usualmente si applica nel reperimento dell'informazione. Per quanto riguarda il *parser* testato, si è mantenuto quello di default impostato su Solr ovvero il *LuceneQueryParser*.

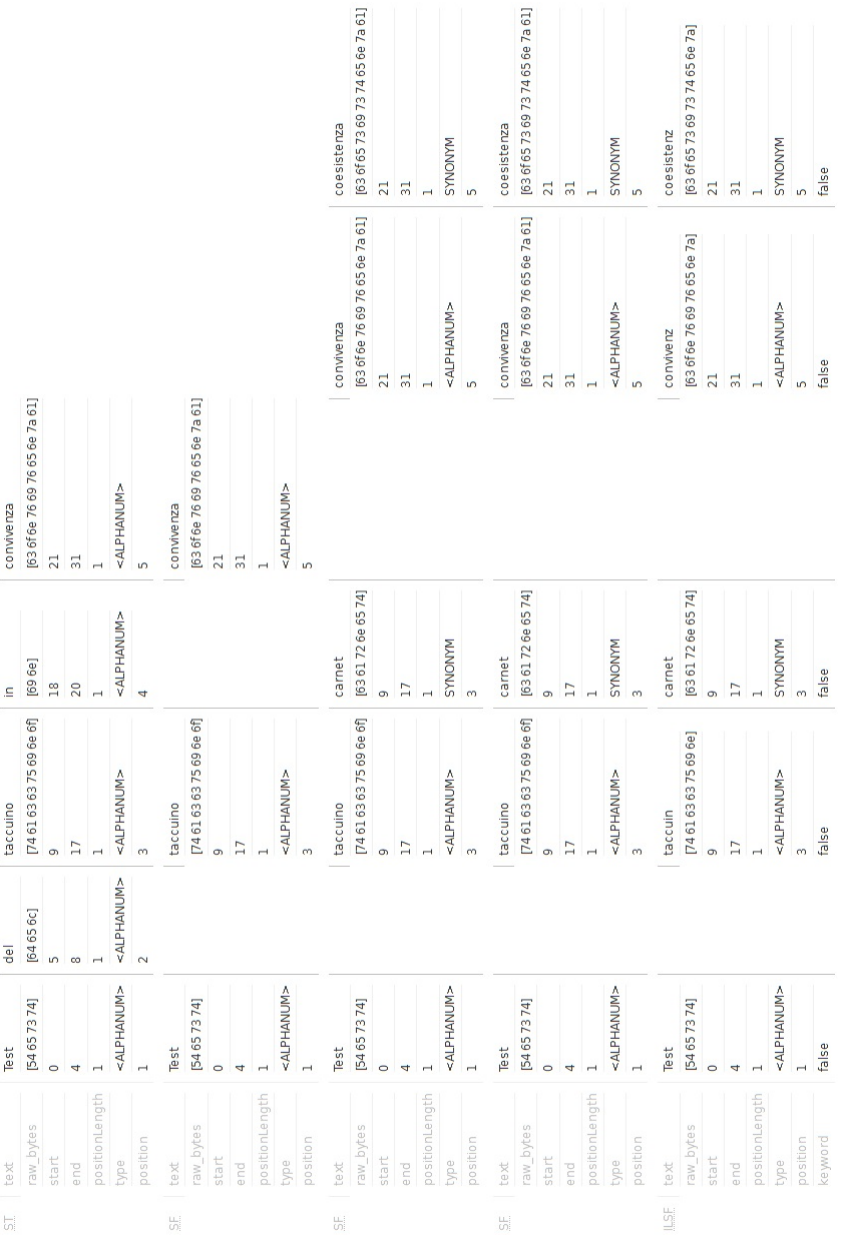


Figura 4.1. Esempio di `FieldType` con espansione dei termini in Solr

4.4 Modelli di reperimento

In questa sezione saranno descritti i due modelli di reperimento utilizzati nel prototipo e successivamente sottoposti a valutazione, ovvero il vettoriale e il probabilistico. La scelta è ricaduta su questi due in quanto sono tra i più importanti ed utilizzati, in particolar modo nel *search engine* Lucene[18]. Ad una introduzione sui principi su cui si basano seguirà la procedura utilizzata per configurarne l'utilizzo su Solr. Si precisa che ciò che segue non è una trattazione dettagliata, ma una sintesi atta a far comprendere l'idea base che li descrive; si rimanda alla bibliografia per una trattazione più approfondita[19].

4.4.1 Modello vettoriale

In questo modello si ha che i descrittori sono rappresentati come vettori di uno spazio lineare di dimensione finita, e i documenti e le interrogazioni vengono mappati attraverso combinazioni lineari di questi vettori. L'utente, esprimendo la sua esigenza informativa, definisce dei pesi c_i per i descrittori t_i che ritiene consistenti con quanto sta cercando; il documento x, e un'interrogazione y possono quindi essere definiti attraverso dei vettori nel modo seguente:

$$\mathbf{x} =< c_\mathrm{i} t_\mathrm{i} > \qquad \mathbf{y} =< c_\mathrm{i} t_\mathrm{i} > \tag{4.1}$$

La funzione di reperimento, che permette l'ordinamento dei documenti, si basa sul calcolo della distanza fra il vettore di ciascun documento e il vettore dell'interrogazione. L'efficacia di tale modello dipende dalla cosiddetta *Cluster Hypothesis*, la quale afferma che le descrizioni dei documenti rilevanti tendono a posizionarsi in maniera ravvicinata rispetto a quelle dei documenti considerati non rilevanti; in tal modo se l'ipotesi è verificata si avrà che il vettore dell'interrogazione sarà vicino al vettore del documento

[18] A tal proposito si consiglia la lettura di *BM25 The Next Generation of Lucene Relevance*, sul blog online all'indirizzo web `http://opensourceconnections.com/blog/2015/10/16/bm25-the-next-generation-of-lucene-relevation/`, a cura di Doug Turnbull.

[19] In particolare si rimanda al capitolo 5, *Modelli di indicizzazione e reperimento*, del testo *Information Retrieval, Metodi e modelli per i motori di ricerca*, Massimo Melucci, FrancoAngeli Editore, e al materiale citato del corso di reperimento dell'informazione, Università degli Studi di Padova.

rilevante per quella interrogazione. Normalmente per calcolare questa distanza si utilizza una misura di similarità: i documenti ai quali corrisponde un valore maggiore della misura di similarità sono considerati più simili alla query. Tra le misure di similarità la più diffusa è la *cosine correlation*, dove si calcola il coseno dell'angolo fra i vettori del documento e la query: se i due vettori sono identici il coseno vale 1, mentre se non condividono alcun termine 0.
Per la pesatura dei termini esistono diversi schemi, tra i più noti, considerando c_{ij} il peso del termine t_j nel documento i:

- Schema di pesatura *binario* dove il peso del termine vale 1 se è presente nel documento, 0 altrimenti.

- Schema di pesatura basato su *Term Frequency* TF

$$c_{ij} = f_{ij} \tag{4.2}$$

  dove f_{ij} è la frequenza del termine t_j nel documento i.

- Schema di pesatura basato su *Inverse Document Frequency* IDF

$$c_{ij} = \frac{\log N}{n_j} \tag{4.3}$$

  dove n_j rappresenta il numero di documenti in cui appare t_j fra tutti gli N della collezione.

- Schema di pesatura basato su TF-IDF

$$c_{ij} = f_{ij} \frac{\log N}{n_j} \tag{4.4}$$

L'implementazione su Lucene, e di conseguenza su Solr, di tale modello si trova con il riferimento di `ClassicSimilarity`[20] e combina lo schema di

[20] Questa dicitura è utilizzata perché fino alla versione precedente a quella trattata era il modello di reperimento impostato di default per la ricerca.

pesatura basato su TF-IDF e la normalizzazione del coseno con altri fattori specifici[21].

Per impostare tale modello in un indice di Solr si deve inserire la seguente specifica nel file `schema.xml`

```
<similarity class="solr.ClassicSimilarityFactory
   "/>
```

Listing 4.3. Configurazione del modello vettoriale su Solr

4.4.2 Modello probabilistico

Come già enunciato nel capitolo 2 la funzione di reperimento del modello probabilistico è data dal rapporto tra la verosimiglianza dell'ipotesi che il documento sia rilevante e quella che il documento non lo sia. Le distribuzioni di probabilità su cui si appoggia sono parametriche, e questi parametri devono essere stimati tenendo conto di informazioni discernibili dai documenti stessi della collezione a disposizione con l'obiettivo di presentare i risultati ordinati secondo una lista non crescente di probabilità di rilevanza rispetto all'esigenza informativa dell'utente. La metrica che si utilizza è una misura di *similarità* tra documento e l'interrogazione basata sulle occorrenze dei termini della query nei singoli documenti. Una delle funzioni di ranking più utilizzate, ispirate da quanto enunciato, è conosciuta come Okapi BM25 e si esprime attraverso la seguente formula:

$$w(t_j, d_i) = \frac{TF(t_j, d_i)}{TF(t_j, d_i) + k((1-b) + b\frac{dl}{avdl})} * \log \frac{N - n_j + 0.5}{n_i + 0.5} * \\ * \frac{(k_3 + 1)TF(t_j, q)}{k_3 + TF(t_j, q)} \tag{4.5}$$

con

- $w(t_j,d_i)$ t_j nel documento d_j
- $TF(t_j,d_i)$ frequenza del termine t_j nel documento d_i

[21] Per maggiori dettagli si veda https://lucene.apache.org/core/6_0_0/core/org/apache/lucene/search/similarities/TFIDFSimilarity.html

- $TF(t_j,d_i)$ frequenza del termine t_j nella quella query q
- dl lunghezza del documento
- $avdl$ la lunghezza media di un documento nella collezione
- N numero dei documenti della collezione
- n_j numero dei documenti che contengono il termine j

con valori per i parametri, stabiliti empiricamente, pari a b=*0.75*, k_1=*1.2*, k_3=*1000*. Come si può notare sono presenti entrambi i fattori relativi alla frequenza di un termine sia nella collezione che nell'interrogazione oltre alla lunghezza del documento ed a un fattore di normalizzazione detto *Pivot*.

In Solr dalla versione analizzata questo modello è quello che si trova configurato di default, maggiori dettagli implementativi si possono trovare facendo riferimento alla classe `BM25Similarity`[22].

Per impostare tale modello in un indice di Solr come detto non è necessario alcuna modifica, se eventualmente si effettuano dei test e tale impostazione non è più valida si dovrà inserire la seguente specifica nel file `schema.xml`

```
<similarity class="solr.BM25SimilarityFactory">
<!-- start with the defaults  -->
      <float name="k1">1.2</float>
      <float name="b">0.75</float>
</similarity>
```

Listing 4.4. Configurazione del modello probabilistico BM25 su Solr

[22] Si veda https://lucene.apache.org/core/6_0_0/core/org/apache/lucene/search/similarities/BM25Similarity.html

Capitolo 5

Collezione sperimentale

In questo capitolo si descrivono le procedure utilizzate per valutare le scelte progettuali descritte nel capitolo 4. Per eseguire tale analisi si è applicata, nei limiti e con le modalità attuabili nel caso studiato in questa tesi, la procedura di valutazione standard in uso da molti anni nel campo del reperimento dell'informazione e utilizzata nelle principali campagne di valutazione internazionali. Di seguito, dopo una sintetica descrizione dell'argomento, si illustrano le azioni intraprese nel caso di studio. Nelle sezioni finali si riportano i risultati ottenuti applicando le specifiche metriche e le corrispondenti analisi; si precisa che per tale scopo si è usata una apposita libreria per MATLAB[1], la quale richiede che i dati in ingresso siano forniti seguendo lo specifico formato, descritto nel seguito.

5.1 Collezione di test

In generale per valutare un sistema si fa ricorso a delle collezioni di test già costruite, le quali vengono fornite in ingresso al sistema e, a seguito delle analisi dei risultati si effettuano le valutazioni. Esistono diverse iniziative, dette campagne di valutazione, svolte a livello internazionale, che vedono collaborare diversi gruppi di ricerca, accademici e industriali, al fine di produrre delle linee guida e poi effettuare le valutazioni dei diversi sistemi di

[1] http://it.mathworks.com/products/matlab/

information retrieval che sono stati sviluppati. Tra le principali campagne di valutazione ci sono: TREC[2] (Text REtrieval Conference), CLEF[3] (Conference and Labs of the Evaluation Forum), NTCIR[4] (NII Testbed and Communitity for Information access Research) e FIRE[5] (Forum for Information Retrieval Evaluation).
La possibilità di utilizzare una collezione già esistente è stata accantonata per il caso di studio data la natura molto specifica della base documentale sulla quale il prototipo deve lavorare, e dalla volontà di verificare il comportamento del prototipo proprio sui dati originali. Si è quindi proceduto, in modo coerente con quanto si fa in una campagna di valutazione, quando non è già disponibile la collezione sperimentale per il task di interesse, con la creazione di una collezione sperimentale a partire dai documenti a disposizione. Per creare una collezione di test è necessario definire prima di tutto un *task* di ricerca: l'obiettivo che si è fissato è stato quello di ottenere dal sistema il maggior numero di documenti ritenuti rilevanti nelle posizioni di testa dalla lista ordinata dei risultati, simulando il comportamento tipico dei motori di ricerca sul web.

Questa la sequenza delle operazioni da eseguire:

- Definizione dell'insieme di documenti che formano la collezione.
- Definizione di un insieme di *topic* che rifletta delle reali esigenze informative degli utenti, che in questo caso sono gli utilizzatori in Siav del software Archiflow. Per topic si intende, nell'ambito della valutazione, l'esigenza informativa degli utenti e si esprime attraverso la formulazione dell'interrogazione o delle interrogazioni.
- Definizione dei giudizi di rilevanza da parte di un gruppo di esperti, per mettere in relazione i documenti della collezione e i topic.

Uno dei problemi più complessi da affrontare è la formulazione dei giudizi di rilevanza in relazione ai topic definiti. Questa operazione, se fatta per

[2] http://trec.nist.gov/

[3] http://www.clef-initiative.eu/

[4] http://research.nii.ac.jp/ntcir/index-en.html

[5] http://fire.irsi.res.in

tutti i documenti di una collezione, richiederebbe tempi irragionevoli[6]. Uno dei metodi più comuni per ovviare a questo problema, utilizzato anche in questa tesi, è quello di estrarre un campione dei documenti a partire dagli esperimenti dei partecipanti: tale metodo prende il nome di *Pooling*. Prima di descrivere tale tecnica è necessario dare alcune definizioni[7].

Definiamo:

$$D = d_1, d_2, .., d_{\mathrm{n}}$$

un insieme di documenti

$$T = t_1, t_2, .., t_{\mathrm{m}}$$

un insieme di topic

Dato un numero naturale

$$N \in \mathbb{N}^+$$

chiamato *lunghezza della run*, una *run* è definita come una funzione

$$R : T \to D^N$$
$$t \mapsto \mathbf{r}_{\mathrm{t}} = (d_1, d_2, .., d_{\mathrm{N}})$$

tale che

$$\forall t \in T, \forall j, k \in [1, N] \mid j \neq k \Rightarrow \mathbf{r}_{\mathrm{t}}[j] \neq \mathbf{r}_{\mathrm{t}}[k]$$

dove $\mathbf{r}_{\mathrm{j}}$ indica j-esimo elemento del vettore $\mathbf{r}_{\mathrm{t}}$.

La tecnica del *pooling* consiste nel creare un insieme di documenti selezionati tra i primi k di ogni run sottomessa, dove il valore di k definisce la

[6] Ad esempio per la campagna di valutazione TREC 1 i documenti giudicare sono 741.856; avendo l'interesse di associare i documenti rilevanti a 50 topic, risulta necessario verificare circa 37 milioni di documenti, e considerato anche un tempo sottostimato di 30 secondi per la valutazione di un documento, sarebbero necessari circa 37 anni per completare il lavoro di valutazione.

[7] Si è utilizzata la notazione definita in: Angelini, M. et al. (2014). *"VIRTUE: A visual tool for information retrieval performance evaluation and failure analysis*, Ferro, N. et al. (2016). *"The twist measure for IR evaluation: Taking user's effort into account"*.

profondità del pool: solo questi documenti saranno giudicati mentre quelli che non sono stati recuperati verranno automaticamente giudicati come non rilevanti ai fini del topic in esame. Ovviamente minore è la profondità del pool, minore sarà il numero di documenti da giudicare. Si è dimostrato che questa tecnica è robusta ed affidabile[8]. Per quanto riguarda la profondità, la più frequentemente utilizzata è pari a 100, ma vi sono situazioni dove è maggiore e altre dove è minore.

5.2 Creazione delle collezioni sperimentali per il caso trattato

Per il caso in esame il problema della creazione di una collezione sperimentale non è stato di immediata risoluzione. Questo perché sono emersi alcuni problemi, in particolare:

- Non si è in un contesto di una campagna di valutazione internazionale e quindi non si hanno, in prima istanza, esperimenti aggiuntivi a quelli effettuabili sul sistema da analizzare, da cui estrarre le *run* per l'operazione di *pooling*.

- Le risorse umane e temporali a disposizioni sono limitate. Infatti non si possono avere per molto tempo persone esperte adibite all'assegnazione dei giudizi di rilevanza, in quanto vengono sottratte alla normale attività lavorativa aziendale.

- Definire i topic non è un'azione di immediata realizzazione. Richiede che ci sia una profonda conoscenza della base documentale e devono essere identificate le necessità informative degli utenti finali.

Per risolvere questi problemi sono state analizzate diverse soluzioni ed alla fine si è stabilita la seguente strategia operativa. Delle tre categorie documentali considerate, documenti provenienti da Archiflow, Eventi e pagine

[8] Si veda [SIGIR1998] J.Zobel. *How reliable are the results of large-scale information retrieval experiment?* e [IPM2000] E. M. Voorhees. *Variations in relevance judgements and the measurement of retrieval effectiveness.*

web, si sono scelte le prime due, considerando per entrambe tutti i documenti a disposizione. Sono stati definiti, attraverso un'opportuna fase di analisi con il committente, cinque topic per ogni categoria, ognuno dei quali tradotto in una query mediante un'interrogazione creata manualmente[9]. Per ottenere altri esperimenti, si è scelto di utilizzare, affiancando a Solr, il motore di ricerca Terrier e due *Desktop Search*[10], precisamente *Copernic*[11] e *Windows Search*, presente nella distribuzione nel sistema operativo del calcolatore utilizzato. A tutti, una volta indicizzati i documenti, sono state sottoposte le interrogazioni e, ottenute le run, calcolati il numero di documenti da giudicare, in base a diverse profondità di taglio. Questo per permettere agli esperti di dominio che assegneranno i giudizi, di stabilire quanti documenti giudicare, in base alle risorse a disposizione. Di seguito i dettagli delle operazioni eseguite.

5.3 Formato dei topic, delle run e del pool

Si è utilizzato il formato TREC utilizzato nella gran parte delle campagne internazionali. Per le run e il topic in particolare, tale scelta è stata dettata dal fatto che per il calcolo delle successive metriche si è utilizzata la libreria MATLAB denominata MATTERS[12], realizzata appositamente per l'attività di valutazione di *information retrieval*. Di seguito la descrizione dei singoli formati.

Topic

Il formato utilizzato per la definizione di un topic è quello che si può vedere in Listing 5.1. I tag che verranno effettivamente utilizzati saranno `NUM`, che identificherà unicamente il topic, e `TITLE`, ove vi sarà definita la query da inviare ai diversi sistemi. Il campo `DESC` serve per la costruzione della query mentre `NARR` per chi dà il giudizio sul documento.

[9] Le run ottenute in questo modo sono dette *manual run*.

[10] https://en.wikipedia.org/wiki/Desktop_search

[11] https://www.copernic.com/

[12] La libreria e la relativa documentazione si trovano all'indirizzo http://matters.dei.unipd.it/

```
<TOP>
<NUM> Numero del topic
<TITLE> Query inerente al topic
<DESC> Descrizione sintetica del topic
<NARR> Descrizione espansa del topic
</TOP>
```

Listing 5.1. Schema di un `topic`

Si precisa che si è deciso di non riportare in questa tesi i topic utilizzati in seguito per la valutazione, in quanto contenenti informazioni sensibili per l'azienda. Si precisa inoltre che non sono stati definiti per ottenere specifiche informazioni ma per valutare nel complesso le varie configurazioni del prototipo realizzato.

Run

Per quanto riguarda invece le run, il formato che si vuole ottenere in risposta è quello mostrato nel Listing 5.2. Il campo `<topic-id>` è il corrispettivo di `<NUM>`, mentre `<run-id>` è l'identificatore della run. I campi `<document-id>` e `<rank>` tengono conto di come i risultati sono stati forniti e in quale ordine. Il campo numerico `<score>`, deve essere consistente con il `<rank>`, ma il valore puntuale non è di interesse in questo caso[13].

```
<topic-id> Q0 <document-id> <rank> <score> <run-id>
```

Listing 5.2. Formato di una `run`

Pool

Il formato del pool è quello mostrato nel Listing 5.3. I campi `<topic-id>` e `<document-id>` sono coerenti con quello delle run da cui il pool viene ricavato, mentre il tag `<relevance-grade>` rappresenta il giudizio assegnato

[13] Elementi utilizzati in seguito come i *desktop search* non prevedono la visualizzazione di un eventuale punteggio assegnato dal quale ricavare l'ordinamento. Quando non fosse presente si è scelto di impostare un ordinamento numerico decrescente unitario.

al documento per il topic specifico, espresso con un 1 se il documento è rilevante, 0 altrimenti.

```
<topic-id> 0 <document-id> <relevance-grade>
```

Listing 5.3. Formato di un `pool`

5.4 Costruzione del pool

Si illustrano qui di seguito gli esperimenti fatti per ricavare le run, dalle quali si estrarrà l'insieme dei documenti da giudicare, formando così il pool. Per i risultati ottenuti dai *desktop search* e da Solr si sono implementati degli script appositi per convertire i risultati nel fromato TREC, cosa invece non necessaria in Terrier dato che fornisce già le risposte alle query in tale formato.

Terrier

La versione usata è la `4.1`. Si sono utilizzate quattro differenti configurazioni per l'operazione di indicizzazione, applicando o meno la rimozione delle *stop word*[14] e, per entrambe le modalità, applicando i modelli BM25[15] e TF-IDF[16]. Per la configurazione si è provveduto ad inserire nel file `terrier.properties`[17] le impostazioni per l'indicizzazione di file testuali `.TXT`, mentre le run sono state ottenute dando in input al sistema un file contenenti i topic nel formato previsto da TREC.

[14] Si è utilizzato come lista di parole per l'italiano la seguente: http://members.unine.ch/jacques.savoy/clef/italianST.txt

[15] http://terrier.org/docs/v4.1/javadoc/org/terrier/matching/models/BM25.html

[16] terrier.org/docs/v4.1/javadoc/org/terrier/matching/models/TF_IDF.html

[17] Per le impostazioni sui campi configurabili si veda http://terrier.org/docs/v4.1/properties.html

Desktop search

Per i *desktop search* la scelta è ricaduta su Copernic versione 4, e Microsoft Search. Si è scelto di usare questi due in quanto non si basano su Lucene, come invece accade per Solr. Questo è importante in quanto, in caso contrario, si metterebbero a confronto risultati provenienti dallo stesso motore di ricerca. Va precisato che tali software non forniscono un ranking dei risultati con un relativo punteggio assegnato ai documenti, ma utilizzano il nome del file come ordinamento dei risultati.

Solr

È tipico, quando si effettua la creazione del pool, includere delle run provenienti direttamente da esperimenti che riguardano il sistema da valutare. In questa circostanza le query sono state inviate a due diversi indici di Solr, precisamente al numero 1 e al numero 2 descritti nello schema della sezione 4.3.6, utilizzando per entrambi il modello di reperimento BM25, con i parametri di default impostati dal sistema.

5.4.1 Scelta del livello di taglio e metriche utilizzate

In totale, per ogni collezione documentale considerata, si sono ottenute otto run: due da Solr, quattro da Terrier, una da Copernic e una da Windows Search. Da queste si ricavano i documenti da inserire nel pool scegliendo i primi k documenti da ognuna. Impostare il livello del taglio k da utilizzare, in modo da ottenere un buon compromesso tra risorse a disposizione e validazione dei risultati, è stato oggetto di analisi. Si è scelto una soglia pari a $k = 10$, e si è saliti fino ad una massima ipotizzata giudicabile pari a $k = 50$. Agli esperti di dominio sono stati presentati questi risultati e, in base alle risorse a disposizione, si è stabilito quale taglio applicare. La creazione di un pool com bassa profondità, ovvero con k«100, viene chiamata *shallow pooling*. In tabella 5.1 si possono vedere i risultati per la collezione dei documenti provenienti da Archiflow mentre in tabella 5.2 quelli relativi agli Eventi.

Dopo aver proposto agli esperti le diverse cardinalità di documenti da giudicare si è scelto di utilizzare il taglio pari a $k = 25$ per gli Eventi, per un totale di 287 valutazioni, mentre per i documenti provenienti da Archiflow

Topic / k	10	15	20	25	30	35	40	50
Topic 1	34	51	63	78	94	108	123	146
Topic 2	25	34	41	53	61	70	78	91
Topic 3	30	45	61	76	91	106	123	152
Topic 4	31	47	62	78	93	106	120	149
Topic 5	31	42	53	65	74	86	96	114
Totale da giudicare	151	219	280	350	413	476	540	652

Tabella 5.1. Tabella con riportati il numero di documenti da giudicare in base ai diversi livelli di *cut-off*, per la creazione di un pool per i documenti provenienti da Archiflow

Topic / k	10	15	20	25	30	35	40	50
Topic 1	15	19	29	42	55	71	86	102
Topic 2	27	33	41	53	61	69	77	80
Topic 3	35	50	65	77	88	95	106	133
Topic 4	27	39	51	62	67	75	84	105
Topic 5	20	28	43	53	60	71	76	90
Totale da giudicare	124	169	229	287	331	381	429	510

Tabella 5.2. Tabella con riportati il numero di documenti da giudicare in base ai diversi livelli di *cut-off*, per la creazione di un pool per gli Eventi

uno inferiore, pari a $k = 20$, per un totale di 280 valutazioni. Va precisato che per quanto riguarda gli Eventi, il compito di assegnare i giudizi era di pertinenza di un solo esperto, mentre per l'altra collezione sono stati assegnati tre esperti. Questo fatto non è dipeso dalle singole velocità di assegnazione di un giudizio ma dalla natura stessa dei documenti: un Evento è un semplice documento di testo, ben strutturato e non vi sono particolari differenze tra i vari elementi mentre, i documenti eterogenei provenienti da Archiflow, richiedono tempi superiori per essere analizzati. L'assegnazione dei giudizi è stata fatta utilizzando una rilevanza binaria: 1 per documenti rilevanti, 0 altrimenti. Si è scelta questa strategia in quanto è la più immediata per fare delle analisi ed è stata considerata valida per il task di ricerca assegnato: ottenere il maggior numero di documenti rilevanti nelle posizioni iniziali della lista dei risultati. A partire da queste considerazioni sono

state calcolate diverse metriche sulle run ottenute. Si utilizza la seguente notazione per una maggiore chiarezza espressiva:

- D = insieme dei documenti recuperati
- D^* = insieme dei documenti rilevanti recuperati
- $|D^*|$ = *recall base*. Ogni topic ha una *recall base* fissa determinata dal pool

Tutte le misure utilizzate assumono valori compresi tra `0` e `1`, dove `0` è il caso pessimo mentre `1` il caso migliore. Si ricorda che la *Precision*, è definita come

$$Precision = \frac{|D^* \cap D|}{|D|}$$

si è calcolata una variante di questa metrica, la p@10, che rappresenta la stessa metrica calcolata però fino al livello 10, ovvero considerando solo i primi dieci documenti recuperati di ogni run.

Infine si è misurata la cosiddetta *Average precision*, che è tanto più elevata quanto la distribuzione dei documenti rilevanti recuperati si attesta nelle posizioni di testa del ranking della run per un dato topic. L'equazione che la descrive è la seguente:

$$AP := \frac{1}{RB_t} \sum_{k=1}^{N} \tilde{r}_t[k] \frac{\sum_{h=1}^{k} \tilde{r}_t[h]}{k} \tag{5.1}$$

ove $\tilde{r}_t[k]$ rappresenta per una run il *relevance weight* per il documento in posizione k, relativo al topic t, che nel caso binario qui in esame assume il valore 1 se il documento risulta rilevante, 0 in alternativa. Il limite N rappresenta la lunghezza della run, mentre RB_t equivale $|D^*|$ per il topic t.

5.5 Risultati ed analisi

La scelta di utilizzare due *desktop search* non si è rivelata particolare efficace come si credeva in origine. Infatti si è notato che entrambi quelli utilizzati non forniscono una classifica basata su dei punteggi, almeno pubblicamente, ma utilizzano di default l'ordinamento per nome del file di origine; si è

mantenuto questo riferimento per le metriche e soprattutto per il recupero dei file, elemento per il quale si è deciso di utilizzarli lo stesso, ma per future operazioni di questo tipo se ne sconsiglia l'applicazione. Fattore ulteriormente negativo, evidenziato dalla lettura dei risultati delle interrogazioni, è che entrambi forniscono gli stessi documenti in uscita per ogni query inserita, quindi di fatto uno dei due era ridondante ai fini della creazione del pool; questo si può verificare con la lettura delle metriche, in quanto tutte le run ottenute per i due *desktop search* sono risultate uguali per ogni singolo topic. Fortunatamente la scelta di utilizzare Terrier, per quanto abbia richiesto uno sforzo suppletivo per la configurazione, si è rivelata ottima, in quanto ha contribuito alla validazione della collezione creata.
Un elemento di criticità si è rivelato essere la definizione dei topic, in quanto ha richiesto una difficile e non immediata simulazione dell'utilizzo futuro del sistema progettato, ma d'altro canto si è rivelato essere un ulteriore fattore di analisi e di confronto sulle diverse scelte da operare. La valutazione da parte degli esperti ha richiesto un impegno complessivo di quattro persone per circa due ore di lavoro ciascuna.
Prima di analizzare i risultati ottenuti, in base alle metriche utilizzate, è bene precisare che le considerazioni che vengono espresse, sono da considerarsi relative allo specifico caso in esame, e che non è possibile generalizzare i risultati formulando giudizi definitivi sulla bontà dei motori utilizzati.

Si riportano le tabelle con le metriche calcolate.

Topic 1	Topic 2	Topic 3	Topic 4	Topic 5
22	9	1	0	13

Tabella 5.3. *Recall Base* per il pool inerente ai documenti provenienti da Archiflow

Topic 1	Topic 2	Topic 3	Topic 4	Topic 5
22	44	65	24	31

Tabella 5.4. *Recall Base* per il pool inerente agli Eventi

Le tabella 5.3 riporta i valori delle *recall base* per il pool relativo ai documenti provenienti da Archiflow. Per il Topic 4 non è emerso alcun documento rilevante, di conseguenza non ha senso calcolare le metriche relative, in quanto la p@10 sarebbe sempre pari a 0 per ogni run, mentre la

AP non sarebbe calcolabile, data la presenza di uno zero a denominatore. Un altro caso molto particolare è quello del Topic 3 della medesima collezione, il cui pool riporta solamente un documento rilevante, e che l'unica configurazione che sia riuscito a recuperarlo tra i primi dieci documenti, tra l'altro in prima posizione, come si nota dalla tabella 5.8, sia quella di Solr con l'espansione dei termini. Questi due casi sono risultato di query particolarmente ricche di termini ma al contempo molto generali.
Per quanto riguarda i Topic 1, 2 e 5, le metriche associate riportate nelle tabelle 5.6, 5.7 e 5.9, evidenziano le buone prestazioni ottenute mediamente da Solr, rispetto a dei risultati non eccellenti ottenuti dagli altri due sistemi, in particolare di Terrier.

Per la collezione relativa agli Eventi, la tabella 5.4 riporta valori di *recall base* abbastanza elevati rispetto al caso precedente, con in particolare certe situazioni, come ad esempio per il Topic 3, dove la percentuale dei documenti rilevanti raggiunge l'84% del totale presenti nel pool. Analizzando le metriche si nota un comportamento medio non soddisfacente per i *desktop search* con due casi, relativi al Topic 1 e al Topic 4, riportati nelle tabelle 5.10 e 5.13, dove non hanno recuperato alcun documento, mentre Solr ed Terrier riportano valori nel primo caso eccellenti, e nel secondo buoni. I valori della tabella 5.11, relativi al Topic 2, riportano valori molto alti per tutti i sistemi, in particolare Solr configurato con il modello vettoriale risulta essere il migliore anche nelle diverse configurazioni. I risultati del Topic 3, il cui pool è caratterizzato da un elevata *recall base*, visibili nella tabella 5.12, evidenziano Terrier come sistema migliore, rispetto anche a tutte le diverse configurazioni di Solr. Il topic 5, tabella 5.14, evidenzia in particolar modo una differenza notevole per quanto riguarda la AP tra il modello vettoriale e probabilistico in Terrier, mentre questo è meno evidente in Solr.

Analizzando le scelte diverse di indicizzazione e di scelta del modello di reperimento in Solr, descritte nelle sezioni 4.3.6 e 4.4, si notano diversi elementi. Per la categoria Eventi il modello vettoriale è risultato essere il più performante mentre le diverse configurazioni per l'operazione di *stemming* non hanno portato a sostanziali differenze. Quest'ultimo risultato è in linea con quanto previsto, data la natura molto schematica della forma di tali documenti, aventi un parte di testo descrittivo molto ristretta. L'uso o meno dell'operazione di rimozione delle *stop word* non ha evidenziato grosse variazioni nelle prestazioni.

Analizzando invece i risultati relativi alla collezione di documenti provenienti da Archiflow la situazione è molto più variabile, si nota in parte un miglioramento delle prestazioni quando si è utilizzato il modello probabilistico, ma per le altre scelte fatte è difficile interpretare i risultati e fare delle considerazioni. La conseguenza di questo risultato è da imputare al numero di topic sottomessi, troppo pochi per il tipo di documenti analizzati.

Nome run	Descrizione
Terr1	run ottenuta utilizzando Terrier, senza la rimozione delle *stop word* applicando il modello BM25
Terr2	run ottenuta utilizzando Terrier, con la rimozione delle *stop word* applicando il modello BM25
Terr3	run ottenuta utilizzando Terrier, senza la rimozione delle *stop word*, applicando il modello TF_IDF
Terr4	run ottenuta utilizzando Terrier, con la rimozione delle *stop word* applicando il modello TF_IDF
Copen	run ottenuta attraverso il Desktop Search Copernic
Windws	run ottenuta attraverso il Desktop Search Windows Search
Solr1P	run ottenuta con Solr utilizzando un indice di tipo 1 con il modello BM25
Solr2P	run ottenuta con Solr utilizzando un indice di tipo 2 con il modello probabilistico BM25
Solr3P	run ottenuta con Solr utilizzando un indice di tipo 3 con il modello probabilistico BM25
Solr4P	run ottenuta con Solr utilizzando un indice di tipo 4 con il modello probabilistico BM25
Solr1V	run ottenuta con Solr utilizzando un indice di tipo 1 con il modello vettoriale
Solr2V	run ottenuta con Solr utilizzando un indice di tipo 2 con il modello vettoriale
Solr3V	run ottenuta con Solr utilizzando un indice di tipo 3 con il modello vettoriale
Solr4V	run ottenuta con Solr utilizzando un indice di tipo 4 con il modello vettoriale

Tabella 5.5. Catalogazione delle run ottenute

Run	p@10	AP
Terr1	0.1000	0.0954
Terr2	0.1000	0.1043
Terr3	0.1000	0.0954
Terr4	0.1000	0.1043
μ Terrier	**0.1000**	**0.0999**
Copen	0.4000	0.1807
Windw	0.4000	0.1807
μ D.S.	**0.4000**	**0.1807**
Solr1P	0.9000	0.6641
Solr2P	0.8000	0.6065
Solr3P	0.7000	0.5908
Solr4P	1.0000	0.5054
Solr1V	0.2000	0.3920
Solr2V	0.2000	0.3942
Solr3V	0.2000	0.3922
Solr4V	0.2000	0.3942
μ Solr	**0.525**	**0.4927**

Tabella 5.6. Metriche Doc_A Topic 1

Run	p@10	AP
Terr1	0.1000	0.2042
Terr2	0.1000	0.2132
Terr3	0.1000	0.2042
Terr4	0.1000	0.2132
μ Terrier	**0.1000**	**0.2087**
Copen	0.5000	0.4362
Windw	0.5000	0.4362
μ D.S.	**0.5000**	**0.4362**
Solr1P	0.6000	0.6649
Solr2P	0.6000	0.4917
Solr3P	0.6000	0.6649
Solr4P	1.0000	0.6530
Solr1V	0.6000	0.6710
Solr2V	0.6000	0.6710
Solr3V	0.6000	0.6710
Solr4V	0.6000	0.6327
μ Solr	**0.6500**	**0.6401**

Tabella 5.7. Metriche Doc_A Topic 2

Run	p@10	AP
Terr1	0.0000	0.0000
Terr2	0.0000	0.0000
Terr3	0.0000	0.0000
Terr4	0.0000	0.0000
μ Terrier	**0.000**	**0.0000**
Copen	0.0000	0.0047
Windw	0.0000	0.0047
μ D.S.	**0.0000**	**0.0047**
Solr1P	0.0000	0.0500
Solr2P	0.0000	0.0526
Solr3P	0.0000	0.0278
Solr4P	1.0000	1.0000
Solr1V	0.0000	0.0556
Solr2V	0.0000	0.0476
Solr3V	0.0000	0.0093
Solr4PV	0.0000	0.0476
μ Solr	**0.125**	**0.1612**

Tabella 5.8. Metriche Doc_A Topic 3

Run	p@10	AP
Terr1	0.3000	0.0726
Terr2	0.1000	0.0627
Terr3	0.3000	0.0726
Terr4	0.1000	0.0627
μ Terrier	**0.2000**	**0.0676**
Copen	0.0000	0.0905
Windw	0.0000	0.0905
μ D.S	**0.000**	**0.0905**
Solr1P	0.7000	0.7256
Solr2P	0.8000	0.7606
Solr3P	0.8000	0.7606
Solr4P	1.0000	0.6926
Solr1V	0.7000	0.5524
Solr2V	0.6000	0.6084
Solr3V	0.6000	0.6084
Solr4V	0.7000	0.6839
μ Solr	**0,7375**	**0,6741**

Tabella 5.9. Metriche Doc_A Topic 5

Run	p@10	AP
Terr1	0.9000	0.7389
Terr2	0.9000	0.7389
Terr3	1.0000	0.7895
Terr4	1.0000	0.7895
μ Terrier	**0.9500**	**0.7642**
Coper	0.0000	0.0000
Windw	0.0000	0.0000
μ D.S.	**0.0000**	**0.0000**
Solr1P	1.0000	1.0000
Solr2P	1.0000	0.9626
Solr3P	1.0000	0.9626
Solr4P	0.8000	0.5092
Solr1V	1.0000	0.7154
Solr2V	1.0000	0.7073
Solr3V	1.0000	0.7073
Solr4V	1.0000	0.9962
μ Solr	**0.9750**	**0.8201**

Tabella 5.10. Metriche Eventi Topic 1

Run	p@10	AP
Terr1	0.6000	0.6358
Terr2	0.6000	0.6406
Terr3	1.0000	0.8734
Terr4	1.0000	0.8516
μ Terrier	**0.8000**	**0,7503**
Coper	1.0000	0.8844
Windw	1.0000	0.8844
μ D.S.	**1.0000**	**0.8844**
Solr1P	1.0000	0.8201
Solr2P	1.0000	0.8804
Solr3P	1.0000	0.8758
Solr4P	1.0000	0.8807
Solr1V	1.0000	0.9261
Solr2V	1.0000	0.9398
Solr3V	1.0000	0.9398
Solr4V	1.0000	0.9332
μ Solr	**1,0000**	**0,8995**

Tabella 5.11. Metriche Eventi Topic 2

Run	p@10	AP
Terr1	0.8000	0.6541
Terr2	0.9000	0.5899
Terr3	0.9000	0.6357
Terr4	0.9000	0.6393
μ Terrier	**0,8750**	**0.6298**
Coper	0.3000	0.1709
Windw	0.3000	0.1709
μ D.S.	**0.3000**	**0.1709**
Solr1P	0.8000	0.4427
Solr2P	0.9000	0.5120
Solr3P	0.9000	0.5120
Solr4P	0.2000	0.2275
Solr1V	0.6000	0.3895
Solr2V	0.9000	0.5029
Solr3V	0.9000	0.5029
Solr4V	0.3000	0.2850
μ Solr	**0.6875**	**0.4218**

Tabella 5.12. Metriche Eventi Topic 3

Run	p@10	AP
Terr1	0.1000	0.2444
Terr2	0.7000	0.5373
Terr3	0.6000	0.4060
Terr4	0.9000	0.6928
μ Terrier	**0.5750**	**0.5750**
Coper	0.0000	0.0000
Windw	0.0000	0.0000
μ D.S.	**0.0000**	**0.0000**
Solr1P	0.7000	0.5698
Solr2P	0.7000	0.5709
Solr3P	0.7000	0.5709
Solr4P	0.3000	0.0807
Solr1V	1.0000	0.6817
Solr2V	1.0000	0.6413
Solr3V	1.0000	0.6413
Solr4V	0.4000	0.1224
μ Solr	**0.7250**	**0.4849**

Tabella 5.13. Metriche Eventi Topic 4

Run	p@10	AP
Terr1	0.4000	0.2781
Terr2	0.4000	0.2781
Terr3	0.6000	0.7539
Terr4	0.6000	0.7539
μ Terrier	**0.5000**	**0.5160**
Coper	0.4000	0.1290
Windw	0.4000	0.1290
μ D.S.	**0.4000**	**0.1290**
Solr1P	0.6000	0.3584
Solr2P	0.6000	0.3999
Solr3P	0.6000	0.3999
Solr4P	0.6000	0.3335
Solr1V	0.9000	0.4631
Solr2V	0.6000	0.4115
Solr3V	0.6000	0.4115
Solr4V	0.4000	0.2863
μ Solr	**0.6125**	**0,3830**

Tabella 5.14. Metriche Eventi Topic 5

Capitolo 6

Conclusioni

In questa tesi si sono utilizzate in un contesto aziendale le conoscenze acquisite durante la formazione universitaria nell'ambito del reperimento dell'informazione. In collaborazione con l'azienda Siav si è realizzato un prototipo di un sistema di reperimento dell'informazione nella forma descritta nel capitolo 2. Nella prima fase di lavoro si è scelto un motore di ricerca adeguato da utilizzare per il caso in questione. L'analisi, presentata nel capitolo 3, ha messo a confronto due motori di ricerca, Solr ed Elasticsearch, ritenuti i migliori candidati per l'adempimento del compito. La corposa documentazione ha permesso di eseguire una comparazione tra i due motori di ricerca senza la necessità di testare in prima persona le caratteristiche indicate. Inoltre, questo confronto ha messo in evidenza che le differenze principali tra i due motori riguardano prevalentemente aspetti marginali che poco hanno a che vedere con il reperimento dell'informazione. Apache Lucene è un elemento comune ad entrambi i motori e questo comporta che le operazioni di indicizzazione e reperimento vengono svolte nello stesso modo. Quindi non si è giunti alla fine ad una scelta basata su un particolare merito ma, date le considerazioni fatte, ha prevalso la conoscenza pregressa, da parte dell'azienda, per la decisione su quale strumento utilizzare.

Il percorso per la realizzazione successiva del prototipo, descritto nel capitolo 4, è stato oggetto di una fase di analisi dei requisiti fatta in collaborazione con l'azienda, in particolare tenendo come riferimento il software Archiflow, da loro prodotto per l'archiviazione documentale. L'obiettivo è stato via via messo a fuoco dopo una serie di valutazioni, quali ad esempio

il concentrarsi su una specifica categoria di documenti per fare in seguito delle analisi mirate, o se considerare l'interezza della collezione; su quali documenti basarsi per la prototipazione, sulla possibilità o meno di considerare l'integrazione di altre sorgenti informative esterne con quelle di diretta pertinenza del software citato, quest'ultimo elemento divenuto strategico a fini di importanti scelte progettuali successive.
Difatti la scelta finale è stata quella di considerare, almeno da un punto di vista prospettico, l'intera collezione di documenti memorizzati e di inserire nei risultati della ricerca anche altri elementi da sorgenti integrative. Si sono aggiunti nel processo di ricerca anche documenti denominati Eventi e si è testata la possibilità di integrare pagine web raccolte mediante il *web crawler* Nutch. Le modalità utilizzate per l'estrazione del testo dai documenti sono risultate soddisfacenti e hanno evidenziato la bontà del metodo di utilizzo degli *IFilter* ad eccezione dei documenti presenti in formato `.RTF`, dove si è utilizzato con maggior profitto un'apposita libreria basata su *Apache Tika* e di quelli in formato `.EML`, dove si è utilizzata una libreria fornita direttamente dall'azienda committente.
Diverse scelte progettuali sono state elaborate per quanto riguarda l'operazione di indicizzazione. Difatti, quattro differenti configurazioni sono state progettate, considerando varie opzioni di intervento sulle operazioni di rimozione delle *stop word* e *stemming*. Si è anche voluto realizzare, grazie ai lavori del progetto MultiWordNet, un dizionario di sinonimi da utilizzare per analizzare anche un meccanismo di espansione dei termini. Si è notato però che il filtro `solr.SynonymFilterFactory` applica, dopo l'espansione dei termini, una successiva fase di analisi di estrazione dei token portando alla ripetizione di alcuni termini. Anche per quanto riguarda i modelli di reperimento dei documenti, si sono considerate due diverse opzioni: il modello probabilistico BM25, divenuto dalla versione di Solr utilizzata il modello di default, e il modello vettoriale TF-IDF, che era il modello di default nelle versioni precedenti.

Nel capitolo 5 si è affrontato il lavoro di creazione di una collezione sperimentale. Questa è stata una decisione non immediata, visto le non poche difficoltà che presentava. Infatti ha richiesto una ulteriore fase di analisi per stabilire la numerosità e la tipologia di documenti da trattare, tenendo sempre presente la disponibilità di risorse umane ed economiche per l'operazione di giudizio della rilevanza dei documenti. Alla fine si è deciso di

considerare l'intero sottoinsieme dei documenti estratti da Archiflow, utilizzati per realizzazione del prototipo e tutti i documenti riferiti come Eventi. Il task di ricerca fissato è stato quello di ottenere il maggior numero di documenti considerati rilevanti nelle posizioni di testa (similmente a quanto accade nell'uso di un *web search engine*). Si sono definiti cinque topic, ovvero cinque esigenze informative, per ognuna delle due categorie considerate. Per l'operazione di assegnazione dei giudizi di rilevanza, la scelta di utilizzare la tecnica del *pooling* ha richiesto lo sforzo suppletivo di dover utilizzare altri motori di ricerca per disporre di ulteriori esperimenti dai quali estrarre nuovi documenti da valutare. Tra questi si sono utilizzati Terrier e due *desktop search*, Windows Search e Copernic; la scelta di utilizzare quest'ultimi due software si è rivelata negativa mentre, in contrapposizione, l'uso di Terrier è stata una scelta valida. I risultati ottenuti, valutati utilizzando le due metriche indicate per il task fissato, sono stati abbastanza significativi per quanto riguarda la collezione degli Eventi, mentre per quanto riguarda i documenti provenienti da Archiflow, gli esiti sono stati molto meno incoraggianti. Sostanzialmente si può affermare che Solr è risultato essere un motore di ricerca molto efficace.

Sviluppi futuri

Di seguito una lista di possibili interventi per un successivo miglioramento del prototipo progettato.

- Ampliare la collezione sperimentale, definendo un insieme più numeroso di topic, in modo da ottenere dei giudizi maggiormente indicativi.
- Utilizzare un dizionario di sinonimi più specifico e mirato al contesto applicativo, inserendo relazioni maggiormente legate al caso d'uso. Inoltre è consigliabile personalizzare l'applicazione del filtro apposito `solr.SynonymFilterFactory` o di realizzare un plugin apposito personalizzato per eseguire l'espansione dei termini.
- Verificare il funzionamento con altri modelli di reperimento e utilizzare le altre modalità di *parser* per le interrogazioni.
- Fare un'analisi schematica del testo estratto attraverso il metodo degli *Ifilter*, ricercando eventuali sequenze non appartenenti al contenuto

testuale ma frutto di una non corretta estrazione. Questa operazione porterebbe diversi benefici in quanto: permetterebbe di verificare la correttezza dell'estrazione, favorirebbe il miglioramento della lista di *stop word* integrando altri vocaboli poco significativi dal punto di vista del contenuto.

- Utilizzare la possibilità di personalizzare, attraverso `FieldType`, l'indice trasposto, favorendo o screditando alcuni specifici metadati, perché considerati poco utili o in contrapposizione particolarmente importanti ai fini dell'ordinamento dei risultati.
- Aumentare le sorgenti informative, integrando ad esempio dei *social network*, in particolare Twitter e LinkedIn, considerati strategici dal committente.

Bibliografia

Agosti, M. (2014/2015 ' 2015/2016). "Documentazione del Corso di reperimento dell'informazione, Università degli Studi di Padova".

Angelini, M. et al. (2014). "VIRTUE: A visual tool for information retrieval performance evaluation and failure analysis." In: *J. Vis. Lang. Comput.* 25.4, pp. 394–413. URL: http://dblp.uni-trier.de/db/journals/vlc/vlc25.html#AngeliniFSS14.

Chang, Yu-Wei (2016). "Influence of the principle of least effort across disciplines". In: *Scientometrics* 106.3, pp. 1117–1133. DOI: 10.1007/s11192-016-1838-0. URL: http://dx.doi.org/10.1007/s11192-016-1838-0.

Croft, W. Bruce, D. Metzler e T. Strohman (2010). *Search engine: Information Retrieval in Practice*. USA: Pearson Education, Inc. ISBN: 9780136072249.

Demartini, G. (2005). "Una metrica di valutazione per l'Information Retrieval: Analisi Critica e sperimentazioni". Tesi di laurea magistrale. Università degli studi di Udine.

Feldman, S. e C. Sherman (2003). "The high cost of not finding information". In: *IDC White Paper*.

— (2009). "The Information Advantage: Information Access in Tomorrow's Enterprise". In: *IDC, Adapted from Hidden Costs of Information Work: A Progress Report by Susan Feldman, IDC 217936, and from Worldwide Search and Discovery Software 2009–2013 Forecast Update and 2008 Vendor Shares by Susan Feldman, IDC 219883.*

Ferro, N. et al. (2016). "The twist measure for IR evaluation: Taking user's effort into account". In: *Journal of the Association for Information Science and Technology* 67.3, pp. 620–648. ISSN: 2330-1643. DOI: 10.1002/asi.23416. URL: http://dx.doi.org/10.1002/asi.23416.

Fielding, R. T. (2000). "Architectural Styles and the Design of Network - based Software Architectures". AAI9980887. Tesi di dottorato. University of California, Irvine. ISBN: 0-599-87118-0.

Garzonio, J. (2012). "Documentazione del corso di Morfologia Linguistica e Glottologia, Università degli Studi di Ferrara".

Gheorghe, R., M. Lee Hinman e R. Russo (2015). *Elasticsearch in Action*. USA: Manning Publications. ISBN: 9781617291623.

Grainger, T. e T. Potter (2014). *Solr in Action*. USA: Manning Publications. ISBN: 9781617291029.

Gulli, A. e A. Signorini (2005). "The indexable web is more than 11.5 billion pages". In: *Proceedings of the 14th international conference on World Wide Web, WWW 2005, Special interest tracks and posters, pp 902–903*. DOI: 10.1145/1062745.1062789. URL: http://doi.acm.org/10.1145/1062745.1062789.

Hawking, D. (2004). "Challenges in Enterprise Search". In: *Proceedings of the Australasian Database Conference ADC2004*. Invited paper: http://david-hawking.net/pubs/hawking_adc04keynote.pdf. Dunedin, New Zealand, pp. 15–26.

— (2010). *Enterprise Search*. A cura di Ricardo Baeza-Yates e Berthier Ribeiro-Neto. http://david-hawking.net/pubs/ModernIR2_Hawking_chapter.pdf. UK: Pearson Educational, pp. 641–684. ISBN: 978-0-321-41691-9.

Marcato, A. (2010). "La Ricerca a Faccette. Aspetti teorici e pratici". Tesi di laurea triennale. Dipartimento di Ingegneria Informatica, Università degli studi di Padova.

Melucci, M. (2013). *Infomation Retrieval: Metodi e modelli per i motori di ricerca*. Milano: Franco Angeli. ISBN: 9788820421120.

Mukherjee, R. e J. Mao (2004). "Enterprise Search: Tough Stuff". In: *Queue - Search Engines: Volume 2 Issue 2, April 2004*.

Pianta, E., L. Bentivogli e C. Girardi. In: *MultiWordNet: Developing and Aligned Multilingual Database. In Proceedings of the First International Conference on Global WordNet, pp. 293-302, 2002.*

Voorhees, E. M. (2000). "Variations in relevance judgments and the measurement of retrieval effectiveness". In: *Inf. Process. Manage.* 36.5, pp. 697–716. DOI: 10.1016/S0306-4573(00)00010-8. URL: http://dx.doi.org/10.1016/S0306-4573(00)00010-8.

Zobel, J. (1998). "How Reliable Are the Results of Large-Scale Information Retrieval Experiments?" In: *SIGIR '98: Proceedings of the 21st Annual International ACM SIGIR Conference on Research and Development in Information Retrieval, August 24-28 1998, Melbourne, Australia*, pp. 307–314. DOI: 10.1145/290941.291014. URL: http://doi.acm.org/10.1145/290941.291014.

Sitografia

Apache solr reference guide. http://apache.panu.it/lucene/solr/ref-guide/apache-solr-ref-guide-6.0.pdf.

Apache Solr vs Elasticsearch, The Feature Smackdown. http://solr-vs-elasticsearch.com/.

BM25 The Next Generation of Lucene Relevance. http://opensourceconnections.com/blog/2015/10/16/bm25-the-next-generation-of-lucene-relevation/.

DB-Engines. http://db-engines.com/en/ranking/search+engine.

Elasticsearch Reference 2.3. https://www.elastic.co/guide/en/elasticsearch/\reference/current/index.html.

Side by Side with Elasticsearch and Solr. https://youtu.be/LA-rbmuSROM, http://www.slideshare.net/sematext/side-by-side-with-elasticsearch-and-solr.

Side by Side with Elasticsearch and Solr - Part 2. https://youtu.be/01mXpZOF-_o, http://www.slideshare.net/sematext/side-by-side-with-elasticsearch-solr-part-2.

Solr open source in Apache Software Foundation. https://issues.apache.org/jira/browse/SOLR-1.

Solr vs ElasticSearch. https://thinkbiganalytics.com/solr-vs-elastic-search/.

Solr vs. Elasticsearch - Case by Case. http://www.slideshare.net/arafalov/solr-vs-elasticsearch-case-by-case, https://www.youtube.com/watch?v=S1Md3LDJPLs.

Solr vs. Elasticsearch — How to Decide? https://sematext.com/blog/2015/01/30/solr-elasticsearch-comparison/.