Politecnico di Torino

Dipartimento di Automatica e Informatica

Corso di Laurea Magistrale in Ingegneria Informatica

Elaborazione delle richieste in linguaggio naturale (NLP) nell'ambito dell'enterprise search e degli agenti conversazionali

---

Laureando:
Marco Montebello

Relatore:
Prof. Riccardo Sisto

Anno Accademico 2017/2018

Indice

1 Introduzione **4**

2 Stato dell'arte **7**

2.1 Information Retrieval e i motori di ricerca 7

2.1.1 Architettura di un sistema di Information Retrieval . . . 8

2.1.2 Modelli e algoritmi per l'IR 8

2.1.3 I Web Crawler 11

2.1.4 Preprocessamento dei documenti 13

2.1.5 Indicizzazione 15

2.2 NLP (Natural Language Processing) 17

2.2.1 Analisi Morfologica e POS-tagging 19

2.2.2 Analisi Sintattica 23

2.2.3 Apache OpenNLP 26

2.3 Elastic Search 27

2.3.1 Distributed Search 28

2.3.2 Query Routing 28

2.3.3 Real-time 29

2.3.4 Analizzatori 29

2.3.5 Valutazione 30

3 Descrizione del sistema **31**

3.1 Architettura del sistema 32

3.2 Tecnologie utilizzate 32

3.3 Moduli 33

3.3.1 Node-Red 33

3.3.2 IBM Conversation 38

3.3.3 Watson Content Analytics Studio 45

3.3.4 Watson Content Analytics 49

3.3.5 Il problema della BoilerPlate extraction 55

3.3.6 Service Layer 56

4 Sperimentazione **71**
4.1 Set di Frasi 71
4.2 Valutazione del Search Engine 72
4.3 Risultati 74

5 Conclusioni e Sviluppi futuri **77**
5.1 Conclusioni 77
5.2 Sviluppi Futuri 78
5.2.1 Adattamento ad altri Domini Enterprise 78
5.2.2 Nuove Funzionalità 79

Capitolo 1

Introduzione

Il presente lavoro nasce dall'esperienza del tesista presso Blue Reply S.r.l e ha come oggetto la realizzazione di un progetto proposto da un cliente in ambito assicurativo, che ha espresso la necessità di avere un motore di ricerca all'interno del proprio portale in grado di rispondere a richieste formulate in linguaggio naturale.

Non è una novità che una grande compagnia indicizzi i propri contenuti per renderli fruibili alla propria utenza in maniera immediata, ma questo tipo di interazione richiede, a volte, l'attenzione da parte dell'utente di seguire una sintassi spesso restrittiva, con termini e concetti ben precisi e talvolta esclusivi del dominio trattato (per lo più sconosciuti all'utente). Bisogna poi sottolineare che, grazie all'introduzione delle nuove tecnologie, l'utilizzo di internet è esploso anche per quella parte d'utenza che generalmente non ha competenze adeguate in campo digitale e che quindi non è in grado di fruire servizi di questo tipo; da quì la necessità, per aziende come quella del cliente di Blue, di consentire all'utente finale un'interazione più naturale che gli permetta di concentrarsi solo sull'espressione del proprio bisogno e non sulla modalità di espressione.

Ciò che permette uno scenario del genere oggi è offerto dalle tecniche di elaborazione del linguaggio naturale (meglio conosciute come NLP) ossia quella branca della scienza informatica o meglio dell'intelligenza artificiale che da decenni ha interessato gli scienziati e che al giorno d'oggi è tornata in auge, di cui continua ad essere un problema centrale: basti pensare a Google Translate e al sistema di riconoscimento vocale della Apple, Siri, che non sarebbero esistiti senza l'interazione uomo-macchina veicolata dall'NLP e che sebbene innovativi sono ben lontani dall'essere perfetti.

Il Natural Language Processing abilita funzionalità quali: Text classification, Traduzione automatica, Comprensione del linguaggio parlato, Infor-

mation extraction, Automatic summarization, Question answering, Acquisizione o estrazione della conoscenza. Quest'ultimo punto è stato fondamentale per la riuscita del progetto ed è stato uno dei punti cruciali sul quale si è incentrato il lavoro di Tesi.

Obiettivo del lavoro è stato, infatti, prima di tutto quello di realizzare un motore di ricerca che collezionasse ciò che è richiesto dal dominio e poi quello di utilizzare le tecniche di elaborazione del linguaggio naturale per analizzare l'intero corpus del motore stesso e fargli acquisire quella conoscenza necessaria a differenziare il comportamento di un qualsiasi search-engine da uno capace di ricevere richieste in linguaggio naturale e in grado di "capire" di cosa stia parlando l'utente che esprime la propria richiesta.

Stesso trattamento è riservato alle query dell'utente che vanno espresse in una sintassi per cui il search engine è predisposto e in modo che anche essa goda della conoscenza del dominio specifico. Come si evincerà dal seguito, il meccanismo di annotazione del contenuto è stato fondamentale a tal fine.

La scelta degli strumenti per l'implementazione della soluzione è ricaduta, dopo le dovute analisi, su Watson Content Analytics (WCA), la piattaforma di ricerca e analisi di IBM® che include soluzioni per indicizzare dati e contenuti strutturati e/o non strutturati da sorgenti dati differenti come database, tool di collaborazione e siti Web. Offre gli strumenti fondamentali che un search engine deve offrire come Crawler, Parser e Indexer ma la potenza di WCA è data dalle soluzioni di analisi del testo basate sull'utilizzo di una pipeline di base per l'elaborazione del testo estendibili con delle funzionalità custom.

Esistono numerose alternative per l'editing delle pipeline di annotazione e la stessa suite di Watson offre Content Analytics Studio, un ambiente di sviluppo per la costruzione e il test di motori di analisi del testo avanzati per uno specifico business-domain.

Altro punto focale della progettazione è stato lo studio di soluzioni al problema del boilerplate removal, ovvero dell'estrazione del contenuto principale delle pagine; contenuti quali header e footer, presenti in ogni pagina web, influenzano o possono influenzare la creazione dell'index e quindi i risultati della ricerca poiché contenenti concetti del dominio.

Ciò che è emerso e che verrà esposto dettagliatamente nel capitolo finale mostra un notevole miglioramento nel comportamento del motore di ricerca che risponde restituendo documenti maggiormente rilevanti alle richieste dell'utente.

Per rendere fruibile l'intero sistema mediante un'interfaccia più intuitiva di quella del form di un portale web si è sviluppato anche un bot, o agente conversazionale, per Telegram in grado di simulare una conversazione naturale con l'utente e che accetta richieste in linguaggio naturale dagli utenti della chat.

Capitolo 2

Stato dell’arte

Information Retrieval, search engine e NLP sono attualmente oggetto di importanti ricerche. Questo capitolo inizia con l’introduzione dei concetti chiave su queste tematiche per fornire una comprensione di base su come lavorano i motori di ricerca e le tecniche NLP e su cosa si fondano gli strumenti che, si vedrà, sono stati scelti per l’implementazione del progetto. A seguire verranno presentati due casi di studio open-source e indicativi dello stato dell’arte in questi due settori: Elastic search, l’enterprise search-engine full text più popolare[1] e Apache openNLP, un toolkit machine-learning completo per il processamento del linguaggio naturale.

2.1 Information Retrieval e i motori di ricerca

Per via della loro immediatezza e della quantità immensa di informazioni di cui riescono a tenere traccia oggi, i motori di ricerca e il Web search più in generale, sono di fatto le metodologie dominanti nello scenario della ricerca di informazioni ed entrambi fanno parte dell’Information Retrieval (IR), quel campo di studio che ha come scopo la soddisfazione del "bisogno informativo dell’utente" ossia quel bisogno che un utente ha di trovare le informazioni di cui necessita da una vasta collezione di documenti. Il Retrieval è classificato come *information pull*: l’utente estrae le informazioni mediante delle richieste a differenza dell’ *information push* per cui le informazioni sono restituite in anticipo rispetto ai bisogni dell’utente che non sollecita direttamente il sistema.

2.1.1 Architettura di un sistema di Information Retrieval

Bing Liu [2, capther 6, p. 212] definisce l'IR come lo studio dell'acquisizione, organizzazione, memorizzazione, estrapolazione e distribuzione delle informazioni la cui unità di base è il documento. In IR un documento indica un contenitore persistente di dati identificabile in modo univoco; l'identificatore consente l'accesso al documento indipendentemente dal suo contenuto e permette, ad esempio, che due documenti identici possano essere trattati come elementi distinti. La figura 2.1 rappresenta lo schema generico di un sistema di Information Retrieval: un sistema informatico o la parte di un sistema informatico progettato e realizzato per svolgere in modo automatico compiti di IR [3]. L'*utente* che genera una *query* di ricerca la fornisce al sistema di recupero (*retrieval system*) tramite il modulo (*query operations*) che la pre-processa generandone una versione eseguibile. In base al proprio modello, questo modulo accetta query di diverso tipo:con le *keyword query* ad esempio gli utenti esprimono la loro richiesta con una lista di termini allo scopo di trovare documenti che ne contengano almeno uno; con le query di tipo booleano si esprime volontariamente la richiesta utilizzando operatori logici quali OR o AND; la query in linguaggio naturale invece corrisponde al tipo di query più complesso ma anche quello ideale.

I documenti, mediante il *crawler*, vengono raccolti nella *document collection* e successivamente processati da un *indexer* per la generazione dell'indice (*Document Index*). Il sistema si serve di questo per il recupero dei documenti rilevanti (contenenti parte o la totalità dei termini della query fornita) ordinandoli in una classifica che è stilata sulla base di punteggi ottenuti dagli algoritmi scelti dal sistema stesso. Questi algoritmi, in genere, prendono in considerazione tantissimi fattori quali la frequenza delle parole più ricercate all'interno di un documento, la presenza o meno delle parole che formano la query nel titolo, i sinonimi delle parole ricercate o la presenza del documento in particolari black-list. La pagina con i risultati della ricerca è la classifica finale, chiamata **SERP** (Search Engine Results Page) e stilata in modo decrescente dal contenuto migliore (voto più alto) al peggiore (voto più basso).

2.1.2 Modelli e algoritmi per l'IR

I modelli su cui si costruiscono i sistemi di Information Retrieval si differenziano per le modalità di rappresentazione delle query, dei documenti e

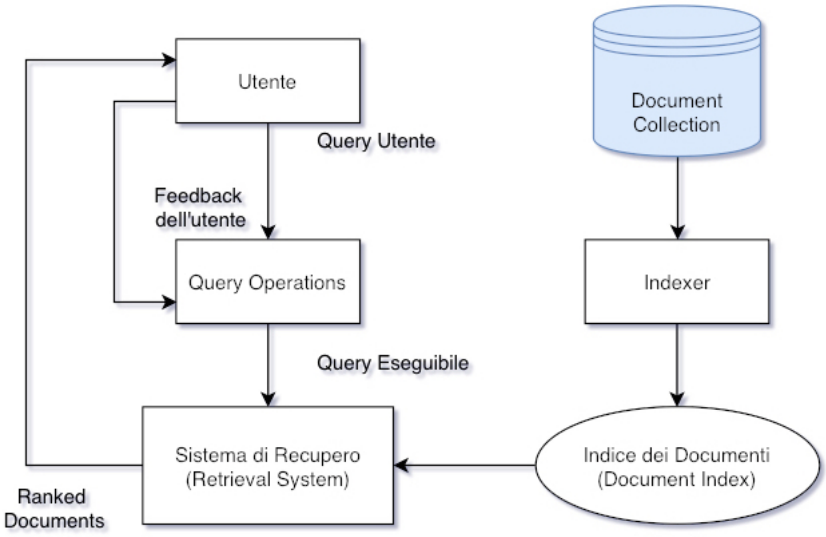


Figura 2.1: Architettura generica di un sistema IR

per il relativo meccanismo di recupero. I modelli *non strutturali* trattano i documenti e le query come un insieme di termini il quale ordine è superfluo. Data una collezione di documenti D, sia:

$$V = \{t_1, t_2, \ldots, t_{|v|}\} \tag{2.1}$$

il set dei termini nella collezione detto *dizionario*. Ad ogni termine del dizionario, per ogni documento, è associato un peso $w_{i,j}$ che indica la sua presenza o meno e la sua importanza all'interno del documento; quest'ultimo è rappresentato sotto forma di vettore di termini:

$$d_j = (w_{1j}, w_{2j}, \ldots, w_{|V|j}) \tag{2.2}$$

dove ogni peso $w_{i,j}$ corrisponde al termine $t_i \in V$ e quantifica il livello di importanza di t_i nel documento d_j. Esistono anche altri modelli detti non classici o *strutturati* che non condividono le assunzioni precedenti. Tra i più importanti modelli non strutturati per l'IR figurano [4]:

Modello booleano: è un modello basato sulla teoria degli insiemi. I pesi dei termini rispetto ai documenti hanno valori binari: $w_{i,j} \in 0,1$ dove 1 indica che il termine è presente nel documento mentre 0 indica la sua assenza. Data una query in formato booleano il sistema restituisce solo i documenti che soddisfano la query sulla base del criterio di decisione binaria; in un sistema del genere un documento può essere solamente *rilevante* o *non rilevante*. La semantica è precisa ma non c'è una nozione di matching parziale e non permette quindi il ranking.

Modello vettoriale: è conosciuto anche come modello **bag-of-words** ed è forse il modello più largamente utilizzato nei motori di ricerca attuali; il documento continua ad essere rappresentato da un vettore di pesi non più vincolati ad essere solamente $\in 0,1$ ma che possono avere un valore qualsiasi ricavato da variazioni dello schema TF o TF-IDF. Lo schema TF pesa i termini secondo la loro frequenza nel documento $f_{i,j}$ (la frequenza del termine t_i nel documento d_j) o, spesso, secondo lo stesso fattore normalizzato su tutti i termini che appaiono nel documento:

$$tf_{ij} = \frac{f_{ij}}{\max\left\{f_{1j}, f_{2j}, \ldots, f_{|V|j}\right\}} \tag{2.3}$$

Secondo lo schema TF, più sono frequenti i termini in un documento, più importanti essi tendono ad essere per il contenuto; lo schema IDF (Inverse Document Frequency), invece, considera i termini che appaiono in molti documenti differenti meno indicativi del contenuto e pesa i termini mediante il fattore idf_i:

$$idf_i = log\frac{N}{df_i} \tag{2.4}$$

con N il numero dei documenti presenti nella collezione e df_i il numero di documenti che contengono il termine i. Infine, lo schema TF-IDF pesa i termini secondo la:

$$w_i j = tf_{ij} \times idf_i \tag{2.5}$$

per la quale un termine che occorre frequentemente in un documento ma raramente nell'intera collezione riceve un peso alto. Il vettore di rappresentazione di una domanda e' tipicamente trattato come un documento e pesato tramite il fattore tf-idf. In alternativa sarebbe possibile fornire pesi assegnati dagli utenti ai singoli termini della query. Questo modello, a differenza del modello booleano, non effettua un matching perfetto delle regole di rilevanza ma i documenti sono ordinati in accordo al loro grado di rilevanza con la query. Un modo di calcolare il grado di rilevanza è calcolare la similarità della query **q** ad ogni documento d_j nella collezione D. Esistono diverse tecniche per il calcolo della similarità quali il metodo Okapi, del *pnw* (*pivoted normalization weighting* [5]) e il metodo della similarità del coseno (cosin similarity) che è il maggiormente utilizzato. Nel metodo della similarità del coseno, dati i due vettori **d** e **q**, la loro similarità è data dal coseno dell'angolo che essi formano:

$$cosine(d_j, q) = \frac{\langle d_j \cdot q \rangle}{\|d_j\| \times \|q\|} = \frac{\sum_{i=1}^{|V|} w_{ij} \times w_{iq}}{\sqrt{\sum_{i=1}^{|V|} w_{ij}^2} \times \sqrt{\sum_{i=1}^{|V|} w_{iq}^2}} \tag{2.6}$$

Il coseno è calcolato come il rapporto tra il prodotto scalare dei due vettori e il prodotto tra le loro norme. Nella equazione 2.6 |V| è la dimensione del dizionario della collezione, w_{ij} il peso del termime i all'interno del documento j mentre w_{iq} il peso all'interno della query. Il ranking dei documenti è fatto usando il loro valore di similarità.

Modelli probabilistici: é un'altra classe di modelli utilizzata dai sistemi di IR e che ha le sue radici nel calcolo delle probabilità e nella statistica. La realizzazione del ranking nei modelli probabilistici richiede la stima di un modello di linguaggio per ogni documento della collezione e successivamente l'ordinamento dei documenti secondo il criterio di verosimiglianza dato il modello stesso. Il criterio di verosimiglianza e in genere le tecniche probabilistiche sulle quali si basa questa classe di modelli sono largamente utilizzate anche nel processamento del linguaggio naturale. Data una query, vogliamo trovare i documenti che hanno la maggior probabilità di essere rilevanti. Lo scopo quindi è quello di calcolare:

$$P(R = true|D, Q) \tag{2.7}$$

dove D rappresenta il documento di cui si vuole calcolare la rilevanza data la query, Q è la query stessa ed R una variabile casuale booleana che rappresenta la rilevanza. Fatto questo è possibile presentare i risultati sotto forma di lista ordinata sulla probabilità di rilevanza.

2.1.3 I Web Crawler

A differenza dei Crawler presenti nei classici sistemi di IR, i Web Crawler, chiamati anche *Spider* o *Robot* sono dei software specializzati nella raccolta di interi siti Web che nella loro implementazione più semplice (*General Purpose Crawler*) scaricano tutti i documenti raggiungibili a partire da un set di pagine di base dette *seed* seguendo i link all'interno delle pagine attraversate; altri Spider recuperano solo i documenti appartenenti ad un particolare topic (*Focused Crawler*) con lo stesso meccanismo. Per fare ciò, il Crawler si appoggia alle operazioni di *Fetch* e di *Parse*: durante la fase di Fetch lo Spider si comporta da client HTTP generando delle GET Request rivolte al server che ospita le pagine e processandone poi le risposte; nella fase di Parse invece si analizza il codice HTML scaricato dal Fetcher per la costruzione di un albero DOM che ne rappresenti la struttura. La figura 2.2 rappresenta i task che un Crawler deve eseguire in sequenza per eseguire correttamente la raccolta di una pagina Web. Così come un sito Web può

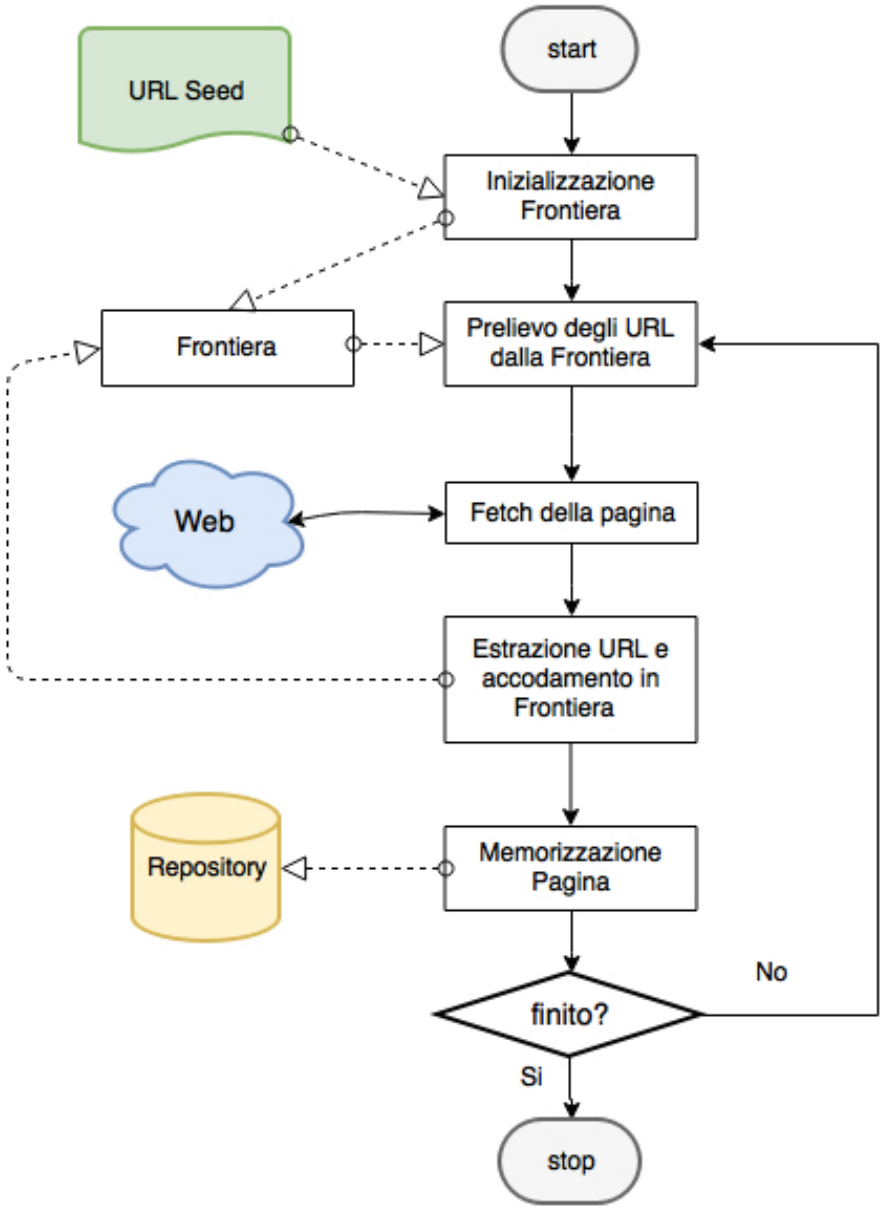


Figura 2.2: Flow chart di un crawler sequenziale di base [2]

essere visto al pari di un grafo in cui le pagine sono i nodi e i link gli archi, un Crawler ha un comportamento riconducibile a quello di un algoritmo di ricerca su di un grafo: esso mantiene una lista di URL da visitare, chiamata *frontiera*, inizializzata con i *seed URL* da cui partire; ad ogni iterazione, preleva un URL dalla frontiera e mediante richiesta HTTP effettua il fetch della pagina corrispondente, poi ne effettua il parsing; mediante il *parser* il crawler estrae il contenuto e quindi la lista degli URL a cui questa punta per aggiungerli alla frontiera, infine ne salva il contenuto in una *repository* interna.Il ciclo continua fino allo svuotamento della frontiera o finché non viene raggiunto un limite impostato a priori. L'algoritmo con cui viene visitata la frontiera del crawler caratterizza le diverse tipologie di crawler: il *Breadth-first crawler* realizza la frontiera mediante una coda FIFO dove il successivo URL da processare è puntato dalla sua testa mentre i nuovi URL vengono aggiunti in coda; possono essere aggiunti alla coda solo pagine non ancora visitate è perciò necessario mantenere una struttura dati che tiene conto delle pagine già visitate dal crawler. I *Depth-first crawler* utilizzano l'algoritmo di ricerca depth-first dove partendo dalla radice (seed) si esplorano tutti i rami per tutta la loro lunghezza prima di proseguire con l'esplorazione di un altro ramo; nei *Preferential crawlers* invece, la frontiera è implementata mediante una coda a priorità dove ad ogni link non ancora visitato viene assegnata una priorità in funzione del valore stimato del valore di un link.

2.1.4 Preprocessamento dei documenti

Prima che un documento sia reso disponibile per il recupero delle informazioni passa attraverso una serie di task che hanno l'obiettivo di eliminare informazioni superflue ai fini della ricerca e mantenere invece quelle fondamentali. Questa serie di task è generalmente implementata mediante una pipeline che prevede operazioni quali rimozione delle stop word, lo stemming e la gestione di caratteri speciali per tipi di testo generico mentre prevede operazioni aggiuntive per le pagine web, di natura differente, e che richiedono ad esempio dei task di rimozione dei tag HTML e identificazione del main content block, ovvero della porzione di pagina associata al contenuto informativo principale di quella particolare pagina Web. La maggior parte di queste tecniche è anche oggetto delle tecniche di NLP, che verranno esposte nel paragrafo successivo.

Stop-Word Removal Le stop-word sono parole utilizzate frequentemente di natura generica e accessoria rispetto alle parole principali di un vocabolario e che non aggiungono alcuna valenza semantica al contesto testuale; tutti i linguaggi forniscono esempi di questa particolare categoria di parole; si pensi agli articoli oppure alle congiunzioni e le preposizioni. Poiché i motori di ricerca hanno come scopo anche quello di classificare le pagine web sulla base del loro contenuto, essi stessi devono essere istruiti a gestire correttamente le stop word.

Stemming Lo stemming è il processo di riduzione della forma flessa di una parola alla sua forma radice, detta *tema* (*stem* in inglese). Il tema non corrisponde necessariamente alla radice morfologica (*lemma*) della parola: normalmente è sufficiente che le parole correlate siano mappate allo stesso tema (che ad esempio andare, andai, andò mappino al tema and), anche se quest'ultimo non è una valida radice per la parola. La creazione di un algoritmo di stemming è stato un annoso problema dell'informatica. Il processo di stemming è utilizzato nei motori di ricerca per l'espansione di interrogazioni e in altri problemi di elaborazione del linguaggio naturale[6].

Nel contesto di un sistema IR, tutte le parole di ogni testo della collezione vengono sostituite con le relativi radici; il risultato è una nuova versione del documento che presenta lo stesso numero di termini ma con meno varianti. Lo stesso trattamento viene riservato anche alle query utente e in questo modo il processo di ricerca e di matching delle informazioni è più efficace poiché la radice espande la selezione su più documenti rilevanti. E' possibile realizzare diversi livelli di stem: aumentando il livello la radice diventa più corta e si amplia il numero di termini che realizzano un match con la radice; al contrario riducendo il grado di stem aumenta la lunghezza della radice e si riduce la quantità di parole che realizzano il matching. Esistono alcuni motori di ricerca che utilizzano il lemma al posto dello stem e il motore di analisi del testo di Watson Content Analytics è uno di questi.

Pre-processamento di pagine Web Le pagine web per via della loro natura necessitano di ulteriori task di pre-processing. E' utile il riconoscimento di particolari campi di testo per far si che termini presenti in diversi campi siano caratterizzati da un diverso peso; ad esempio, i termini presenti all'interno del titolo di una pagina hanno molta più importanza di quelli presenti in una barra di collegamenti ad altre pagine. Task fondamentali sono anche il riconoscimento del blocco informativo principale per l'eliminazione del contenuto superfluo, noto come problema del boilerplate, e della rimo-

zione dei tag HTML per la fruizione del contenuto in una rappresentazione più naturale; esistono diverse librerie disponibili per la maggior parte dei linguaggi oggi presenti che semplificano i task di cui sopra.

2.1.5 Indicizzazione

Un'opzione per la ricerca in un sistema IR, sebbene impraticabile per collezioni molto grandi, sarebbe quella di scandire documento dopo documento l'intero database alla ricerca di quelli che contengono i termini introdotti dalla query utente. Oggi, però, la tecnica di indicizzazione considerata come la più efficace per il recupero dei documenti testuali di interesse è l'*Indice Invertito*; questo tipo di indice anziché rappresentare tutte le parole presenti in un documento, come nel caso di un indice semplice, rappresenta tutti i documenti in cui appare una specifica parola (per ogni parola di ogni documento).

Dato il set di documenti

$$D = \{d_1, d_2, \ldots, d_N\} \tag{2.8}$$

l'Inverted Index associato alla collezione D consiste di due parti:

1. Un vocabolario V (2.1) contenente tutti i termini distinti nella collezione (si veda la fig.2.3)

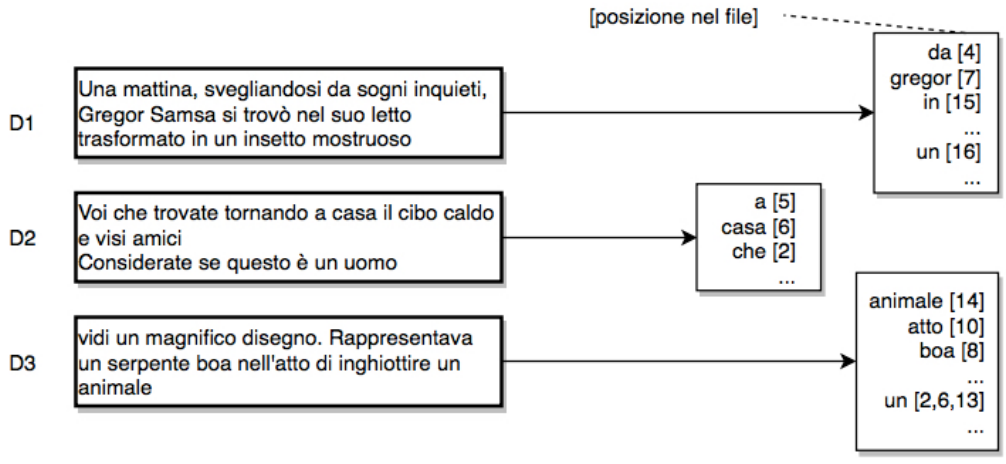


Figura 2.3: Costruzione del vocabolario di termini a partire dal contenuto dei documenti in un sistema di IR[7]

2. Una lista invertita per ogni termine nel vocabolario e ad ognuno dei quali è associata una lista di attributi (detta anche di *posting*) nella forma: $< id_j, f_{ij}, [o_1, o_2, \ldots, o_{|fij|}] >$. id_j è l'id del documento che contiene il termine t_i, f_{ij} è la frequenza con cui appare il termine t_i nel documento d_j e o_j sono gli offset dei termini in d_j; una rappresentazione del genere permette di includere negli indici non solo termini ma in alternativa anche altri pezzi di informazioni che possono essere utili in funzione dell'algoritmo di ranking o di retrieval (si veda la fig.2.4).

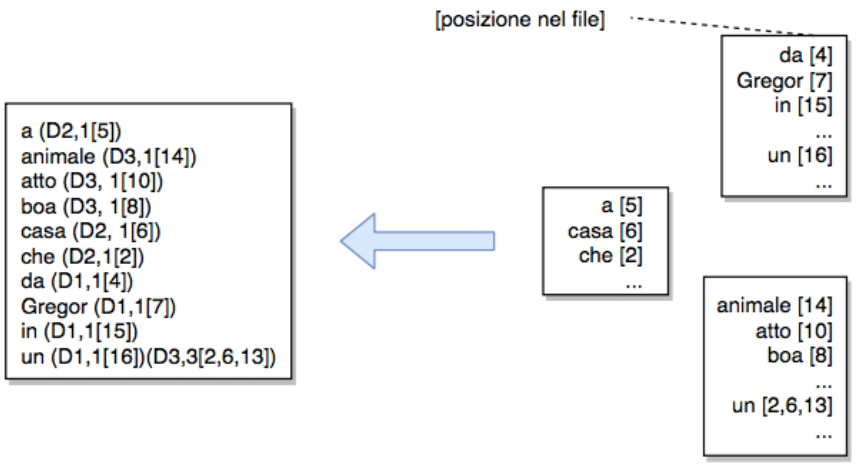


Figura 2.4: Generazione degli indici in un sistema di IR, ad ogni elemento del vocabolario vi è associata una lista invertita [7]

La costruzione dell'indice può essere realizzata utilizzando una qualsiasi tra le tante strutture dati considerate efficienti per la ricerca (K-d Tree, B+ tree, R tree, Grid files). Per aggiungere un nuovo termine all'indice lo si scandisce sequenzialmente fino alla fine o fino a un'occorrenza del termine stesso; nel primo caso si aggiunge un nuovo elemento alla struttura dati per la rappresentazione del termine altrimenti si accodano id del documento ed eventuali altre informazioni sull'elemento (ad esempio l'offset) all'indice invertito del termine. Data una query, l'interrogazione dell'indice (fig.2.5) comporta la ricerca di ogni termine all'interno dell'indice invertito e l'estrazione della posting list associata ad ognuno di essi. Ci si ritrova così ad avere un insieme di liste, una per ogni termine presente nella query; realizzandone il merge e calcolandone l'intersezione si ottiene l'insieme dei documenti che contengono tutti o parte dei termini della query. In conclusione si calcola un rank sulla base delle funzioni di rilevanza scelte da chi ha progettato il sistema (somiglianza del coseno, Okapi). La ranking

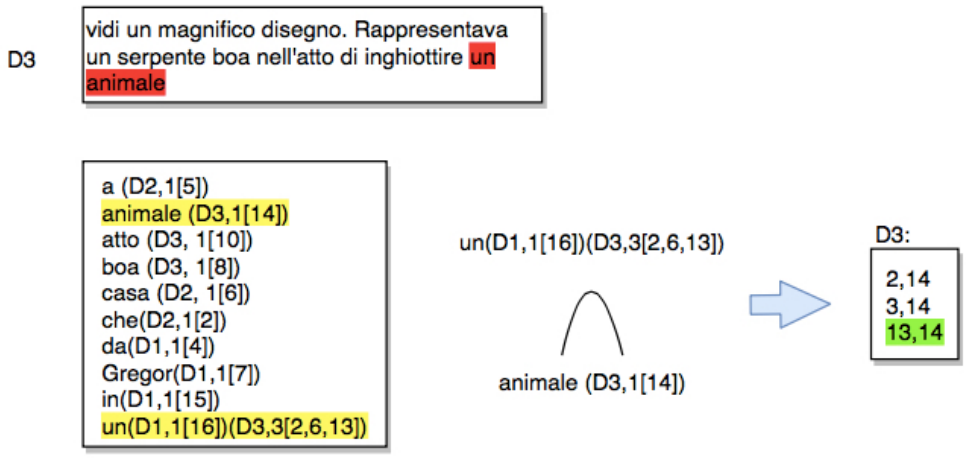


Figura 2.5: Interrogazione di un sistema IR con Inverted Index [7]

function OkapiBM25 rappresenta lo stato dell'arte tra le funzioni di ranking **TF-IDF-like**.

Sharding Quando il numero dei documenti indicizzati cresce, la *posting list* per alcuni termini può risultare significativamente estesa e sebbene il tempo di look-up per i termini nell'indice invertito è costante, il tempo per scorrere la posting list cresce linearmente comportando un potenziale basso throuput del sistema. Operazioni come la deframmentazione possono aiutare a mantenere le performance a breve termine ma la soluzione migliore prevede la suddivisione dell'indice in parti più piccole mediante l'operazione che prende il nome di *sharding*. Gli *shard* possono essere distribuiti su differenti nodi abilitando di fatto la possibilità di ricercare in modalità distribuita. Gli shard sono molto noti nel contesto dei database e si riferiscono al partizionamento orizzontale (per righe) delle tabelle in entità indipendenti e filtrabili mediante query. Nei sistemi di Information Retrieval il partizionamento avviene per documenti.

2.2 NLP (Natural Language Processing)

Il linguaggio è stato e continua ad essere studiato da specialisti di differenti discipline quali linguistica, filosofia, logica e psicologia ed è perciò difficile avere una visione esaustiva sulle metodologie e sui risultati ottenuti in tutti questi campi ma con il termine Natural Language Processing o Elaborazione del linguaggio Naturale ci si limita a considerare solamente quell'in-

sieme di tecniche computazionali che processano sia il linguaggio umano parlato che quello scritto e che si distinguono da altri sistemi di elaborazione dati per la "conoscenza" del linguaggio. Le applicazioni che rispondono a questa definizione sono innumerevoli: dal semplice *word counter* passando per i *bot* in grado di rispondere automaticamente alle richieste dell'utente alla traduzione simultanea del parlato. Scopo dell'NLP è perciò quello di progettare gli algoritmi atti al trattamento automatico del linguaggio naturale. La conoscenza del linguaggio cosi come la generazione automatica di testo implica una conoscenza e un'elaborazione suddivisa in diversi livelli e che può essere vista come una pipeline:

1. *Lessico*: il dizionario delle parole utilizzate

2. *Morfologia*: lo studio di come le varie parti delle parole , i *morfemi* (radice e affissi), o unità minime di prima articolazione, sono combinate per formare una parola (*mangiare / mangia-re*). L'analisi morfologica mira a scomporre una parola nei suoi morfemi, un problema simile allo stemming che riduce una parola alla sua forma base o lemma e che abbiamo visto fondamentale nel mondo dell'Information Retrieval per la creazione di un indice efficiente.

3. *Sintassi*: lo studio sulle modalità di combinazione delle parole par la costruzione di una frase a livello di costituenti,relazioni grammaticali (legame fra soggetti e oggetti) e di dipendenza (*volere* regge un infinito o ha un complemento oggetto mentre *trovare* può avere solo un complemento oggetto)

4. *Semantica*: studia le tecniche per l'interpretazione di una frase che richiedono di conoscere il significato di ogni parola e come queste si combinano insieme per dare un significato complessivo alla frase

5. *Pragmatica*: studia il modo in cui il senso di una frase è usato per manifestare uno scopo o un'intenzione all'interno di un determinato contesto o particolare situazione. E' il contesto a permettere la correlazione tra il significato letterale della frase e l'intenzione o concetto che si vuole trasmettere e comprende aspetti quali conoscenze sociali o culturali condivise tra gli interlocutori nonché informazioni spazio-temporali sulla loro situazione o sul progresso all'interno del discorso.

Applicazioni riguardanti la lingua parlata generalmente implicano l'analisi di un ulteriore livello, il più basso di tutti che riguarda l'analisi Fonetica e Fonologica ossia dei suoni che costituiscono un linguaggio.

Il problema fondamentale che tutte le tecniche NLP si trovano ad affrontare e mirano ad eliminare è l'*ambiguità*, quel fenomeno intrinseco ad un elemento (linguistico) per cui una struttura è ambigua se possiede più interpretazioni linguistiche; anche l'ambiguità può essere presente a vari livelli. A livello morfologico accade che significati diversi coincidano nel medesimo [1] significante e che se condividono la grafia vengano definiti *omografi* altrimenti se caratterizzati dalla stessa realizzazione fonetica prendano il nome di *omofoni*; quando invece due significati diversi condividono entrambi vengono detti *omonimi*. A livello sintattico e strutturale si ha ambiguità se la sintassi di una frase può avere diverse interpretazioni con significati diversi. Un esempio tipico utilizzato in linguistica è quello a seguire dove non è ben chiaro quali siano il soggetto o il verbo:

«Una vecchia legge la regola.»

In una frase del genere sarebbe lecito chiedersi se la frase stia alludendo ad un'anziana signora intenta a leggere una regola oppure ad una vecchia legge che definisce come è regolata "una" situazione.

2.2.1 Analisi Morfologica e POS-tagging

Le parole o *token* in un corpus possono essere raggruppate in classi dette POS (Part of Speech o classi morfologiche) che forniscono informazioni fondamentali per la determinazione del ruolo della parola stessa nella frase e sebbene la grammatica tradizionale preveda poche tipologie di POS (sostantivi, verbi, aggettivi, perposizioni, avverbi . . .) esistono dei modelli più recenti quali il Penn Treebank tagset (si veda la tabella 2.1), il C5 e il C7 tagset che per l'inglese considerano un numero di classi maggiore e rispettivamente pari a 45, 61 e 146. In genere il *tag* è un'annotazione postfissa alla parola ed è introdotta dal carattere '/'. L'operazione che prevede di agganciare questi tag ad ogni parola di un testo viene chiamata *POS-tagging* ed è indispensabile in attività come l'estrazione di termini complessi, named entity recognition, speech recognition mentre è il punto di partenza

[1] In linguistica, nella definizione di segno formulata da F. de Saussure, l'immagine acustica o visiva, ossia l'elemento formale, la 'faccia esterna' del segno (quella interna è il significato) che consente, sul piano della langue, di identificare le sue diverse realizzazioni foniche concrete che si collocano sul piano della parole.[8]

per applicazioni linguistiche molto più complesse come parsing, information extraction,traduzione automatica o la costruzione di modelli predittivi per correttori automatici. Questa tecnica richiede una stringa di parole insieme al tagset scelto per la particolare applicazione/lingua e produce l'associazione di ogni parola con il miglior tag tra più tag compatibili; la presenza di diversi TAG per una stessa parola comporta la necessità anche per il POS tagger di risolvere ambiguità. Per collezioni di dimensione risibile è possibile la risoluzione manuale di queste ambiguità, soluzione invece impensabile per collezioni utilizzate per esempio da complessi sistemi di IR che richiedono un meccanismo di annotazione automatica generalmente realizzato mediante tecniche di apprendimento supervisionato.

Tag	**Description**	**Example**	**Tag**	**Description**	**E.**
CC	Conjunction	and, but	SYM	Symbol	+,€
CD	Cardinal number	one, two	TO	"to"	to
DT	Determiner	a, the	UH	Interjection	ah, uh
EX	Existential 'there"	there	VB	Verb, base form	eat
FW	Foreign word	culpa	VBD	Verb, past tense	ate
IN	Preposition	of, in, by	VBG	Verb, gerund	eating
JJ	Adjective	yellow	VBN	Verb, past particip.	eaten
JJR	Adj. comparative	bigger	VBP	Verb, non-3sg pres	eat
PDT	Predeterminer	all, both	"	Left quote	"
POS	Possessive ending	's	"	Right quote	"
PP	Personal pronoun	I, you	(	Left parenthesis	(<
PP$	Possessive pronoun	my, your	)	Right parenthesis	) >
RB	Adverb	often	,	Comma	,
RBR	Adv. comparative	faster	.	Sentence-final pun	. ! ?
RBS	Adv. superlative	fastest	:	Mid-sentence punt.	: ; -

Tabella 2.1: Un subset del Penn Treebank tagset che prevede più classi dalla grammatica tradizionale.

Rule Based Tagger I tagger basati su regole sono stati i primi tagger introdotti e prevedono la creazione di un ampio set di regole che specificano le modalità di risoluzione di ambiguità nel testo. La disambiguazione nel tagging di una parola è realizzata analizzando le caratteristiche linguistiche della parola stessa, la parola che la precede e quella a seguire; per esempio

se la parola precedente è un articolo, la parola in esame deve essere un nome. Queste regole hanno lo scopo di eliminare le ipotesi incompatibili con il contesto.

ADVERBIAL-THAT RULE

input: that
if
(+1 A/ADV/QUANT);
(+2 SENT-LIM);
(NOT -1 SVOC/A)
then eliminate non-ADV tags else eliminate ADV tag

Tra i tagger più utilizzati per l'inglese figurano TAGGIT e ENGTWOL che contengono rispettivamente 3300 e 1100 regole di disambiguazione a fronte di decine di TAG definiti. Nei tagger basati su regole difficilmente si supera un'accuratezza del 90% e per questo sono molto più utilizzati algoritmi di tipo probabilistico. L'accuratezza in un processo di classificazione quale il POS-tagging può essere calcolata mediante diverse metriche, una delle quali è la *Precision/Recall*. Nella Precision/Recall i termini *vero positivo*, *vero negativo*, *falso positivo* e *falso negativo* sono utilizzati per confrontare il TAG di un oggetto con quello desiderato (la classe a cui in realtà appartiene l'oggetto).*Precisione* e *recupero*(Recall) sono definite come:

$$Precisione = \frac{veropositivo}{veropositivo + falsopositivo} \tag{2.9}$$

$$Recupero = \frac{veropositivo}{veropositivo + falsonegativo} \tag{2.10}$$

Tagger probabilistici La maggior parte dei tagger attualmente presenti sullo scenario NLP si basano su un approccio di apprendimento supervisionato che è un approccio molto comune in machine learning e linguistica computazionale di cui il tagging è un esempio classico. Invece di formulare esplicitamente regole per tagging, un set di documenti viene suddiviso in un training set da annotare manualmente per allenare il modello e un test set da utilizzare per valutare il modello stesso. Il modello più semplice, intuitivo ma anche più diffuso, è il modello Hidden Markov; i POS-tagger basati su modelli HM scelgono la sequenza di TAG che massimizzano la probabilità p(T|W), ovvero la sequenza a massima verosimiglianza (Viterbi) data la sequenza di parole osservate W.

$$T = argmax_T p(T|W) = \prod_{i=1}^{n} P(t_i) \quad T = t_1 t_2 \dots t_n \quad W = w_1 w_2 \dots w_n \tag{2.11}$$

dove, con le dovute approssimazioni, la probabilità dei TAG è data a sua volta da

$$P(t_n) = P(w_n|t_n)P(t_n|t_{n-2}, t_{n-1}) \tag{2.12}$$

con

$$P(w_n|t_n) \approx \frac{\#(w_i t_i)}{\#t_i} \quad e \quad p(t_i|t_{i-2}ti-1) =\approx \frac{\#(t_{i-2}t_{i-1}t_i)}{\#(t_{i-2}t_{i-1})} \tag{2.13}$$

Ad Esempio per la frase:

Il**/ART** cane**/N** abbaia

la probabilità che la parola *abbaia* sia taggata con V è data da

$$P(t_3 = V) = P(abbaia|V)P(V|ART, N) \tag{2.14}$$

dove le due probabilità P(abbaia|V) e P(V|ART,N) vengono calcolate contando le due frequenze (2.13) da un training corpus già annotato. Con gli algoritmi di tagging probabilistico si ottiene un'accuratezza anche superiore al 96%.

Stemmer Spesso il POS tagging viene considerato l'anello di unione tra il livello morfologico e quello sintattico che servendosi dell'output del livello sottostante ottiene informazioni sulla frase di cui fa parte una parola e sul suo contesto. I tag stessi tra i quali disambiguare, infatti, vengono generati a partire dall'analisi morfologica che associa ad ogni parola in ingresso il suo stem(radice) e delle *feature* come ad esempio numero/persona/modo/-tempo. Un esempio di output di analisi morfologica può essere il seguente:

student	student+ N + SG
students	student + N + SG
cities	city + N + PL
looks	look + V + 3SG look + N + PL

Un'analisi morfologica per parole come *looks* nell'esempio precedente provocano la generazione di ambiguità (più TAG per una stessa parola).

2.2.2 Analisi Sintattica

L'analisi sintattica ha il compito di assegnare una caratterizzazione sintattica alla frase identificando i suoi*costituenti*[3] (o *chunck*), le sue relazioni grammaticali e relazioni di dipendenza. Per l'identificazione dei costituenti i motori di elaborazione del linguaggio utilizzano i **parser** che presi in input una frase ed una grammatica hanno il compito di riconoscere se la prima può essere generata dalla grammatica specificata e, in caso affermativo, assegnare alla frase la giusta rappresentazione sintattica che prende il nome di *albero di parsing*; il nodo radice è detto *Sentence* o *S*, i nodi terminali contengono le parole della frase mentre i nodi intermedi rappresentano la struttura sintattica assegnata alla frase. E' possibile che esista più di un albero per la stessa frase (sinonimo di ambiguità strutturale) mentre si parla di ambiguità di collegamento quando una parte può essere inserita nell'albero di analisi in più punti.

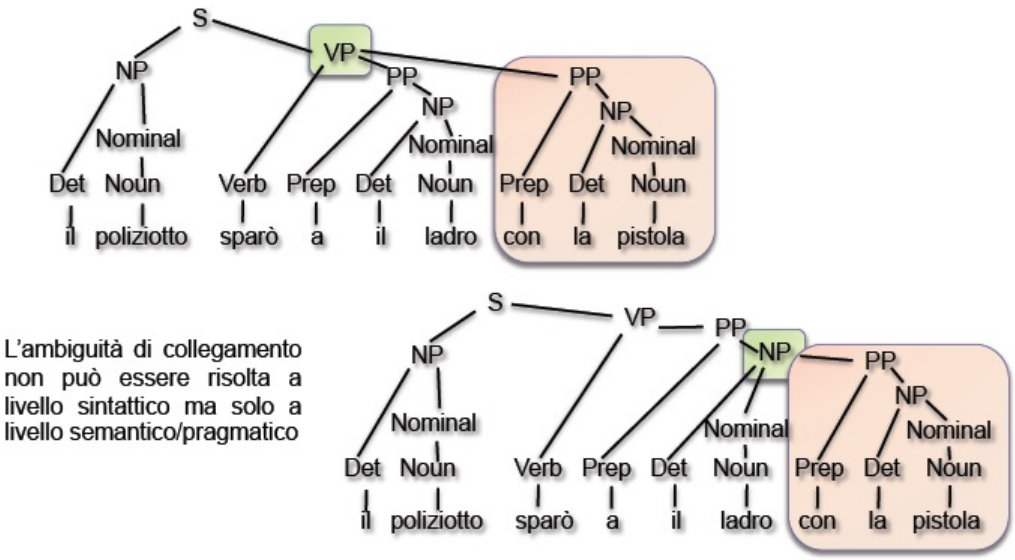


Figura 2.6: Esempio di albero sintattico a seguito dell'operazione di parsing su una frase [9]

Il parsing può essere visto come il processo di ricerca dell'albero di parsing più adatto alla rappresentazione della frase tra tutti quelli generabili da una grammatica ed è un processo che richiede spesso conoscenza statistica/semantica. A tale scopo sono stati realizzati nel tempo diversi algoritmi;

[3]I costituenti sono gruppi di parole organizzate in un unico blocco che hanno una specifica funzione grammaticale. Esempi di Chunk sono NP, VP e PP rispettivamente Noun Phrase, Verbal Phrase e Preposition Phrase.

l'algoritmo di Earley è basato sulla programmazione dinamica e allo scopo di incrementare l'efficienza del task memorizza tutti i sotto-alberi visitati (o *stati*), relativi ai componenti della frase, come entry in una struttura dati chiamata *chart*. Il chart memorizza gli stati visitati durante l'analisi in modo efficiente e per ogni parola di una frase il chart contiene la lista di tutti gli alberi di analisi parziali che sono stati generati fino a quel punto. Nella posizione corrispondente alla fine della frase il chart contiene una codifica compatta di tutti gli alberi di analisi possibili. Ogni stato del chart contiene tre elementi:

1. Un sottoalbero corrispondente ad una regola grammaticale
2. Il grado di completamento del sottoalbero (si usa una regola puntata di cui un esempio è visibile in fig. 2.7)
3. La posizione del sotto albero rispetto all'ingresso codificata con una coppia di indici corrispondenti all'inizio del sottoalbero e alla posizione del punto

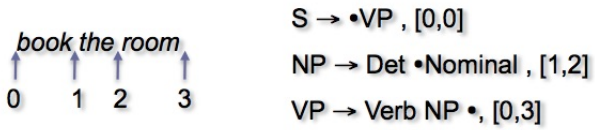


Figura 2.7: esempio di stati di un chart [9]

L'algoritmo viene eseguito una sola volta scandendo da sinistra a destra tutta la frase e ad ogni passo si prende in considerazione uno stato e si applica una fra le tre operazioni possibili, generando un nuovo stato nella posizione corrente del chart o in quella successiva; l'algoritmo procede sempre in avanti non rimuovendo stati già generati ma aggiungendone sempre di nuovi e solo quando è presente nel chart lo stato S $\rightarrow \alpha\cdot$,[0,N] nell'ultima posizione del chart un'analisi viene considerata terminata con successo. Le tre operazioni possibili sono:

1. **Previsione**: crea nuovi stati nell'entrata corrente del chart, rappresentando le aspettative top-down della grammatica; Si genera uno stato per ogni possibile espansione del non-terminale e lo si inserisce nello stesso insieme del chart; Il punto di inizio e fine degli stati inseriti sono gli stessi dello stato che li ha generati.

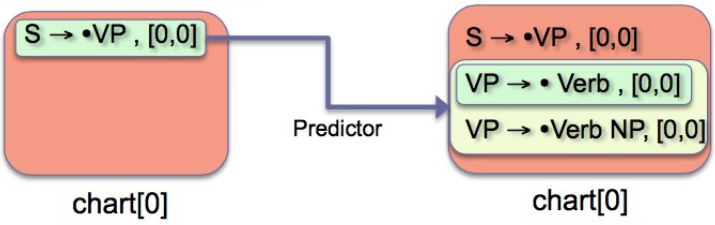


Figura 2.8: Predittore per uno stato del chart riferito all'esempio di fig. 2.7 [9]

2. **Scansione**: Quando uno stato ha un elemento terminale (POS) alla destra del punto lo scanner esamina l'input. si verifica se nell'input esiste, una parola la cui categoria combacia con quella prevista dallo stato a cui la regola si trova. Se il confronto è positivo, la scansione produce un nuovo stato in cui l'indice di posizione viene spostato dopo la parola riconosciuta e la posizione del punto è incrementata.

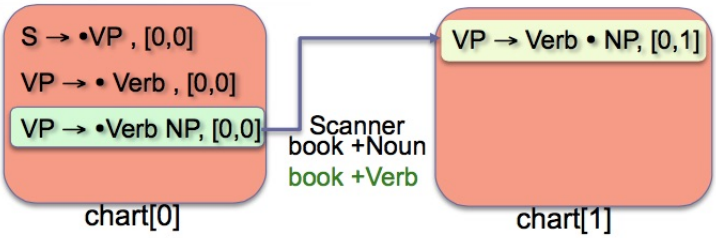


Figura 2.9: Scanner per uno stato del chart riferito all'esempio di fig. 2.7 [9]

3. **Completamento**: Si applica a uno stato quando il punto ha raggiunto l'estremità destra della produzione (determina che un sintagma significativo è stato riconosciuto) e verifica se l' avvenuto riconoscimento è utile per completare qualche altra regola rimasta in attesa di quella categoria.

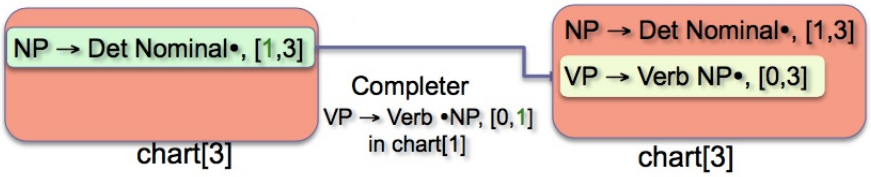


Figura 2.10: Completer per uno stato del chart riferito all'esempio di fig. 2.7 [9]

Ottenuto il chart è possibile ricostruire un albero di analisi estraendo l'arco o gli archi che uniscono i nodi dal primo all'ultimo. Gli alberi validi corrispondono agli stati S $\alpha\cdot$, [0,N] nell'ultima posizione del chart.Per la ricostruzione occorre mantenere traccia di quali completamenti sono stati fatti dal Completer e a partire dalla regola completa nella posizione N del chart ripercorrere all'indietro, ricorsivamente, le sostituzioni che sono state memorizzate.

2.2.3 Apache OpenNLP

Attualmente sono presenti tantissimi tool per l'elaborazione del linguaggio naturale e il mondo open source offre librerie di elevatissima qualità per la risoluzione dei problemi più comuni nel processamento dei testi quali sentiment analysis, topic identification, autolabeling di contenuti. Molte di queste possono essere considerate come librerie che forniscono i blocchi base per soluzioni molto più complesse e innovative e che dispensano il programmatore dal concentrarsi su come reinventare tutta la ruota per farlo focalizzare sulla logica applicativa. Tra i progetti attualmente degni di nota figurano tool quali la Stanford's Core NLP Suite, il Natural Language Toolkit se si sceglie di implementare soluzioni in Python, Apache Lucene e Solr(tecnicamente nate per tutt'altro scopo offrono un numero considerevole di tool per l'elaborazione del testo) e infine openNLP.

Apache OpenNLP è una libreria rilasciata sotto licenza Apache e supporta i task più comuni dell'NLP quali tokenizzazione, segmentazione delle frasi, POS-tagging e parsing; permette quindi l'implementazione di una completa pipeline di annotazione. Essa offre un approccio diametralmente opposto alla Standford Suite in quanto fondata sul machine learning e sull'addestramento di modelli per il processamento del linguaggio naturale; fornisce infatti una vasta gamma di modelli pre-built per diversi linguaggi oltre che le risorse da cui questi modelli sono stati generati (testi annotati). Sfortunatamente la maggior parte dei modelli forniti non è stata realizzata per l'italiano ma il Language Detector di openNLP, un componente il cui modello è disponibile al download pubblico, è in grado di riconoscere fino a 103 lingue tra le quali figura la nostra, esistono poi diversi modelli third-part realizzati esclusivamente per l'italiano. Il modello del Language Detector riconosce linguaggi nello standard ISO 639-3 ed è stato allenato mediante corpus Leipzig permettendo un'accuratezza del 98,81 % e precisione pari a 1 nel riconoscimento di frasi italiane [10].

2.3 Elastic Search

Elastic search è un server di ricerca open-source con funzionalità Full Text costruito su Apache Lucene, un framework Java nato per i sistemi di Information Retrieval, con l'obiettivo di poter lavorare in ambito Big-data per cui inseguendo un'elevata scalabilità, affidabilità e come dice il nome stesso, elasticità; esso fa parte di Elastic Stack, una suite base open source di Elastic che combina la potenzialità dei suoi moduli Elasticsearch, Logstash, Kibana e Beats per l'indicizzazione, la ricerca e la visualizzazione facilitata, in tempo reale, di log e di dati strutturati e non strutturati.In ElasticSearch un *cluster* è un insieme di *nodi*, gli elementi minimi che garantiscono il funzionamento del sistema, che condividono lo stesso cluster name[11]. All'interno del cluster un solo nodo viene eletto *Master* e si occupa di gestire operazioni come la creazione,eliminazione di indici e il bilanciamento degli indici e degli shard quando vengono introdotti o rimossi dei nodi dal cluster. Come risultato gli sviluppatori posso aggiungere o rimuovere nodi con lo scopo di incrementare o ridurre le risorse allocate per le proprie soluzioni su richiesta. Cosi come Solr, un altro search engine molto popolare, ElasticSearch utilizza Lucene come libreria core per l'indicizzazione e lo scoring dei documenti per cui vi è una fiducia implicita verso le capacità di base nella ricerca offerte da questo prodotto. Lucene offre diversi modelli IR in particolare offre un modello ottimizzato di tipo vettoriale che fa uso di pesi TF-IDF come discusso nel paragrafo 2.1.2. Tutte le funzionalità del motore sono esposte mediante API che accettano e rappresentano informazioni in formato JSON. E' possibile, ad esempio, specificare l'analyzer da utilizzare per la collezione di documenti oppure specificare un *mapping* per definire la struttura dell'indice; con il mapping si specifica con quale tipo di dato si vuole rappresentare un particolare campo del documento e se debba essere o meno analizzato dall'analyzer scelto per l'indice. I campi supportati da ES sono boolean, long, double, date e string e quando il motore indicizza un documento che contiene un campo sconosciuto fino ad allora, ES utilizza un mapping dinamico che stima il tipo di dato a partire da quelli JSON. Il mapping abilita le cosiddette ricerche Exact Value che si differenziano dalle Full Text search poichè più semplici da realizzare e per cui la decisione è binaria: il valore soddisfa la query oppure no.

2.3.1 Distributed Search

L'aspetto che ha permesso ad ES di prendere il sopravvento è la ricerca distribuita. Al contrario di Solr, ES è stato creato con lo scopo di lavorare su sistemi distribuiti in particolare con piattaforme distribuite come EC2 di Amazon e di supportare diverse piattaforme cloud come GigaSpaces, Coherence and Terracotta .Ciò si traduce nel fatto che gli indici di elastic search risiedono efficientemente su diversi server in modalità fail-safe. Com è noto la programmazione distribuita è in generale maggiormente complicata rispetto ai sistemi stand-alone ma se fatta correttamente offre resilienza nei confronti di attacchi malevoli, degradazione delle perfomance e maggiore sicurezza in generale. ElasticSearch permette la suddivisione degli indici in *shard* con una o più repliche nei nodi del cluster mentre il bilanciamento e il routing delle richieste allo shard corretto è delegato al cluster stesso; si pensi ad esempio ad uno scenario in cui un cluster è composto da 3 nodi e gestisce un indice suddiviso in due shard ognuno dei quali possiede due repliche non residenti nello stesso nodo; qualsiasi richiesta può essere inviata a qualsiasi nodo del cluster in quanto ognuno in questa architettura è in grado di gestire qualsiasi tipo di richieste.Ogni nodo conosce la posizione di ogni documento nel cluster e può inoltrare le richieste al nodo corretto questo assicura che anche in caso di guasti o di problemi software calino drasticamente le probabilità che il sistema vada completamente offline e diventi inutilizzabile. Sebbene il sistema supporti lo sharding degli indici il numero degli shard di un indice deve essere settato prima che l'indice venga creato e non può essere incrementato o decrementato su richiesta [11]; in compenso ElasticSearch ha introdotto un meccanismo che accetta la creazione di indici multipli per una stessa collezione per cui al posto di modificare il numero di shard è possibile costruire un nuovo indice dello stesso tipo quando il numero di documenti diventa troppo alto da mantenere per uno solo di essi (attualmente il numero massimo consentito è di 2 miliardi per shard, dovuto ad un limite restringente di lucene).

2.3.2 Query Routing

Nella fase iniziale di risoluzione della query questa è portata in broadcast dal nodo ricevitore, che assume il compito di *coordinatore*, ad ogni shard (sia primaria che replicata) dell'indice; ogni shard esegue la ricerca e crea una lista di risultati locale; tutti i risultati locali vengono consegnati al coordinatore che ne fa il merge e realizza una lista di risultati generale e la

torna all'utente in formato JSON.E' possibile specificare se limitare l'inoltro della query a un sottoinsieme degli indici. Per aumentare il livello di scalabilità ElasticSearch implementa un meccanismo sofisticato per il routing delle query ai nodi del cluster mediante il quale è possibile istruire il sistema affinchè solo i nodi che contengano i documenti interessati dalla ricerca processino la query. Gli indici supportano differenti *tipi* e ad ogni tipo può essere associato il relativo routing-value; la politica di default dell'indexer è quella di mantenere insieme tutti i documenti con lo stesso tipo nello stesso shard e se nella query di ricerca viene specificato il tipo, la query verrà inoltrata ai soli nodi che contengono quel tipo di documento.

2.3.3 Real-time

In tema real-time il sistema gode del meccanismo di *per-segment search* ereditato da Lucene e che permette l'allineamento dell'indice nell'ordine di alcuni millisecondi dopo che un aggiornamento è stato completato e che in Elasticsearch è chiamato *refresh*. Questo meccanismo si differenzia dai precedenti aggirando il collo di bottiglia introdotto dalla scrittura diretta dei dati sul disco(commit).viene sfruttata infatti la cache del filesystem;una volta in cache i dati possono essere letti e aperti come qualsiasi altro file senza aspettare che ne venga fatto il commit sul disco.lucine sfrutta proprio questo meccanismo per rendere i nuovi dati visibili e ricercabili. In elastic search gli shard vengono rinfrescati ogni secondo di default mediante questo meccanismo;per cui i nuovi dati non sono visibili immediatamente ma dopo un secondo circa Mediante refresh API pè possibile rinfrescare manualmente tutti o specifici indici. Questo rende elasticsearch l'architettura adatta per la ricerca di tipo real-time.

2.3.4 Analizzatori

Gli *analyzer* di ElasticSearch sono utilizzati nelle ricerche Full Text e l'analisi prevede 3 step sequenziali:

1. Il *character Filter* è responsabile di ripulire/riordinare le stringhe prima della fase di tokenizzazione eliminando tag HTML o convertendo caratteri come '&' in *and*

2. Il contenuto passa al *Tokenizer* che ha il compito di suddividerlo in termini semplici (token)

3. Infine, vengono applicati i *Token filter* per operazioni quali la rimozione di stop word, conversione in lowercase e l'aggiunta di sinonimi.

2.3.5 Valutazione

Per via della giovinezza di Elastic search altri progetti come SoIR, che è un altro server di ricerca open source basato sempre su Apache Lucene, possiedono delle community molto più mature e una documentazione più ricca e intuitiva, entrambi aspetti determinanti e influenti nella scelta di una soluzione rispetto ad un'altra. In ogni caso, Elastic Search si è imposta come unica e vera soluzione nativa per la creazione di motori di ricerca sicuri in ambienti distribuiti come quello del Cloud: è nato per essere altamente scalabile, leggero e facilmente configurabile, integrabile e offre delle API molto più intuitive e accessibili della concorrenza.

Capitolo 3

Descrizione del sistema

In questo capitolo vengono illustrate le scelte progettuali, l'architettura e le tecnologie utilizzate per la realizzazione del motore di ricerca. Il corpus scelto per il motore è quello delle pagine web e dei documenti informativi presenti sul portale dell'agenzia assicurativa; tutti i dati raccolti sono stati forniti dalla compagnia in questione e rispecchiano lo stato del loro portale web al momento in cui è iniziato il lavoro di tesi. Per la raccolta dei metadati relativi ai fascicoli informativi presenti sul portale è esposta un' API anch'essa fornita preventivamente. Il sistema è stato realizzato prevedendo una parte di Infomation Retrieval per la raccolta/mantenimento dei dati, l'utilizzo di tecniche di NLP per l'elaborazione delle richieste in linguaggio naturale e per l'acquisizione della conoscenza del dominio e un servizio di integrazione che elabora le query utente utilizzando questa conoscenza per esprimere richieste puntuali al motore. La scelta tecnologica è ricaduta sulla suite Watson di IBM e in particolare sui software Content Analytics (WCA di seguito), un insieme di prodotti IBM che integrano in loro tutte queste funzionalità oltre a strumenti di analisi testuale e di personalizzazione dei crawler. Per offrire un'esperienza utente migliore si è scelto di realizzare anche un agente atto a simulare una conversazione umana con gli utenti del servizio in alternativa al tradizionale front-end; questa scelta ha implicato l'utilizzo di ulteriori tecnologie quali Conversation, altro prodotto IBM, e Node-Red per l'integrazione dei servizi necessari per il funzionamento del bot. Scopo ultimo di questo lavoro è quello di dimostrare l'ottimizzazione nel ranking dei risultati del motore di ricerca data dalla conoscenza del dominio e per testarlo sono state raccolte alcune frasi d'esempio. Le frasi raccolte sono all'incirca 750 e sono state scelte per stimare la copertura del sistema e per comprendere la natura delle frasi che un utente potrebbe proporre al motore.

3.1 Architettura del sistema

L'architettura realizzata per l'implementazione del servizio è rappresentata in Figura 3.1.

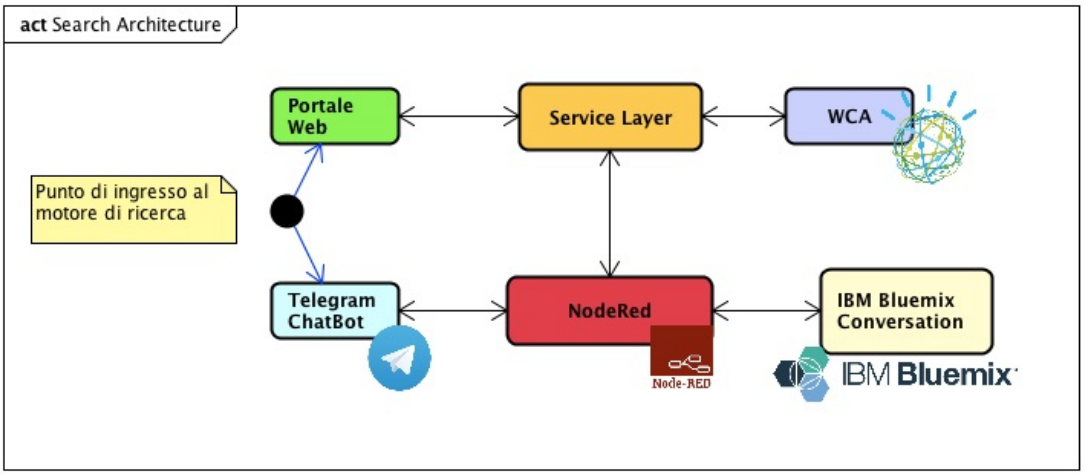


Figura 3.1: Architettura del sistema realizzato

Sono previsti due punti di ingresso al sistema per l'utente finale che ha la possibilità di accedere alle funzionalità del motore di ricerca attraverso la pagina Web del portale oppure mediante l'interazione con un agente conversazionale in veste di assistente virtuale; entrambe le soluzioni sono in grado di elaborare richieste in linguaggio naturale. Il bot sfrutta le potenzialità di Node-Red, un tool di programmazione a flussi, per interrogare i servizi offerti dal Conversation e dal Service Layer dell'architettura. IBM Bluemix Conversation è stato scelto per riconoscere le intenzioni dell'utente e distinguere tra la volontà di effettuare una ricerca sul portale e le richieste di altra natura permettendo la costruzione di un vero e proprio dialogo. Il Service Layer si occupa di elaborare le query utente arricchendole con informazioni derivanti dalla conoscenza del dominio e di interrogare i servizi di Watson Content Analytics mediante le API da esso esposte; quest ultimo, da ora in poi chiamato WCA, è il cuore pulsante del motore di ricerca.

3.2 Tecnologie utilizzate

Il Service Layer è basato sul framework SpringBoot e scritto interamente in **Java** mentre la gestione delle dipendenze è demandata ad **Apache Maven**. I moduli dei flussi Node-Red che richiedono una logica applicativa accet-

tano codice scritto in **Javascript** mentre tutte le componenti coinvolte dal progetto che espongono servizi si basano su un architettura di tipo **REST**.

La rappresentazione degli elementi utilizzati per la comunicazione tra i vari moduli del sistema è basata su oggetti **JSON** che permettono un'interpretazione flessibile del contenuto informazionale e possono essere trattati come semplici stringhe o come oggetti complessi.

3.3 Moduli

I moduli che compongono l'architettura rappresentata in figura 3.1 sono descritti nei paragrafi a seguire.

3.3.1 Node-Red

Node-Red è un tool di sviluppo visuale di supporto alla programmazione basata sul paradigma a flussi [12]. Questo paradigma prevede che un'applicazione venga costruita mediante l'unione di una serie di *black box* dei quali bisogna conoscere la funzionalità ma non necessariamente l'implementazione interna. Questi blocchi possiedono una o più porte sia in ingresso che in uscita utilizzate per connettersi in una architettura all'interno della quale comunicano scambiandosi dati sotto forma di messaggi (noti come *information packets*). Gli *ip* possono trasportare del contenuto informativo oppure possono fungere da segnali per poter innescare l'avvio di un processo relativo al blocco cui essi sono destinati. Con il paradigma a flussi, quindi, il programma non è più una sequenza di istruzioni ma un insieme di flussi di dati e segnali scambiati tra i blocchi di un'architettura in maniera asincrona. I blocchi dei flussi sono comunque caratterizzati da codice da eseguire, per cui lo sviluppo di un'applicazione prevede due livelli : quello visuale e quello implementativo, realizzato esclusivamente in Javascript, per ciascun blocco. Il flusso realizzato per questo lavoro è raffigurato in Fig.3.2 e Fig.3.3.

Il flusso inizia con l'arrivo di un nuovo messaggio da parte dell'utente e il nodo che gestisce tale evento è il *Telegram Receiver*. Telegram mette a disposizione *BotFather* definito dalla stessa come "un bot per comandarli tutti" [13] che è un'app pubblica disponibile sotto forma di chatbot che guida l'utente alla creazione e configurazione di un bot mediante una serie di comandi. Scelto un nome e uno username per il bot che si vuole creare, in questo caso rispettivamente **WatsonContentAnalytics** e **wca_bot**,

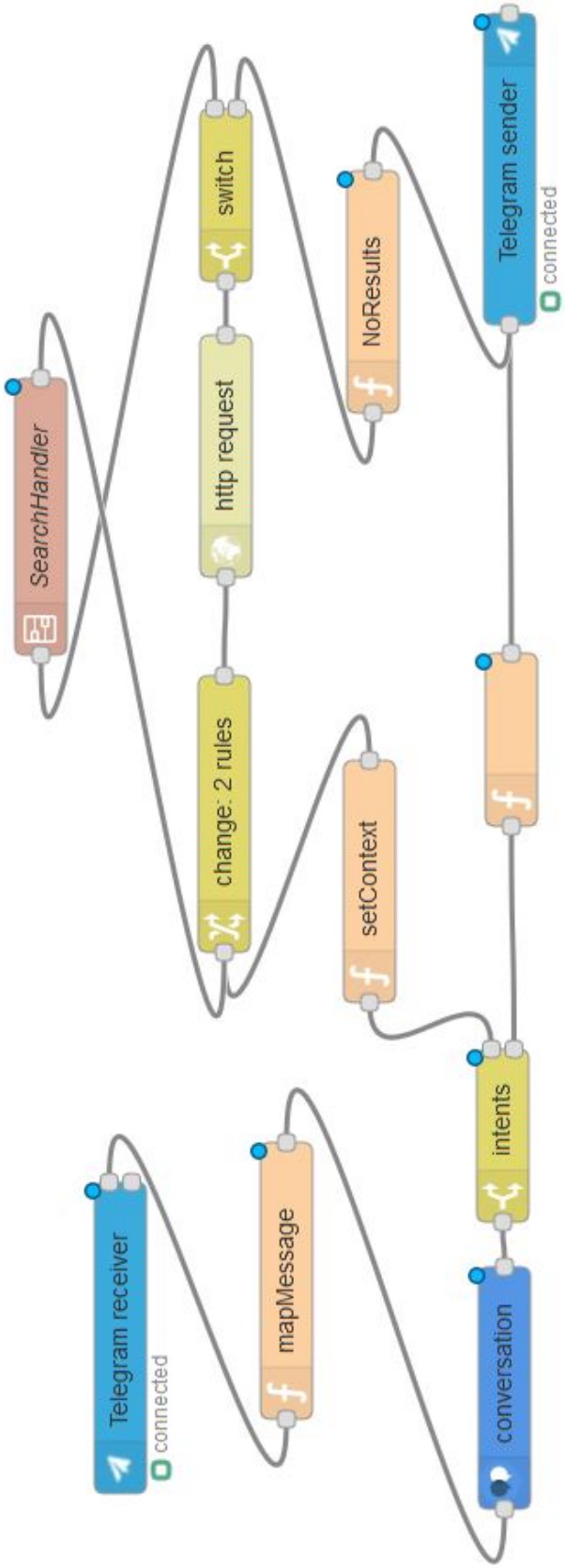


Figura 3.2: Flusso Node-Red principale per la realizzazione dell'agente conversazionale previsto dall'architettura del progetto.

BotFather restituisce il *tokenHTTP*, un identificativo univoco che permette di controllare il bot mediante le API HTTP che Telegram espone e che Node-Red utilizza nel suo flusso; per wca_bot, il Bot Father ha restituito il tokenHTTP **499115660:AAGWD6kPgZTfeISNp9Mcccp6Jazqqk42w4Q**. Il package **node-red-contrib-telegrambot**, attualmente disponibile nella versione 2.7.3, è un wrapper del **node-telegram-bot-api** e permette di manovrare il flusso applicativo del bot una volta configurato con il tokenHTTP rilasciato dal "padrino".

Tornando al Telegram Receiver esso riceve messaggi dal bot e invia al nodo a seguire un Information Packet con il seguente formato:

- **msg.payload** contiene tutti i dettagli del messaggio ricevuto e da inoltrare al nodo successivo
  - **`chatId`** : l'identificativo della chat che deve essere passato al nodo successivo quando si vuole rispondere all'interno della stessa chat.
  - **`type`** : il tipo del messaggio ricevuto (testo, immagine, audio, posizione, video, voce, contatto)
  - **`content`** : il contenuto del messaggio (`string`, `file_id`, o un oggetto contenente il binario dei dati)
- **msg.originalMessage** contiene l'oggetto message originale proveniente dal nodo sottostante della libreria node-telegram-bot-api.

Il messaggio ricevuto dalla chat Telegram viene quindi inoltrato, dopo averne estratto il contenuto informativo, al nodo incaricato di interrogare il servizio Conversation (vedi par. 3.3.2). Quest'ultimo fa parte della Palette **node-red-contrib-watson-ucg-custom** realizzata da IBM per il supporto a tutti i servizi Cloud di Watson. Il nodo Conversation si occupa di inoltrare il contenuto testuale del messaggio al servizio, per riconoscere l'intento dell'utente, e di restituirne la risposta. A questo punto, il flusso controlla che effettivamente sia stato riconosciuto almeno un intento tra quelli per cui l'agente conversazionale è addestrato (#greetings, #personalQuestions, #search, #faqId). In particolare lo *switch-node* di Node-Red, chiamato *intents* in figura 3.2, valuta la condizione:

```
msg.payload.intents[0] is not null
```

e solo se:

```
msg.payload.intents[0].intent == search
```

il flusso prosegue per quella parte dell'architettura predisposta alla gestione dell'intento di ricerca e che andrà ad interpellare WCA; per ogni altro caso la risposta prevista dal conversation è inoltrata direttamente all'utente mediante il *Telegram Sender* e senza alcuna elaborazione ulteriore. Una volta intrapreso il ramo dedicato alla gestione della ricerca viene eseguito il nodo funzione *setContext*. Esso si occupa di settare alcune variabili di contesto affinchè informazioni indispensabili, come la richiesta dell'utente, non vengano perse mano a mano che il flusso prosegue per i nodi lungo il ramo. Il codice Javascript che esegue *setContext* è il seguente:

```
var flowContext = this.context.flow;
flowContext.set('query', msg.payload.input.text);
flowContext.set('chatId', msg.chatId);
flowContext.set('count',1);}}
```

dove **query** è il testo in input al *Conversation* e che coincide con la query che andrà inoltrata al motore di ricerca, **chatId** è l'id univoco che si aspetta Telegram per legare i messaggi a una stessa chat, **count** è invece utilizzata per scorrere i risultati del motore di ricerca, viene così generata una sorta di "memoria" per l'agente che ricorda quali risultati ha già mostrato all'utente.

Il nodo *change: 2 rules* estrae in modo appropriato, dal corpo dell'information packet che lo ha raggiunto, i parametri da inviare in POST all'endpoint di search del *service layer* e li passa al nodo *http request* che si occupa di eseguire a tutti gli effetti la Request indirizzata all'endpoint **http://<host-service>:<port>/searchAPI?input=**{**post**}**&page=**{**page**}. {*post*} e {*page*} indicano il mapping con i relativi campi creati e inizializzati al nodo *change: 2 rules*. Il nodo Switch a seguire verifica il contenuto della risposta del service layer. Se:

```
msg.payload.es_apiResponse.es_totalResults > 0
```

il flusso prosegue nel subFlow *searchHandler* (si veda la Fig. 3.3) altrimenti si notifica all'utente che non è stato trovato alcun risultato costruendo all'interno del nodo *NoResults* il seguente messaggio:

```
msg.payload =
  chatId : msg.chatId,
  type : "message",
  content : "Non ho trovato nulla! Sicuro di avermi chiesto
     qualcosa relativo a un'assicurazione?"
```

e recapitandolo sempre mediante il *Telegram Sender* che si occupa di farlo visualizzare sulla chat dell'utente. Se si è intrapreso il subFlow *search-*

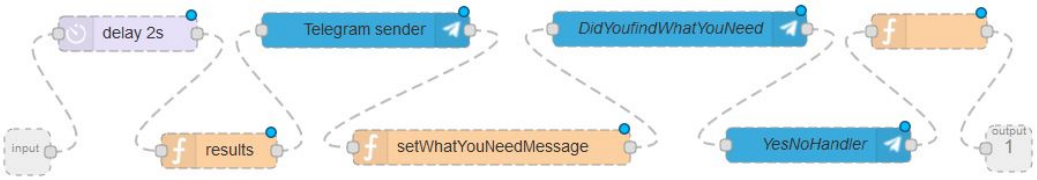


Figura 3.3: Composizione del flusso Node-Red secondario utilizzato dal flow in figura 3.2 per la restituzione dei risultati all'utente e per la gestione di una sua eventuale risposta

Handler si è in grado di estrarre i risultati restituiti dal motore di ricerca mediante il service layer intermiediario; il nodo *results* è il nodo responsabile della presentazione dei risultati all'utente. In particolare per ogni elemento della pagina dei risultati ne estrae *title* e *href* per restituire nei suggerimenti all'utente i link alle pagine del portale.

```
var href =
   msg.payload.es_apiResponse.es_result[0].es_link[0].href;

var title =
   msg.payload.es_apiResponse.es_result[0].es_title);
```

Una volta consegnati i suggerimenti, il flusso costruisce un ulteriore messaggio per l'utente preoccupandosi di chiedergli se la sua richiesta fosse stata soddisfatta. *YesNoHandles* è un callback query node di Telegram che rimane in ascolto sulla risposta inline ad un preciso messaggio inviato all'utente:

```
msg.payload = {

  chatId : msg.payload.chatId,
  type : "message",
  content:'Hai trovato quello che ti serviva?',

  };
```

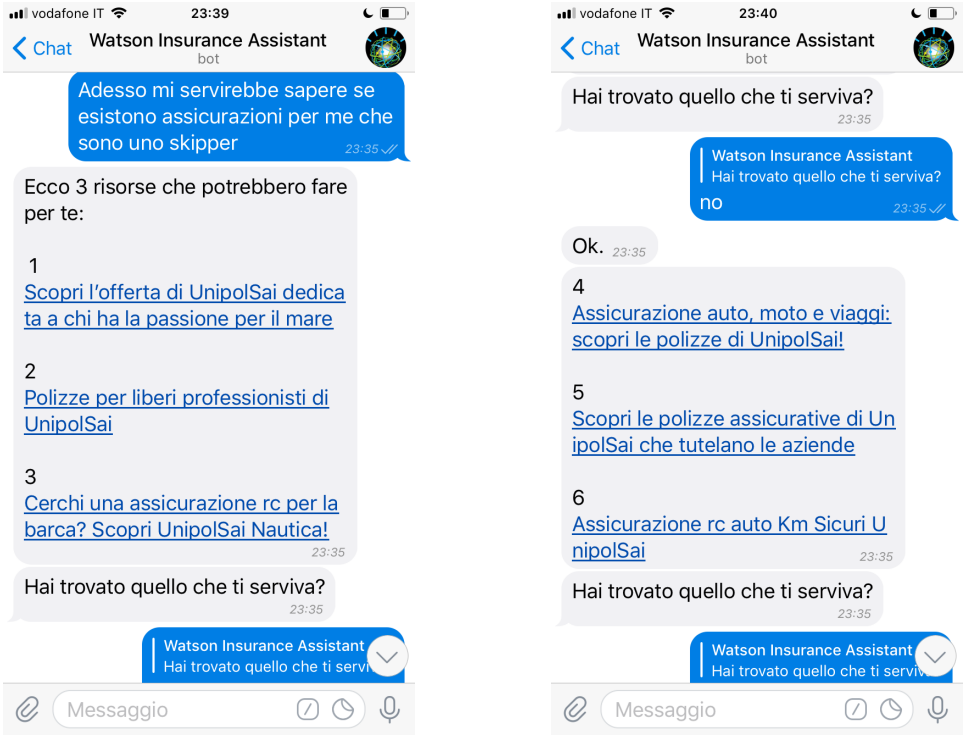
Ci si aspetta una risposta negativa per scorrere le result page di WCA di volta in volta di una quantità pari a 1:

```
var page = flowContext.get('page')+1;
flowContext.set('count',page);

if(msg.payload.content === 'no') {

    msg.payload={

        chatId : chatId,
        type : 'message',
        content : flowContext.get('query'),
        page : page

    };
 return msg;

}

if(msg.payload.content === 'si') {

  flowContext.set('count',1);
  return null;

}
```

L'esperienza utente risulta essere quella rappresentata in figura 3.4, dove è possibile osservare il comportamento del sistema al riconoscimento dell'intento di ricerca (Fig.3.4a) e la gestione della risposta inline che permette di scorrere i risultati di ricerca (Fig.3.4b).

3.3.2 IBM Conversation

Conversation è il modulo dell'architettura preposto a distinguere e gestire le differenti intenzioni dell'utente ed é un'applicazione presente nel catalogo di IBM Bluemix, servizio Cloud della stessa azienda offerta come Platform as a Service (PaaS); grazie a esso è possibile costruire agenti conversazionali, soluzioni in grado di comprendere input in linguaggio naturale, e usa tecniche di Machine Learning per rispondere agli utenti simulando una conversazione tra umani [14]. Gli utenti, generalmente, vi interagiscono



(a) Gestione intento di ricerca (b) Scorrimento risultati di ricerca

Figura 3.4: Esperienza utente offerta dal Bot su architettura Node-Red

mediante delle interfacce che richiamano le API esposte e a cui inoltrare l'input dell'utente; come descritto in precedenza, per questo progetto si è scelto di utilizzare l'app Telegram ma nulla vieta di porci un robot parlante. L'applicazione sviluppata si connette ad un **workspace**, un contenitore per il proprio flusso di dialogo e i dati di training; il servizio interpreta l'input dell'utente e dirige il flusso della conversazione grazie a un modello realizzato a partire dai dati di training. I **training data** consistono nelle seguenti astrazioni:

- **Intenti**: Un intento è l'intenzione che l'utente si prevede possa avere nell'interagire con il servizio. Per ogni intento è necessario aggiungere qualche esempio che riflette il tipo di richieste che l'utente può porre al bot.

- **Entità**: Le entità rappresentano termini o oggetti che forniscono un contesto per un intento. Ad esempio un'entità può essere il nome di una città che aiuta a distinguere per quale località l'utente sta richiedendo un'informazione nel dialogo.

Affinché si riconosca l'intenzione dell'utente ad effettuare una ricerca sul motore, è stato configurato l'intento **#search**; gli intenti nella forma

#faqid servono invece ad assistere l'utente rispondendo direttamente a domande frequenti (FAQ) senza dover interpellare il motore di ricerca. Sono stati definiti anche altri tipi di intenti per gestire poi la conversazione in modo più naturale (si veda fig. 3.5). Alcune frasi d'esempio utilizzate

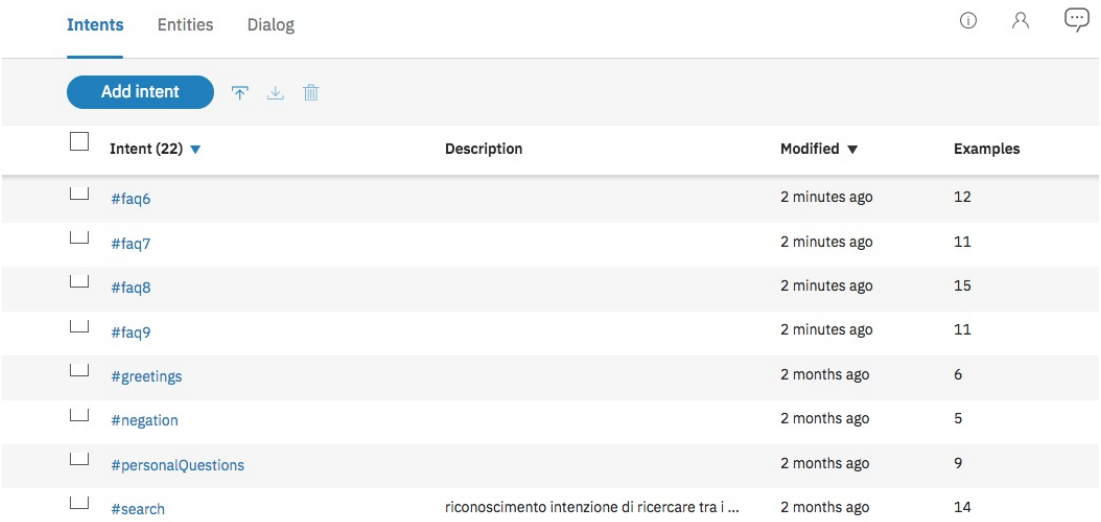


Figura 3.5: Schermata di gestione degli intenti in IBM Bluemix Conversation

nell'addestramento del classificatore per l'intento **#search**, sono:

- "*vorrei cercare*"
- "*troveresti per me*"
- "*mi servirebbe sapere*"
- "*vorrei sapere se esistono*"
- "*vorrei avere informazioni su*"

Oltre a queste, anche keyword precise come "*trova*" o "*cerca*"e maggiore è il numero di esempi e maggiore sarà la confidenza nel riconoscere gli intenti. Non è stato, però, necessario configurare alcuna entità in quanto il contesto è mantenuto dall'architettura costruita su Node-Red (node *setContext*). Una volta aggiunti i dati di training, il workspace utilizza e addestra un Natural Language Classifier, altro servizio della piattaforma Bluemix costruito su un Algoritmo di Deep Learning proprietario, a riconoscere gli intenti dell'utente e per i quali il servizio deve rispondere; per decidere, Conversation, costruisce un albero le cui foglie prendono il nome di *dialog node*. Ogni nodo contiene ed è costituito da almeno una condizione (su un

intento o un'entità) e una risposta e si può pensare ad essi come ad un costrutto if/then: se la condizione è verificata viene restituita la risposta. Il servizio visita l'albero dall'alto verso il basso e se verifica la condizione di un particolare nodo ne innesca l'esecuzione che produrrà la risposta associata. Le condizioni sono del tipo:
if (intents==#search) then "OK"

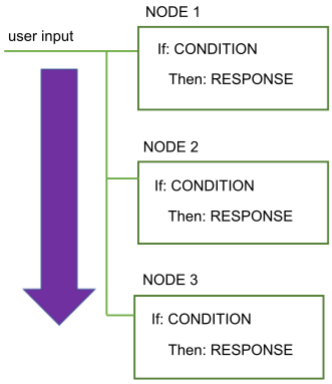


Figura 3.6: Albero di Dialogo e i costrutti if/then dei suoi nodi

E' possibile testare il comportamento del classificatore simulando le richieste utente, il classificatore stesso a quel punto restituirà l'intento riconosciuto e la risposta associata se prevista. Dopo aver configurato l'intero ecosistema si effettua il deploy del servizio e alla fine di questa fase sono restituiti credenziali ed endpoint con cui raggiungerlo. I dati in figura 3.7 sono quelli utilizzati in Node-Red per configurare i nodi dedicati al Conversation.

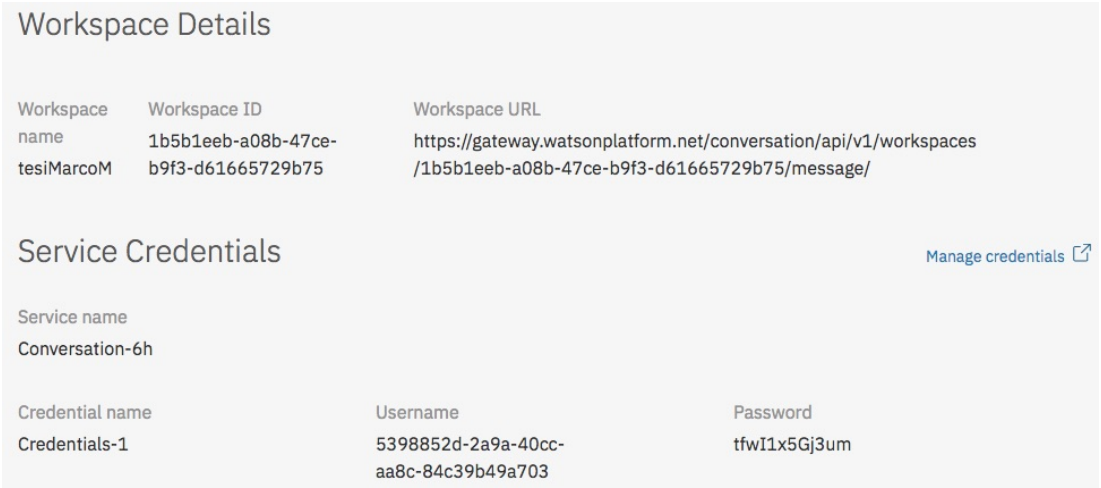


Figura 3.7: Schermata di deployment di un workspace in IBM Bluemix Conversation

In Listing 3.1 e Listing 3.2 sono rappresentate rispettivamente un esempio di richiesta digitata da un utente mediante chat e inoltrata tramite REST API al servizio di cui si è effettuato il deploy e la relativa risposta del servizio.

```
{"input":
        {
        "text": "Cosa prevede il decreto Bersani per la polizza
            auto?"
        }
}
```

Listing 3.1: Esempio di Request inoltrata al Conversation


```
{
  "intents": [{
    "intent": "faq1",
    "confidence": 0.9289345741271973
  }],
  "entities": [],
  "input": {
    "text": "Cosa prevede il decreto Bersani per la polizza
       auto?"
  },
  "output": {
    "text": ["Il Decreto Bersani stabilisce che chi compra un
       veicolo (nuovo o usato) puo' usufruire della stessa
       classe di merito che ha gia' maturato per un altro
       veicolo della medesima tipologia (ad es. autovettura con
        autovettura). Tale diritto e' applicabile ai soli
       proprietari persona fisica ed esteso ai familiari
       conviventi presenti nello stato di famiglia."],
    "nodes_visited": ["node_1_1518562022203"],
    "log_messages": []
  },
  "context": {
    "conversation_id": "ec3d278a-cd60-486a-ba5e-37cd282d07a1",
    "system": {
      "dialog_stack": [{
```

```
        "dialog_node": "root"
      }],
      "dialog_turn_counter": 1,
      "dialog_request_counter": 1,
      "_node_output_map": {
        "node_1_1518562022203": [0]
      },
      "branch_exited": true,
      "branch_exited_reason": "completed"
    }
  }
}
```


Listing 3.2: Response ricevuta dal Conversation associata alla Request contenente il testo:"Cosa prevede il decreto Bersani per la polizza auto?"

dove:

- **input:** è il testo digitato dall'utente in chat
- **output:** è la risposta che il bot fornisce quando viene visitato un nodo del dialogo
- **intents:** contiene l'insieme di intenti riconosciuti nella frase; l'output è relativo al primo nodo visitato che verifica la condizione su un intento o un'entità.
- **entities:** sono le entità riconosciute nell'input
- **context:**mantiene le proprietà del contesto, tra cui l'id della conversazione.

Come visto nel paragrafo 3.3.1 risposte del genere, non riconducibili all'intento **#search**, sono automaticamente restituite all'utente mediante la chat di Telegram senza alcuna elaborazione aggiuntiva da parte dell'architettura Node-Red, In Fig. 3.8 un esempio di interazione con il bot sulla base del riconoscimento di un*intent* di tipo *#faqId*. Al contrario, se riconosciuta la volontà di ricerca, il testo della risposta è ignorato e si procede nel percorrere il flusso. Per chiarezza, un esempio di Request contenente una frase riconducibile all'intento di ricerca e la relativa risposta sono descritti al Listing 3.3 e 3.4.

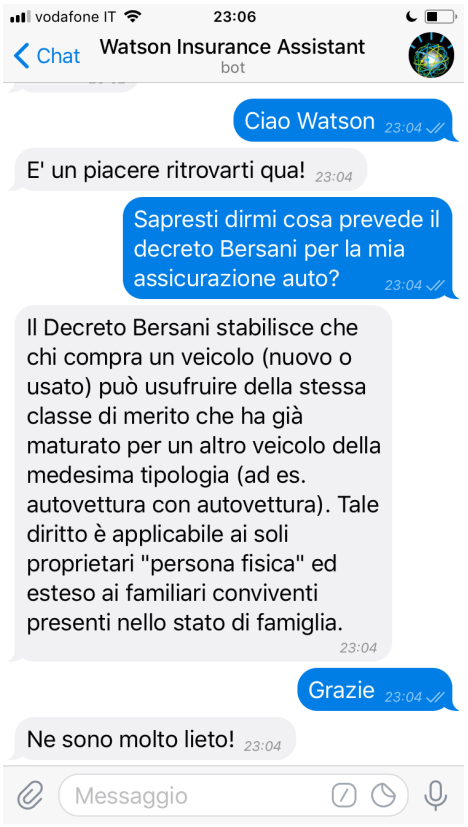


Figura 3.8: Restituzione della risposta ad una FAQ mediante il bot Telegram

```
{"input":
        {"text":"cerco un'assicurazione per il condominio che
           amministro"}
}
```

Listing 3.3: Esempio di Request al Conversation contenente una frase riconducibile all'intento di ricerca

```
{
  "intents": [{
    "intent": "search",
    "confidence": 0.2298691177328948
  }],
  "input": {
    "text": "cerco un'assicurazione per il condominio che
       amministro"
  }
  ...
```

```
}
```

Listing 3.4: Response a seguito di una Request riconducibile all'intento #search

In questo caso è l'input dell'utente l'entità che prosegue nel flusso e, inviato al service layer per essere arricchito, viene inoltrato al motore di ricerca per l'ottenimento dei risultati poi consegnati all'utente mediante l'interfaccia della chat.

3.3.3 Watson Content Analytics Studio

Watson Content Analytics Studio è il modulo che si occupa di costruire la conoscenza da iniettare al motore di ricerca. WCA Studio, in particolare, offre strumenti per realizzare motori di analisi del testo non strutturato specializzati per un particolare dominio di interesse e che risultano molto più potenti rispetto a quelli visti fino a qui. Ogni dominio tipicamente ha il suo insieme di termini rilevanti e alcuni di questi cambiano significato se fuori dal dominio stesso, così come una particolare sequenza di parole in un contesto può significare qualcosa di rilevante, mentre in un altro può essere completamente irrilevante. Sono forniti strumenti per potenziare le tecniche di analisi del testo permettendo la creazione di dizionari ad-hoc per un particolare dominio e quando in un testo, che viene analizzato dal motore generato da WCA Studio, appare uno dei termini presenti nel dizionario, questo viene annotato con la sua forma base e con il nome del dizionario di appartenenza. Oltre che dai termini di una frase, le annotazioni possono essere associate a particolari sintassi quali sequenze di numeri e/o caratteri oppure l'ordine di apparizione di particolari termini come i pattern che indicano la presenza di un numero di telefono in un testo, di una targa o ancora di un indirizzo e che spesso variano di stato in stato. Per l'estrapolazione di queste espressioni domain-specific, WCA Studio fornisce una UIMA pipeline che consiste di quattro annotatori UIMA (si veda la figura 3.9).

Ogni annotatore è progettato per essere data-driven affinchè l'utente possa plasmarlo al dominio target creando risorse domain-specific, come dizionari e regole. Nello specifico i Task supportati da Content Analytics Studio sono:

- Costruzione di **Dizionari** di termini domain-specific sotto forma di tabelle di un DataBase. Tutte le entry di un dizionario possono possedere solamente una forma normale mentre possono essere associate

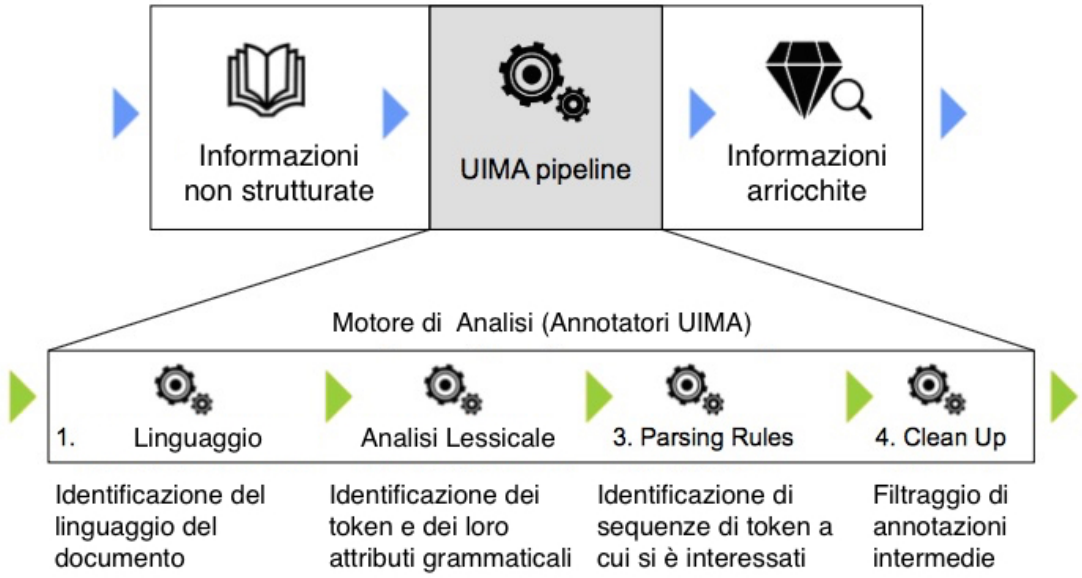


Figura 3.9: Esempio di struttura logica di un documento convertito in formato CAS [15]

a più di una proprietà mediante la creazione di Colonne aggiuntive. I dizionari sono risorse appartenenti ad annotatori Lessicali.

- Sviluppo di **Character rule** per riconoscere pattern di caratteri che rappresentano tipi specifici di informazioni, come numeri di telefono o indirizzi email. Anche le Character Rule fanno parte degli annotatori Lessicali di una pipeline UIMA.

- Sviluppo di **Parsing rule** per riconoscere pattern di parole che possano rappresentare particolari concetti, come ad esempio i nomi di persona [Posto che un token identifica un nome grazie a un dizionario Nomi e che il secondo token identifica invece un cognome grazie al corrispettivo dizionario, è possibile costruire una parsing rule secondo la quale un token annotato *Nome* che ne precede un altro annotato invece *Cognome* corrisponde ad un'altra entità che verrà annotata *Persona*]. Le Parsing Rule costruiscono regole a partire da annotazioni prodotte da annotatori precedenti all'interno della pipeline UIMA e per questo rappresentano una categoria a parte all'interno della stessa.

- Analisi dei documenti di test per annotarne il testo e controllare il contenuto associato ad ogni annotazione

- **Export** della pipeline UIMA direttamente sulla collezione predisposta sul server di Watson Content Analytics e generazione automatica di *fields* e *facet* direttamente mappate alle annotazioni esportate.

La pipeline di annotazione, una volta esportata su una collezione, è utilizzata insieme agli strumenti di annotazione di default per annotare i documenti durante la fase di *indexing* e dalle Real Time NLP API di WCA per effettuare l'analisi di un testo o di un contenuto binario inviato in input alla stessa. Le *facet* (descritte nel par. 3.3.4) sono popolate durante questa fase e corrispondono all'astrazione utilizzata per rappresentare le annotazioni nel testo; al Listing 3.8 un esempio di analisi di una query utente con la restituzione delle *text_facet* rilevate. Le annotazioni accrescono la conoscenza del dominio del motore di ricerca e nel caso delle NLP API restituiscono un'analisi contenente informazioni trasportate da tale conoscenza altrimenti mancanti. Come sarà possibile vedere nella sezione dedicata al Service Layer sviluppato, le NLP API e, quindi, le annotazioni sono state fondamentali per l'arricchimento delle query allo scopo di renderle adatte al dominio. Nella costruzione dei dizionari per l'analisi lessicale del testo si è scelto di replicare la struttura del sito della compagnia che è suddivisa in Aree tematiche ognuna delle quali è a sua volta suddivisa in sotto aree che raggruppano tutte le pagine relative a informazioni/prodotti che vi appartengono. Una parte di questa struttura è visibile nell'immagine 3.10 dove è possibile osservare le Aree tematiche del portale e l'insieme delle sotto aree definite per l'Area **Mobilità**. Per replicare questa struttura si è scelto

Figura 3.10: Aree e Sottoaree tematiche del portale oggetto di studio

quindi un **modello tassonomico** e i dizionari implementati sono tre:

1. **Area:** Nelle entry del dizionario Area sono presenti termini quali: Mobilità, Casa, Lavoro, Protezione, Risparmio e tutte le loro forme inflesse (Case, Lavori, Protezioni, Risparmi).

2. **Sottoarea:** Le entry di questo dizionario sono relative ai termini utilizzati per descrivere le sotto aree del portale. Per questo dizionario è stata creata anche la colonna Area all'interno della quale ad ogni entry viene associata anche l'area a cui la sottoarea appartiene (La entry Nautica di Sottoarea è associata all'Area Moblilità).

3. **Elemento:** Questo dizionario continene tutti i termini specifici del dominio ed è quello da cui viene generata la conoscenza. Per questo dizionario oltre alla colonna Area è stata creata anche la colonna Sottoarea che ospiterà come intuibile la sottoarea di appartenenza (Parte del dizionario è rappresentato in Figura 3.12).

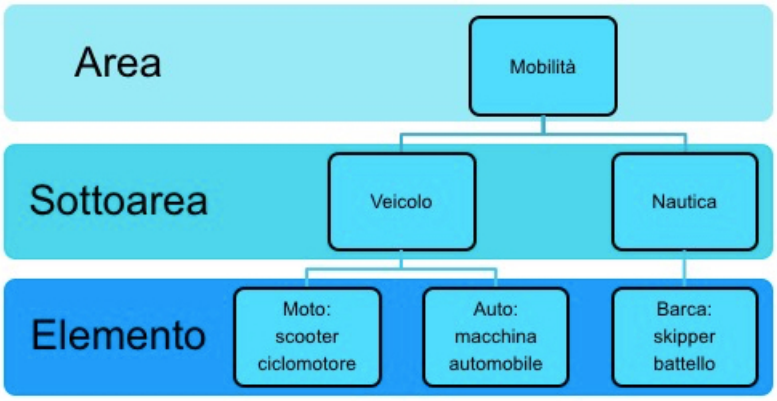


Figura 3.11: Modello tassonomico utilizzato per l'analisi sintattica del testo custom realizzato mediante WCA Studio

E' la gerarchia in questione che permette la restituzione nella result page di Watson non soltanto di tutti quei documenti contenenti esattamente le parole espresse nella query utente ma anche di tutte quelle pagine le cui tematiche sono legate ad esse poiché avviene il matching tra le *textfacet* contenute nella query utente e quelle contenute nei documenti dell'indice. Non è perciò, al momento, prevista l'implementazione di *character rules* e *parsing rules* che invece possono essere utilizzate per gli sviluppi futuri che verranno discussi nel capitolo dedicato.

Element	Part of Speech	Area	SottoArea
Elemento 1 - 58. aeromobile - vita smart			
aeromobile	Noun	Mobilità	Viaggio
agricoltura	Noun	Lavoro	Agricoltura
agriturismo	Noun	Lavoro	Agricoltura
attività smart	Noun	Lavoro	AttivitàSmart
auto	Noun	Mobilità	Veicolo
azienda	Noun	Lavoro	Impresa
bagaglio	Noun	Mobilità	Viaggio
barca	Noun	Mobilità	Nautica
casa	Noun	Casa	Casa
casa smart	Noun	Casa	Casa
cassa forense	Noun	Professione	Lavoro
circolazione	Noun	Protezione	Infortunio
condominio	Noun	Casa	Condominio
constatazione amichevole	Noun	Mobilità	Veicolo
contact center pronto assistance servizi	Noun	Assistenza	PAS

Figura 3.12: Parte del dizionario Elemento

3.3.4 Watson Content Analytics

IBM Watson Explorer Content Analytics permette di raccogliere e analizzare sia contenuti strutturati che non strutturati in documenti, email, database, siti web o altre repository di varia natura; è una suite completa per il crawling, il parsing e l'analisi del contenuto dei documenti e per la creazione di un indice ricercabile. WCA fornisce quindi tutte le tecniche necessarie per analizzare i dati di una collezione e li rende disponibili per la ricerca; la sua architettura è mostrata in figura 3.13.

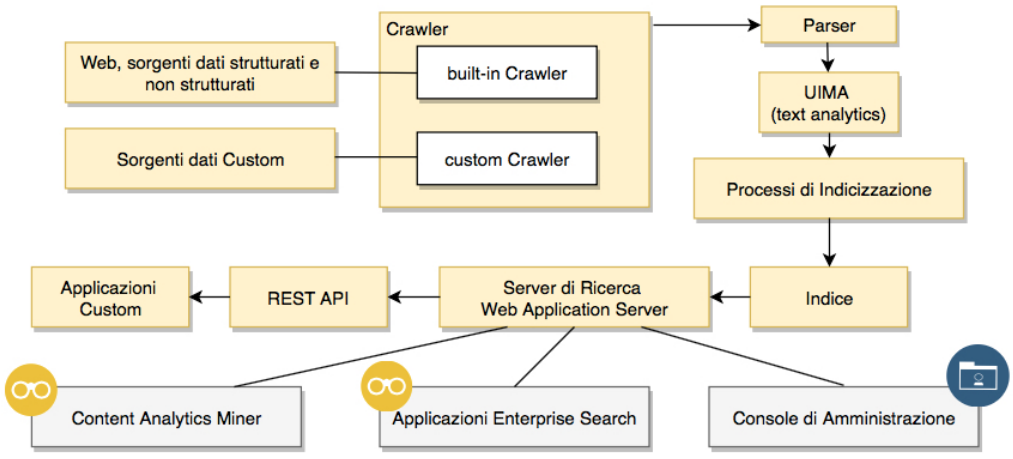


Figura 3.13: Architettura di Watson Explorer Content Analytics [15]

Il modello dati su cui si fonda WCA è invece rappresentato in figura 3.14. L'entità fondamentale in WCA è il *crawled document* che consiste del suo contenuto originale e di uno o più metadati associati; un documento logico

è creato dal documento originale e da qualsiasi altra informazione che lo arricchisce e che è stata aggiunta durante la fase di processamento, esso consiste di campi e di facet che sono popolate o dal contenuto originale del documento oppure dagli annotatori con le informazioni ottenute a seguito dell'analisi testuale. Le ***Facet*** sono molto simili agli *index field* ma sono state create per rappresentare diverse dimensioni, o viste, tra tutti i documenti di una stessa collezione e permettono di agire sulla ricerca restringendo il corpus della collezione preventivamente aumentandone la precisione.

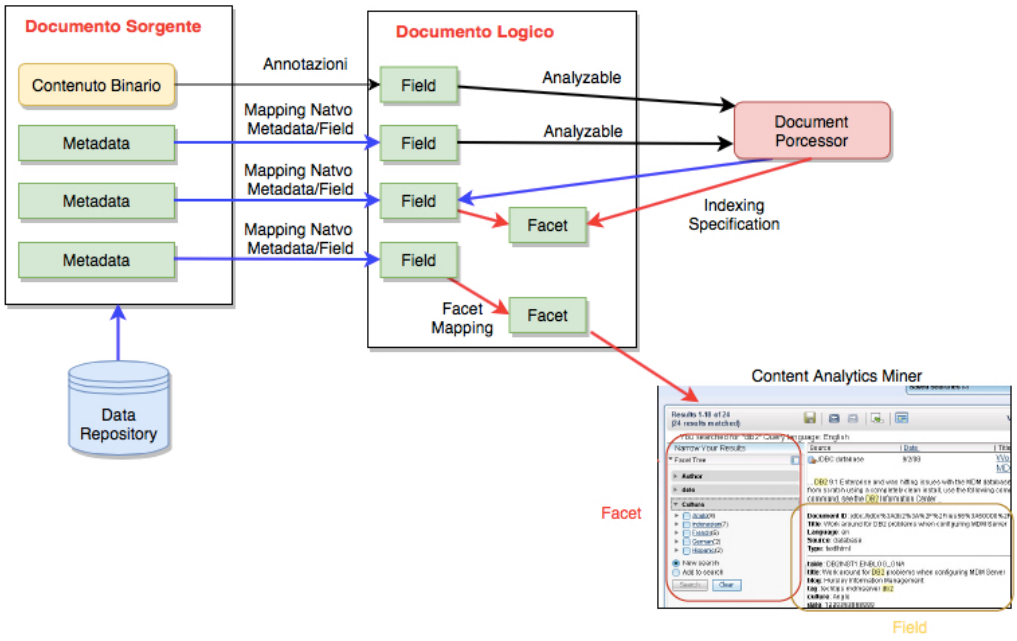


Figura 3.14: Data Model utilizzato da WCA per le sue collezioni [15]

Sono disponibili diversi ***Crawler*** per le più comuni sorgenti dati ed è fornito un SDK completo in Java per lo sviluppo di Custom Crawler o di Plugin per manipolare il comportamento di base di un crawler standard. In tutti i casi ciò che viene raccolto è l'insieme dei dati *crudi*(in inglede *raw*) che sono successivamente elaborati dal *Parser*. Il ***Parser*** identifica la codifica e il tipo di contenuto di ogni documento raccolto per estrarne il testo e convertirlo dalla sua forma nativa in un documento in formato CAS pronto per poter essere elaborato. Esso si serve infatti di un annotatore, o meglio di una pipeline di annotazione, per arricchire il testo con particolari *tag* che veicolano informazioni fondamentali per il data retrieval (sopratutto se in uno specifico dominio). Gli annotatori identificano la lingua in cui è scritto il documento e suddividono il contenuto in token per la fase di analisi testuale detta anche *text-analysis*; i token sono la base di partenza per la rilevazio-

ne di costituenti, per la realizzazione del POS-tagging e per l'estrazione automatica di Named Entity come Nomi, Località e Aziende. Le funzionalità degli annotatori WCA possono essere estese da pipeline di annotazione personalizzate importate mediante tool esterni come ICA Studio e come descritto nel paragrafo precedente. Sia gli annotatori built-in che le pipeline custom rappresentano i documenti processati nel formato *Common Analysis Structure* (**CAS**), una struttura dati object-based ormai standard per gli annotatori di text anaytic object; questa struttura contiene logicamente il documento da analizzare ed è la rappresentazione con cui i motori di analisi presentano e condividono i loro risultati. In CAS è possibile rappresentare oggetti, proprietà e valori e diversi tipi di oggetti possono essere messi in relazione in un'unica gerarchia (Figura 3.15).

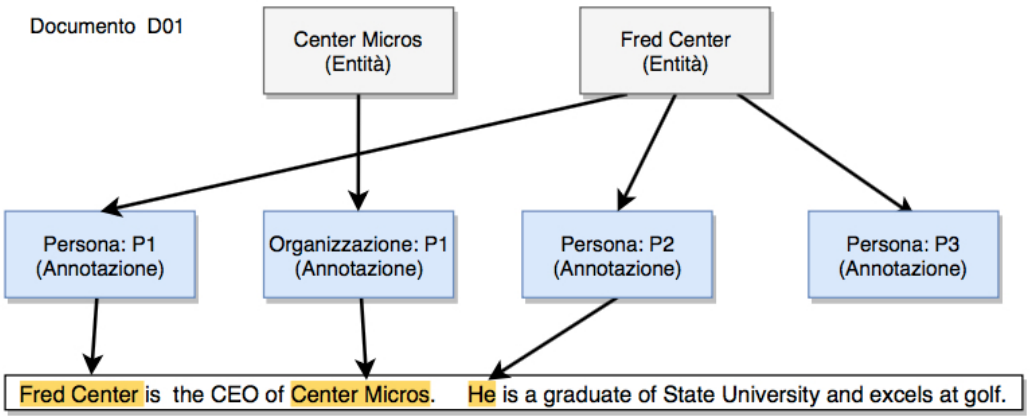


Figura 3.15: Esempio di struttura logica di un documento convertito in formato CAS [15]

WCA utilizza proprio CAS come formato per mantenere le informazioni risultanti dell'analisi dei documenti; in particolare ogni annotatore della pipeline arricchisce con le proprie informazioni il CAS che di volta in volta viene reso disponibile all'annotatore successivo della pipeline. L'output della UIMA pipeline è un CAS che contiene il contenuto del documento originale arricchito dalle annotazione aggiunte da tutti gli altri annotatori. Tutto questo è permesso dal framework UIMA che implementa ed espone interfacce verso il CAS. Le informazioni vengono poi indicizzate, in modo tale che gli utenti possano interrogare l'indice stesso attraverso le loro applicazioni di ricerca. Utilizzando query arricchite con lo stesso strumento di annotazione la conoscenza diviene condivisa tra l'utente e il motore che è possibile dire "parlino" la stessa lingua. Il numero di documenti rilevanti,

grazie a questa conoscenza, aumenta. L'***Indexer*** è il componente responsabile della costruzione dell'indice; l'indice di Watson Content Analytics si basa sull'indexer open-source *Apache Lucene* estendendone le funzionalità. Per l'esplorazione e l'interrogazione dell'indice possono essere utilizzati l'Enterprise Search Application o il Content Analytics Miner, i tool grafici di base forniti da Watson, o ancora è possibile utilizzare le API REST esposte dal prodotto per creare applicazioni custom. Le API esposte da WCA possono essere utilizzate per creare applicazioni di ricerca e di amministrazione; offrono, infatti, totale controllo all'utente per quanto riguarda la gestione delle collezioni, il monitoring e la gestione delle componenti del sistema, il push di nuovi documenti nelle collezioni, la ricerca di documenti all'interno delle collezioni e la ricerca tra le facet di una collezione. Queste API prendono il nome di Discovery REST API e sono organizzate in Search REST API e Admin REST API; entrambe accettano e supportano come tipo di ritorno dati sia in formato XML che JSON. Le API di maggiore interesse per il progetto sono :

- **/api/v10/query/typeahead**: utilizzata per il servizio di completamento e suggestion di una query utente
- **/api/v10/search**: utilizzata per il servizio di ricerca a partire da una query utente
- **/api/v10/analysis/text**: fornisce l'analisi lessicale della query utente

Le pipeline utilizzate per migliorare l'estrazione delle informazioni possono essere popolate mediante differenti approcci ed esistono strumenti adatti per creare soluzioni che seguano uno o una combinazione di essi:

- Rules-Based: questo approccio è basato sulla costruzione di regole che se soddisfatte permettono di riconoscere particolari entità all'interno del testo. E' l'approccio più semplice ed efficace per corpus non molto estesi sebbene difficile da mantenere quando la dimensione e la complessità del corpus cresce a dismisura. IBM permette la creazione di annotatori rule-based mediante Watson Explorer Content Analytics Studio
- Machine Learning: una approccio valido per corpus molto estesi è quello machine-learning che utilizza modelli statistici per eseguire analisi in linguaggio naturale anche se difficili da debuggare; come tutte le soluzioni machine-learning di tipo supervisionato prevede una

fase di apprendimento mediante un training set generalmente composto da frasi di esempio appartenenti al dominio di interesse e richiede la costruzione di un ground truth. Watson Conversation è un valido strumento che permette la creazione di soluzioni che seguono questo tipo di approccio.

- Soluzioni Ibride: esistono strumenti che sfruttano entrambe gli approcci per creare modelli più complessi e scalabili. Le regole solitamente vengono utilizzate per accelerare il training e migliorare la precisione dei modelli machine-learning. Nella suite Watson questo lavoro è svolto da Watson Knowledge Studio.

Dopo che i documenti sono stati elaborati dal Crawler e successivamente analizzati viene creato l'indice invertito della collezione e il processo di indicizzazione arricchisce in modo incrementale le informazioni sui documenti che sono stati aggiornati, aggiunti o eliminati dall'indice per assicurarsi che gli utenti finali abbiano accesso sempre alle informazioni più aggiornate. All'atto di costruzione di un indice, il processo di indicizzazione realizza un'analisi globale dei documenti e durante questa fase vengono applicati algoritmi per l'identificazione di documenti duplicati e per la ricostruzione della struttura dei collegamenti sopratutto nelle pagine web. Le collezioni di Watson prevedono indici partizionati per il bilanciamento del sistema affinchè ogni partizione gestisca all'incirca lo stesso numero di documenti; l'accesso alle partizioni è trasparente agli utenti che vedono un unico grande indice logico.

Il *Search run-time* è il componente server responsabile di servire le richieste di ricerca e di analisi dagli utenti; queste richieste sono eseguite mediante REST API e sia il Content Miner che l'Enterprise Search Application sono due esempi di applicazioni che possono essere costruite utilizzando solamente queste API. I risultati sono restituiti seguendo il ranking calcolato a partire dalla query utente e questo può essere in parte influenzato mediante dei fattori di boost applicabili a particolari campi di ricerca o direttamente ai termini della query. L'administration Console offre una visuale completa di queste componenti e delle modalità di configurazione per ognuno di essi Per l'implementazione della soluzione in esame è stata creata una Content Analytics collection, il tipo di collezione che più si adatta alle finalità del progetto per le sue potenzialità analitiche, da ora identificata con il suo id **taxonomy_collection**. La collezione è vuota finchè non viene definito e avviato almeno un **Crawler** insieme all'**Indexer** e al *Search run-time Server*. *Crawler, Indexer/Parser e Search runtime* sono componenti del prodotto

completamente gestibili dalla console di amministrazione (visibile in figura 3.16).

Figura 3.16: Administration Console di IBM Watson Content Analytics

Per l'indicizzazione dei contenuti delle pagine Web del portale assicurativo, la collezione **taxonomy_collection** è stata configuranta con un crawler di tipo Web. La console di amministrazione consente sia di modificare le proprietà del Crawler che il suo spazio di indirizzamento. Tale spazio va configurato specificando l'*Url iniziale*, ovvero l'URL a partire dal quale il crawler segue i collegamenti per raggiungere tutte le altre pagine del portale. In questa sezione è possibile anche limitare i tipi di documento da includere/escludere, il livello massimo di percorso dell'URL da visitare e le impostazioni di connessione per l'eventuale passaggio da un proxy; è anche possibile creare delle regole affinchè vengano permesse o filtrate pagine caratterizzate da particolari prefissi HTTP e/o URL appartenenti a particolari domini. In questo caso sono stati ammessi tutti i tipi di file interni al portale ed escluse tutte le pagine web esterne al dominio web della compagnia assicurativa. Nelle proprietà del Crawler va specificato invece un eventuale Plugin; questo permette di personalizzare il comportamento del crawler prima (PreFetch-Plugin) e dopo (PostParse-Plugin) che venga effettuato il Fetch della pagina web che a questo punto può subire ulteriori elaborazioni.

La sezione dedicata all'*Indexer* permette di avviare il server di indicizzazione e di rigenerare l'indice qualora fosse stata modificata la configurazione dello stesso senza dover effettuare nuovamente il crawling dell'intero corpus. Mediante la console di amministrazione è possibile creare nuovi campi di indicizzazione, nuove facet, gestire anche la gerarchia delle facet generate e mappare i metadati estratti dai documenti, in questo caso le pagine HTML e i documenti PDF incontrati durante il crawling, con i campi creati ad-hoc per la collezione. Ogni campo può essere configurato per essere restituibile nei risultati di ricerca, per essere visibile nel riepilogo dei risultati; può anche essere impostata la ricerca per campi affinchè si ricerchi tra i documenti che possiedono un particolare valore di un campo.

E' un comportamento simile a quello della ricerca per facet ma mentre la ricerca per campi esegue la ricerca sul corpus per intero il secondo lo filtra restringendolo prima di effettuare la ricerca effettiva. Alcuni campi di indicizzazione per la collezione *tpd_taxonomy* sono stati creati automaticamente nella fase di esportazione della pipeline di annotazione di ICA Studio e sono indicati nella console di admin sotto la voce Origine di immissione con il valore CAS-indice; questi campi sono:

- **area**
- **sottoarea**
- **elemento**

Altri campi sono invece creati di default e corrispondono al contenuto testuale del documento, il suo titolo nonché la sua URI e sono rispettivamente: **body, title e docID**. L'ultimo tipo di campi è quello creato manualmente e popolato con i metadati estratti dal documento. Tutti i campi precedentemente descritti sono popolati durante la fase di indicizzazione insieme alle facet.

3.3.5 Il problema della BoilerPlate extraction

Per il problema dell'esclusione delle componenti superflue dalla pagina web discusso al paragrafo 2.1.4 (problema del boilerplate) che rischia di alterare il ranking dei risultati del search engine è stato realizzato un PostParse plugin che fa utilizzo della libreria boilerpipe a tale scopo. I PostParse Plugin lavorano dopo il Fetch di una pagina Web da parte del Crawler Web e appena prima dell'operazione di Parse eseguita invece dall'Indexer. Boilerpipe, invece, è una libreria Java rilasciata sotto licenza Apache 2.0 e distribuita da Google che fornisce algoritmi per la rilevazione e la rimozione del contenuto testuale superfluo (non principale) all'interno di una pagine Web come i Menu, gli Header, Footer e tutti i componenti generalmente associabili a particolari template di design. L'estrazione del contenuto avviene nell'ordine dei millisecondi e richiede semplicemente il documento in input per ottenerlo nella forma filtrata dai dati superflui. La logica del codice del plugin è descritta a seguire:

```
PostparsePluginArg1 arg = (PostparsePluginArg1)arg0[0];

//ottengo contenuto binario del documento
```

```
byte[] content=arg.getContent();

doc_filtered=DefaultExtractor.INSTANCE.getText(content);
byte[] b = doc_filtered.getBytes();
arg.setContent(b);    //nuovo contenuto da indicizzare
arg.setSave(true);    //permetti di idicizzare il documento
```

Listing 3.5: Logica applicativa alla base del PostParsePlugin realizzato per la risoluzione del problema al problema della BoilerPlate extraction.

BoilerPipe offre diversi tipi di extractor, DefaultExtractor è il più generico ed è quello più adatto per i dati provenienti da più sorgenti e per questo motivo è stato scelto. La classe PostparsePluginArg1 fa parte delle API offerte da WCA e permette di operare sul contenuto di cui il crawler ha eseguito il Fetch; la funzione getContent permette di estrarre il contenuto binario della pagina in un array di byte, al contrario setContent definisce il nuovo contenuto dato in input lo stesso tipo di array. setSave impone al crawler se indicizzare la pagina processata o se scartarla dall'indice dopo l'elaborazione; in questo caso dopo aver filtrato la pagina dal contenuto testuale superfluo si vuole la sua indicizzazione.

3.3.6 Service Layer

Per fare query all'altezza del livello di conoscenza acquisito dal motore di ricerca, grazie all'azione della pipeline di annotazione, si è reso necessario lo sviluppo di un livello di integrazione aggiuntivo. Quest ultimo è stato sviluppato per prendersi carico delle richieste digitate dall'utente per elaborarle e inoltrarle a WCA e, viceversa, per inoltrare la risposta ottenuta all'utente; tale risposta verrà poi formattata nella forma ritenuta più opportuna dall'interfaccia utente scelta. Il Service Layer definisce i metodi a seguire per la costruzione degli endpoint da richiamare durante l'elaborazione e che rispettivamente espongono le API di search, RealTimeNLP e typeahead per il suggerimento delle query :

```
1 private String getSearchEP(String input, int page)
2 private String getAnalysisEP(String input)
3 private String getATypeaHeadEP(String input)
```

Listing 3.6: Metodi costruttori degli URL da richiamare per servizi di search, analysis e typeahead

getSearchEP, getAnalysisEP e getATypeaHeadEP sono di appoggio agli endpoint /search e /suggestion, esposti dal service layer e sono alimentati dalle informazioni contenute nel file **application.properties**:

```
server.port=8080
server.connection-timeout=10000

app.wex.collection=unipol_taxonomy
app.wex.port=8393
app.wex.host=localhost

#Search properties
app.wex.search.enablehref=true
app.wex.search.output=application/json
app.wex.search.pagesize=3

#Analysis
app.wex.collection.analysis=unipol_taxonomy
app.wex.analysis.output=application/json

#Type a Head
app.wex.typeAhead.output=application/json
app.wex.typeAhead.lang=it_IT

#Formattare come JSONArray
app.wex.mykey=["area","area.sottoarea","area.sottoarea.elemento
   "]
app.wex.mykey.format=[["area"],["area","sottoarea"],["area","
   sottoarea","elemento"]]
```

Listing 3.7: File di configurazione application.properties da cui i metodi getSearchEP, getAnalysisEP e getTypeaHeadEP attingono per la costruzione degli URL.

Per quanto riguarda il servizio di type-ahead delle query utente, si è realizzato ed esposto il metodo *suggestionAPI* che si limita a richiamare l'API esposta da Watson mediante le librerie **java.net** man mano che si digita un nuovo carattere. La risposta del server, di cui si effettua il parsing in un JSONObject delle librerie **org.json**, è restituita al front-end che si occupa di estrarne solo il contenuto testuale suggerito all'utente. Di seguito parte

del codice scritto allo scopo:

```
@PostMapping(value = "/suggestion", produces = "application/json")
public @ResponseBody String getSuggestion(@RequestParam(required =
    true) String input)
{
        JSONObject res = callTypeHead(input);
        result = res.toString().replaceAll("#", "");
        return result;
}

private JSONObject callTypeHead(String input)
{
        JSONObject result = null;
        input = input.replaceAll(" ", "+");
        URL type_head = new URL(getATypeaHeaedEP(input));
        HttpURLConnection connection = (HttpURLConnection)
            type_head.openConnection();
        connection.setRequestMethod("POST");
        BufferedReader br = new BufferedReader(new InputStreamReader
            ((connection.getInputStream()),"UTF-8"));
        String serverOutput = "";
        String line = null;
        while ((line = br.readLine()) != null)
                serverOutput += line;
        result = new JSONObject(serverOutput);
        return result;
}
```

Per quanto riguarda il servizio di ricerca viene esposto come risorsa HTTP invece il metodo bean /search che, presi in ingresso la query dall'utente e il numero di result page desiderato, restituisce un oggetto JSON cotenente la risposta ottenuta da WCA; il settaggio di *page* permette lo scorrimento della SERP.

```
@CrossOrigin
@PostMapping(value = "/search", produces = "application/json")
public @ResponseBody String callSearchWCA(
        @RequestParam(required = true) String input,
        @RequestParam(required = false, defaultValue = "1") int page)
```

Il Bean *search* richiama al suo interno callSearchWCA. E' al suo interno che avviene la costruzione della query arricchita; in particolare il metodo getAnalysis, che si fa carico di richiamare il servizio di analisi offerta dal WCA, formatta e restituisce un JSONObject contenente i risultati dell'analisi stessa. Si è scelto di utilizzare oggetti JSON piuttosto che una semplice stringa contenente la query arricchita per offrire la possibilità di portarsi dietro altre informazioni importanti ottenute dall'analisi e che, possibilimente, possano abilitare a sviluppi futuri. Ad esempio è possibile riconoscere mediante la pipeline di annotazione particolari keyword o pattern riconducibili a intenzioni dell'utente; tale informazione dovrebbe essere restituita a front-end per la sua gestione. La logica applicativa di callSearchWCA è, comunque, brevemente riassunta nel blocco a seguire e consta nell'estrarre la query arricchita da getAnalysis, convertirla in un formato ASCII safe per gli URL, realizzare la chiamata POST all'endpoint di search del WCA e infine, dopo averlo parsificato, restituire il risultato all'utente finale. Per realizzare la POST request si è scelto di utilizzare la classe **java.net.HttpURLConnection** offerta direttamente dal **JDK**.

```
1  private JSONObject callSearchWCA(String input, int page)
2  {
3          JSONObject result = null;
4          JSONObject analysis = getAnalysis(input);
5
6          JSONObject es_query = analysis.getJSONObject("es_query");
7          String query = es_query.getString("query");
8          String url = getSearchEP(query, page);
9          url.encode();
10         URL search = new URL(url);
11         HttpURLConnection connection = (HttpURLConnection)
12                 search.openConnection();
13         connection.setRequestMethod("POST");
14         BufferedReader br = new BufferedReader(new InputStreamReader(
               (connection.getInputStream()), "UTF-8"));
15         String serverOutput = "";
16         String line = null;
17         while ((line = br.readLine()) != null)
18                 serverOutput += line;
19         result = new JSONObject(serverOutput);
20         return result;
21 }
```

Listing 3.8: Logica applicativa del metodo CallSearchWCA

Il risultato è inoltrato così come restituito da WCA all'utente finale. La logica applicativa scritta per **getAnalysis** verifica l'esistenza di annotazioni per la query utente e ne tiene in memoria una loro rappresentazione mediante una collezione; in particolare viene definita la Map< path, List< keyword, begin, end > > ***analysisStruct*** che, ad ogni path, ovvero ad ogni tipo di annotazione rilevata nella frase, associa l'insieme delle annotazioni realizzate sul testo. La rilevazione di queste annotazioni avviene mediante l'elaborazione del JSON body restituito dal servizio di analisi del WCA e di cui si ha un esempio a seguire per la query "*cerco assicurazioni per skipper*" in input. Le annotazioni sono visibili all'interno dell'Array JSON "textfacets" appartenente all'oggetto metadata.


```
{
    "metadata": {
        "textfacets": [
    {
        "path": ["_phrase", "noun_phrase", "nouns"],
        "end": 19,
        "keyword": "cerca assicurazione",
        "begin": 0
    },
    {
        "path": ["_word","noun","general"],
        "end": 19,
        "keyword": "assicurazione",
        "begin": 6
    },
    {
        "path": ["area"],
        "end": 31,
        "keyword": "Mobilita",
        "begin": 24
    },
    {
        "path": [ "area","sottoarea"],
        "end": 31,
```

```
        "keyword": "Professione",
        "begin": 24
    },
    {
        "path": ["area","sottoarea"],
        "end": 31,
        "keyword": "Nautica",
        "begin": 24
    },
    {
        "path": ["area", "sottoarea", "elemento"],
        "end": 31,
        "keyword": "professionista",
        "begin": 24
    },
    {
        "path": [ "area", "sottoarea", "elemento"],
        "end": 31,
        "keyword": "barca",
        "begin": 24
    },
    {
        "path": [ "_word","noun","general"],
        "end": 31,
        "keyword": "skipper",
        "begin": 24
     },
    {
        "path": ["area"],
        "end": 31,
        "keyword": "Lavoro",
        "begin": 24
        }
]
}
}
```


Listing 3.9: Analisi di una query di esempio restituita dal motore di analisi di Watson Content Analytics

La costruzione della query arricchita, per come è stata pensata, è subordinata al raggruppamento di tutte le annotazioni associate alla stessa parola e c'è perciò la necessità di mantenere anche le informazioni riguardanti l'indice di inizio, *begin*, e di fine parola, *end*, a cui è associata l'annotazione stessa; da qui la presenza della lista List<keyword, begin, end> all'interno della *analysisStruct*. La mappa costruita a seguito dell'analisi per la query in esame è quindi quella di seguito:

Path	**Lista<Annotazioni>**
Area	{key:Lavoro, begin:24, end:31}, {key:Mobilità, begin:24, end:31}
Area.Sottoarea	{key:Professione, begin:24, end:31}, {key:Nautica, begin:24, end:31}
Area.Sottoarea.Elemento	{key:professionista, begin:24, end:31}, key:barca, begin:24, end:31
_word.noun.general	{key:assicurazione, begin:6, end:19}, {key:skipper, begin:24, end:31}
_phrase.noun_phrase.nouns	{key:cerca assicurazione, begin:0, end:19}

Tabella 3.1: Struttura di suporto al service layer per la costruzione della query semantica da inviare al motore di ricerca WCA

Data la presenza di un solo concetto del dominio all'interno della frase è evidente che le annotazioni di un particolare tipo previsto dalla tassonomia si riferiscano tutte alla stessa parola, ovvero *skipper*; sarebbe stato differente il caso in cui si fosse usata una query del tipo: "*vorrei assicurare la mia abitazione e la mia imbarcazione*". Ritornando alla *getAnalysis*, questo è quello che succede più nel dettaglio:

```
1  public JSONObject getAnalysis(String input){
2
3    JSONObject result;
4    JSONObject analysis = callWCAAnalysis(input);
5    Map<String, List<Element>> analaysisStruct = new HashMap<>();
6    List<Element> list;
7
8    if (analysis.has("metadata") &&
9          analysis.getJSONObject("metadata").has("textfacets")) {
10
```

```
11      JSONArray textfacets = new JSONArray(analysis.getJSONObject("
            metadata").getString("textfacets");
12
13      for (int i = 0; i < textfacets.length(); i++) {
14
15        JSONObject facet = textfacets.getJSONObject(i);
16        fillStruct(facet);
17        JSONObject query = buildSemanticQuery(struct, input);
18        result.accumulate("es_query", query);
19        return result;
20
21      }
22    }
23  }
```

Listing 3.10: Logica applicativa del metodo getAnalysis

La costruzione della query semantica nella forma in cui viene sottoposta all'API di search del Content Analytics avviene chiamando il metodo **buildSemanticQuery** con il supporto di **lemFilter**. Mediante **buildSemanticQuery** avviene la distinzione tra i concetti del modello tassonomico del dominio contenuti nella query digitata dall'utente e gli altri token che invece non veicolano alcun tipo di conoscenza ulteriore. Si estraggono dalla struttura di analisi tutti gli elementi (keyword,begin,end) associati alle *myKeyword* del modello tassonomico: [area,area.sottoarea,area.sottoarea.elemento] ovvero i concetti del dominio e le relative classi d'appartenenza. In questa fase avviene perciò l'estrazione della conoscenza del dominio mediante la costruzione di un piccolo dizionario domain-specific relativa alla query utente e che viene memorizzato nella Lista **concepts**. Gli elementi che man mano vengono aggiunti a questa lista vengono rimossi dalla struttura che alla fine di questa fase mantiene solamente le annotazioni prodotte dall'analisi lessicale e non di dominio; da questa vengono ulteriormente estratti i token della frase, eliminando invece costituenti inutili allo scopo, che sono memorizzati all'interno della lista **tokens**. L'analisi, come è possibile osservare dal Listing 3.9, restituisce le annotazioni realizzate dalla pipeline, il token relativo (o lemma) insieme agli indici di *begin* e di *end*. In base alla configurazione scelta durante l'esportazione della pipeline sul motore di analisi di WCA è possibile scegliere se questo debba restituire la forma base, o lemma, dei token oppure il covered text (il token stesso). Per il nostro intento si è scelto di avere a disposizione sia la forma base che il token stesso presentandoli come sinonimi e ricercare un concetto nella for-

ma (token OR lemma), infatti, in questa fase si controlla la presenza del token all'interno della struttura tra gli elementi appartenenti alla textfacet *area.sottoarea.elemento* e nel caso di assenza viene aggiunto dopo averlo estratto direttamente dalla query dell'utente mediante gli indici suddetti.

```
1  private JSONObject buildSemanticQuery(Map<String, List<Element>>
       struct, String input) {
2
3    List<Element> concepts = new ArrayList<>();
4    List<String> mykeyword = appConfiguration.getMyKeyWord();
5    List<String> format = appConfiguration.getMyKeyWordFormat();
6    for (int i = 0; i < mykeyword.size(); i++) {
7
8     String key = mykeyword.get(i);
9       if (struct.containsKey(key)) {
10        List<Element> elements = struct.get(key);
11        for (Element e : elements) {
12          if(!concepts.contains((Element) e))
13           concepts.add(e);
14          String elem = input.substring(e.getBegin(), e.getEnd());
15          Element element;
16          if(i == (mykeyword.size()-1)){
17            element = new Element("("+elem+")^10", e.getBegin(), e.
                 getEnd());
18            if(!concepts.contains(element))
19                          concepts.add(element);
20          }
21         }
22        }
23        JSONArray tokens = new JSONArray();
24        for (String key : struct.keySet()) {
25          if (key.indexOf("_word") == 0) {
26            String[] split = key.split("[.]");
27            String pos = split[1];
28            List<Element> list = struct.get(key);
29            for (Element e : list) {
30              String element=input.substring(e.getBegin(), e.getEnd())
                   ;
31              String lemma= e.getKey();
32              JSONObject token = new JSONObject();
33              token.accumulate("element", element );
```

```
34              token.accumulate("characterOffsetBegin", e.getBegin());
35              token.accumulate("characterOffsetEnd", e.getEnd());
36              token.accumulate("lemma", lemma);
37              token.accumulate("pos", pos);
38              tokens.put(token);
39              }
40            }
41          }
42        query = lemFilter(query, concepts, tokens);
43  }
```

Listing 3.11: Suddivisione dei concetti del dominio e dei token della query per la realizzazione della query semantica espansa mediante lemFilter

La funzione **lemFilter**, in breve, realizza:

1. Filtraggio basato su POStag non essenziali ai fini della ricerca: sono ammessi solamente *noun* / *adjective* / *concetti del modello*. Tag relativi ad un verbo generalmente indicano un'intenzione e qui l'intenzione è gia nota ed è la ricerca di qualcosa; altri Tag relativi ai costituenti della frase sono a loro volta ritenuti inutili.

2. Espansione dei token/concetti con lemma (token OR lemma)

3. Espansione dei concetti con elementi di livello superiore del modello concettuale (token OR iperonimo)

4. Assemblaggio dei termini espansi della query per l'ottenimento della query semantica.

In particolare si realizza la Collection **termTreeMap** per ospitare ogni componente della query già espansa con i relativi lemma o iperonimi posti in **OR** l'un l'altro. Attingendo da questa struttura, *lemFilter*, si occupa di porr

```
1  private String lemFilter(String query, List<Element> concepts,
       JSONArray tokens) {
2          String result = query;
3          String newQuery;
4          List<String> queryParts=new ArrayList<>();
5          Map<String, String> termTreeMap = new HashMap<>();
6          for (Element concept : concepts) {
7          String key=concept.getBegin()+"_"+concept.getEnd();
8          String value_part=concept.getKey();
```

```
9         String value="";
10        if (termTreeMap.containsKey(key)) {
11                value = termTreeMap.get(key);
12                value += " OR " + value_part;
13        }
14        else
15                value+=value_part;
16      termTreeMap.put(key, value);
17    }
18        for (int j = 0; j < tokens.length(); j++) {
19                JSONObject token = tokens.getJSONObject(j);
20                String word = token.getString("element");
21                int begin = token.getInt("characterOffsetBegin");
22                int end = token.getInt("characterOffsetEnd");
23        String lemma = token.getString("lemma");
24        String pos = token.getString("pos");
25        if(termTreeMap.containsKey(begin+"_"+end)){
26                String value=termTreeMap.get(begin+"_"+end);
27            value+=" OR "+word;
28            termTreeMap.put(begin+"_"+end, value);
29                        continue;
30           }
31                String queryPart = null;
32                if (posOK.contains(pos)) {
33                        if (!word.equals(lemma))
34                        queryPart = "(" + word + " OR " + lemma + ")
                              ";
35                        else
36                                queryPart =  word ;
37                        queryParts.add(queryPart);
38                }
39        }
40
41        for(String concept:termTreeMap.values()){
42                String queryPart = "(" +concept + ")";
43                if(!queryParts.contains(queryPart))
44                        queryParts.add(queryPart);
45        }
46        for (String key : queryParts) {
47                if (newQuery == null)
48                        newQuery = key;
```

```
49                  else
50                          newQuery += " AND " + key;
51          }
52          if (newQuery != null)
53                  result = newQuery;
54          return result;
55  }
```

Listing 3.12: Assemblaggio della query semantica mediante i concetti rilevati dall'analisi realizzata da WCA

Di seguito una rappresentazioe semplificata della classe Element realizzata per astrarre e contenere gli elementi risultanti dall'analisi del WCA caratterizzata quindi da parola/concetto *key,* indice *begin* di inizio token nella query e l'indice di fine *end*.

```
1  public class Element{
2
3          String key;
4          int begin;
5          int end;
6
7          public Element(String keyword,int begin,int end){
8                  key=keyword;
9                  this.begin=begin;
10                 this.end=end;
11         }
12 }
```

Il confronto tra il comportamento del motore di ricerca in assenza delle elaborazioni realizzate dal service layer interposto tra l'utente e Watson e il comportamento in sua presenza è marcato dalle figure 3.17 e 3.18. La query di riferimento è: "vorrei avere informazioni su assicurazioni per skipper".

Se il motore ha il comportamento di base le query sono inoltrate direttamente all'API di search di WCA; i token sono posti in AND e questo e il risultato ottenuto è quello in Fig.3.17. La query inoltrata è esattamente quella digitata dall'utente che equivale a :
"vorrei AND avere AND informazioni AND su AND assicurazioni AND per AND skipper".

Nella versione in cui si richiama il Service Layer implementato per l'arricchimento della query mediante l'espansione di quest'ultima si abilita an-

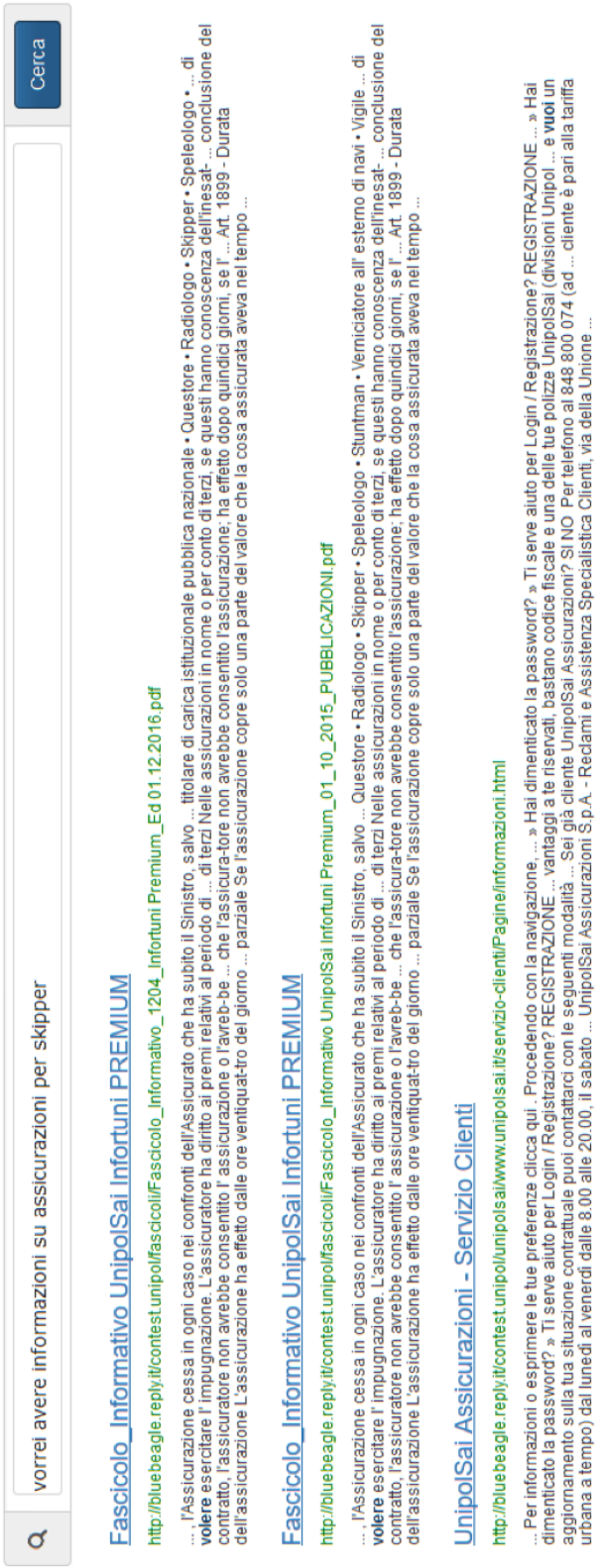


Figura 3.17: Risultati per la baseline del motore di ricerca; nessuna elaborazione da parte del service layer che non arricchisce la query sulla base della conoscenza del dominio

che la ricerca di termini concettualmente legati alla parola *skipper*, che appartiene al dominio, restituendo risultati maggiormente coerenti alla richiesta; ad esempio, il comportamento appena descritto è visibile in Fig. 3.18 dove la query, in quest'ultimo caso, viene trasformata in quella seguente:
(skipper^10 OR Mobilità OR Lavoro OR Professione OR Nautica OR professionista OR barca) AND (informazioni OR informazione) AND (assicurazioni OR assicurazione).

I token sono posti in OR con i loro lemmi, i concetti dei dizionari con gli elementi di livello superiore del modello tassonomico; token e concetti espansi tra di loro, invece, sono posti in AND.

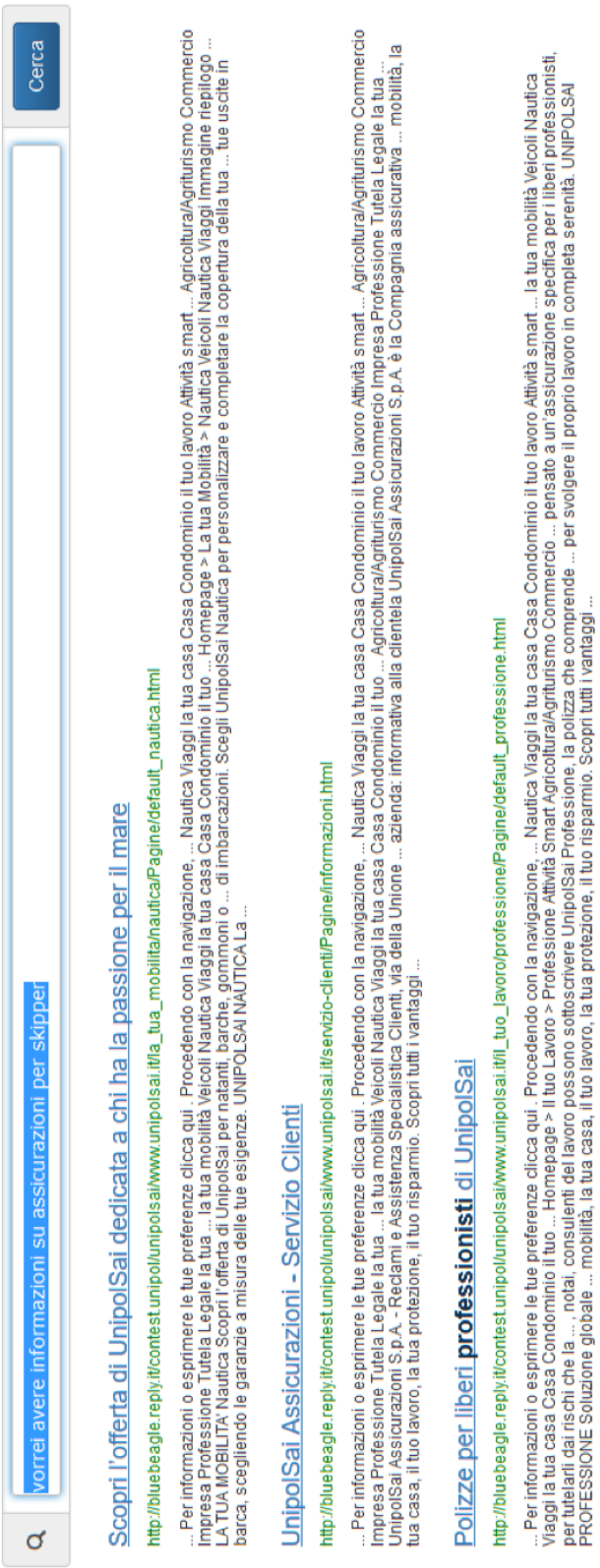


Figura 3.18: Risultati del motore di ricerca a seguito di una query "semantica" arricchita dal service layer implementato.

Capitolo 4

Sperimentazione

La sperimentazione si è incentrata sulla valutazione del motore di ricerca e sul confronto tra il suo comportamento di base e quello a seguito dell'elaborazione interposta tra l'utente e Watson. Si è tralasciata la valutazione dell'agente conversazionale che, ad oggi, si basa esclusivamente su un servizio cloud, IBM Watson Conversation, e su cui non si ha possibilità di contribuirvi attivamente se non con la scelta di un dataset adeguato. Inoltre, la distinzione dell' intent di ricerca da quelli relativi al riconoscimneto di FAQ è subordinato alle parole chiave viste al Cap.3. Le frasi raccolte per la valutazione del motore di ricerca sono esattamente 365 e questa si è focalizzata sulle capacità del motore di riprodurre la conoscenza del dominio, tipica di un essere umano. A tale scopo è stata definita una misura ad-hoc sulla base della conoscenza stessa.

4.1 Set di Frasi

Le frasi che sono state utilizzate per la realizzazione e per la valutazione del sistema si suddividono in 2 classi:

1. Della **Classe 1** fanno parte le frasi raccolte in parte mediante un campione di potenziali utenti, identificati in sede di tirocinio, in parte fornite dall'ufficio marketing dell'azienda assicuratrice e utilizzate per addestrare e testare il modello del Conversation per il riconoscimento di FAQ. Queste frasi riconducono a concetti del dominio per cui non si hanno informazioni ben precise nel corpus del motore di ricerca e per cui sono state predisposte delle Frequently Asked Question.

2. Della **Classe 2** fa parte l'insieme delle query più digitate dagli utenti sul portale della compagnia assicurativa nel periodo corrispondente al

primo trimestre fiscale del 2017 (Q1) e per cui si hanno dei risultati attesi all'interno del corpus.

Le classi che il sistema riesce a coprire sono tali per la loro semplicità d'interpretazione e la loro aderenza al modello di elaborazione scelto.

4.2 Valutazione del Search Engine

Precisione e Recupero, (in inglese precision e recall) sono due comuni metriche, viste anche per la valutazione dei POS-tagger nel Cap.2, utilizzate in larga misura nella valutazione dei sistemi di information retrieval. La Precision può essere vista come una misura di fedeltà mentre la Recall è una misura di completezza; nell'IR sono definite in termini di insieme di documenti restituiti dal motore rispetto ad una query e insieme di documenti attinenti all'argomento cercato.

$$Precisione = \frac{|\{\text{documenti attinenti}\} \cap \{\text{documenti recuperati}\}|}{|\{\text{documenti recuperati}\}|} \tag{4.1}$$

$$Recupero = \frac{|\{\text{documenti attinenti}\} \cap \{\text{documenti recuperati}\}|}{|\{\text{documenti attinenti}\}|} \tag{4.2}$$

L'utilizzo di Precision e Recall per la valutazione dei motori di IR introduce però alcuni aspetti problematici:

1. I documenti di una collezione devono essere valutati manualmente da persone esperte e non sempre la valutazione è del tutto corretta.

2. La valutazione è binaria (rilevante / non rilevante) e non sempre è immediato e facile catalogare così nettamente un documento.

3. Le misure sono pesantemente influenzate dal dominio, cioè dalla collezione e dalle query infatti un motore potrebbe avere delle ottime prestazioni in un determinato dominio ma non in un altro.

Data l'influenza di queste problematiche si è scelto quindi di definire una metrica realizzata ad-hoc per cui l'insieme delle frasi è risultato sufficiente per valutare adeguatamente il meccanismo di interpretazione del sistema. Per la valutazione del motore si utilizzano solo le frasi appartenenti alla classe 2; si tiene conto della capacità del sistema di interpretare correttamente le frasi di input e dell'aderenza del risultato ottenuto dal motore di ricerca

alle aspettative dell'utente. In particolare viene confrontato il comportamento in assenza e poi in presenza della conoscenza del dominio. In questa fase non è stato testato numericamente il modo in cui il Conversation interpreta le query utente la cui valutazione prevederebbe test più complessi poichè non andrebbe valutata solo la bontà con cui il sistema distingue tra le due classi ma dovrebbe tenere conto nella valutazione complessiva anche dei risultati ottenuti dalla valutazione del motore di ricerca. La valutazione si è focalizzata sulle capacità del motore di riprodurre la conoscenza del dominio tipica di un essere umano; si controlla infatti che il risultato atteso da un essere umano sia effettivamente presente tra i risultati restituiti da Watson. A tale scopo sono stati definiti 3 indici, di seguito indicati con **P3**, **P5** e **P10**, basati sulla conoscenza del dominio e per il calcolo dei quali gli elementi valutati sono:

1. Una Domanda o *query*.
2. La lista delle risposte attese a seguito della query digitata dall'utente sotto forma di URL che ci si aspetta venga restituito
3. La posizione di ogni risposta attesa all'interno della pagina dei risultati, o SERP.

L'indice *P3* è definito come la percentuale di risposte attese apparse tra le prime 3 posizioni della pagina dei risultati; *P5* è definito come la percentuale di risposte attese apparse tra le prime 5 posizioni mentre *P10* come la percentuale di risposte che ci si aspetta di ottenere e che figurano in una qualsiasi tra le posizioni della prima pagina dei risultati.
Il valore rappresentato da #RA è il numero totale di risultati attesi a fronte dell'insieme delle domande D. Si ha quindi:

$$\#RA = \sum_{i=1}^{\#D} \#RA_i \tag{4.3}$$

dove #D è il numero totale di domande a disposizione, in questo caso pari a **365**.
Con $\#RA3_{HIT}$ si indica, invece, il numero di volte che un risultato atteso figura tra le prime 3 posizioni della prima pagina dei risultati. P3 è quindi calcolato come il rapporto:

$$P3 = \frac{\#RA3_{hit}}{\#RA} \tag{4.4}$$

In maniera del tutto analoga sono calcolati P5 e P10 una volta noti $\#RA5_{HIT}$ e $\#RA10_{HIT}$:

$$P5 = \frac{\#RA5_{hit}}{\#RA} \quad P10 = \frac{\#RA10_{hit}}{\#RA}$$

I test sono stati effettuati costruendo uno script PHP ospitato sullo stesso server su cui è stata memorizzata la replica del portale utilizzata per la progettazione e implementazione del sistema e che se richiamato da browser preleva una per volta le domande contenute in un file di testo ed esegue la query richiamando il service layer e a sua volta il WCA. Ottenuta la risposta è confrontato ogni URL della risposta attesa per ogni domanda con quelli presenti nella pagina dei risultati e se ne valuta la posizione. La posizione è considerata valida solo se appartenente alla prima pagina dei risultati; non sono considerate valide posizioni superiori alla decima.

4.3 Risultati

I risultati ottenuti per ogni singola frase di richiesta, fornita al sistema per la valutazione, indicano se il risultato atteso è stato restituito a fronte della ricerca; in caso positivo è indicata a video la posizione, nel caso contrario un messaggio lo notifica. In Figura 4.2 si ha un estratto di ciò che viene visualizzato sul browser durante l'esecuzione dello script.

I valori ottenuti con l'esecuzione dei test sul corpus del motore che, si ricorda, conta esattamente 6243 documenti tra pagine WEB e Fascicoli informativi in formato PDF, sono riportati nelle tabelle 4.1 e 4.2. Nel dettaglio, la prima tabella (4.1) racchiude i risultati ottenuti testando la versione detta di Baseline ovvero la versione pura del search engine che si avvale degli annotatori di default forniti da WCA, senza alcuna conoscenza del dominio e per cui le richieste non sono preventivamente elaborate ma inoltrate direttamente al servizio di IBM.

Posizione	#Hit RA	#Domande	#RA
3	267	365	748
5	293		
10	330		

Tabella 4.1: Risultati del test eseguito sulla Baseline del motore di ricerca.

Il numero di risultati attesi che sono stati restituiti tra le prime 3 posizioni è pari a 267, tra le prime 5 è pari a 293 mentre il numero di risultati attesi presenti tra la posizione 1 e la 10 della pagina dei risultati è pari a 330. La seconda tabella, invece, racchiude i risultati ottenuti passando per

Domanda: attestato di rischio
Risposta aspettata: http://bluebeagle.reply.it/contest.unipol/unipolsai/www.unipolsai.it/Pagine/attestato_rischio.html
Trovata! Posizione: 1

Domanda: come faccio ad ottenere l'attestato di rischio
Risposta aspettata: http://bluebeagle.reply.it/contest.unipol/unipolsai/www.unipolsai.it/Pagine/attestato_rischio.html
Trovata! Posizione: 1

Domanda: constatazione amichevole
Risposta aspettata: http://bluebeagle.reply.it/contest.unipol/unipolsai/www.unipolsai.it/denuncia_sinistro/Pagine
/FAQ.html
Trovata! Posizione: 3

Domanda: come si fa la constatazione amichevole
Risposta aspettata: http://bluebeagle.reply.it/contest.unipol/unipolsai/www.unipolsai.it/denuncia_sinistro/Pagine
/FAQ.html
Trovata! Posizione: 3

Domanda: scatola nera
Risposta aspettata: http://bluebeagle.reply.it/contest.unipol/unipolsai/www.unipolsai.it/la_tua_mobilita/veicoli/Pagine
/unipolsai-km-e-servizi.html
Trovata! Posizione: 1

Domanda: come faccio ad installare la scatola nera
Risposta aspettata: http://bluebeagle.reply.it/contest.unipol/unipolsai/www.unipolsai.it/la_tua_mobilita/veicoli/Pagine
/unipolsai-km-e-servizi.html
Trovata! Posizione: 1

Figura 4.1: Estratto dell'output generato dall'esecuzione dello script PHP di test realizzato per la valutazione del motore di ricerca.

l'archiettura realizzata ma l'interpretazione rimane la stessa; sono considerati quindi l'espansione della query e la presenza di annotatori *custom* per la comprensione del dominio da parte del motore. I dati estratti da

Posizione	#Hit RA	#Domande	#RA
3	477	365	748
5	528		
10	595		

Tabella 4.2: Risultati del test eseguito sulla Baseline del motore di ricerca.

entrambe le tabelle permettono di calcolare P3,P5 e P10 sia per la *baseline*(BL) che per la versione *taxonomy* (TX) del motore e che risultano valere rispettivamente:

$$P3_{BL} = 0.35 \quad P5_{BL} = 0.39 \quad P10_{BL} = 0.44$$

$$P3_{TX} = 0.62 \quad P5_{TX} = 0.69 \quad P10_{TX} = 0.80$$

I risultati fin qui descritti sono schematizzati nel grafico in Figura 4.2. E'

possibile notare che per la versione Taxonomy, basato su un modello tassonomico di risorse per l'analisi del testo (allo scopo di realizzare una base di conoscenza per il dominio scelto), si ha un notevole incremento dei risultati dichiarati coerenti e restituiti tra le prime posizioni della SERP.

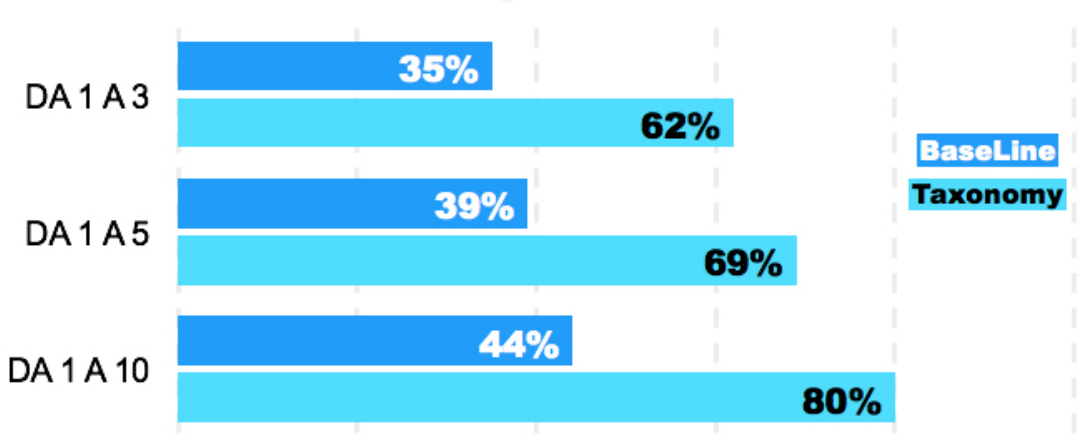


Figura 4.2: Comparazione dei risultati dei test realizzati sia sulla versione Baseline che Taxonomy del motore di ricerca costruito.

Capitolo 5

Conclusioni e Sviluppi futuri

5.1 Conclusioni

La costruzione del motore di ricerca richiesto dal cliente in grado di rispondere a domande in linguaggio naturale è stata quindi possibile grazie all'integrazione di servizi di natura diversa (on premisis, cloud e implementati per lo scopo in sede di tirocinio). L'architettura che è stata proposta per la realizzazione del servizio sfrutta una base di conoscenza per catturare le intenzioni espresse da un utente. Ogni modulo dell'architettura utilizza come linguaggio di comunicazione e di rappresentazione dell'informazione il formato JSON, molto utilizzato nell'ambito dei servizi web (RESTful, SOAP, ecc.) come i servizi, tutti REST, realizzati e interrogati nell'implementazione di questo progetto. Il dominio del cliente è stato abbastanza generale da permettere la strutturazione di un'ontologia di dominio facilmente trasportabile su altri domini nell'ambito assicurativo. La modularità dell'architettura permette un'elevata manutenibilità e la separazione delle funzionalità alla base del progetto.

- **Node-Red**: ha permesso di realizzare un'architettura complessa per l'interconnessione dei diversi servizi e lo fa mostrando allo sviluppatore un'interfaccia a flusso in alternativa a migliaia e migliaia di righe di codice. Integrare nell'architettura ulteriori servizi di elaborazione della query sarebbe semplice e immediato aggiungendo nodi preposti a questo nel flusso.

- **Conversation**: essendo un servizio Cloud in grado di adattare il modello ML su cui si basa la costruzione della conversazione dell'agente conversazionale è possibile addestrare il modello stesso a riconoscere nuovi *intent*, gestire nuove richieste o modificare il dominio di lavoro.

- **Watson Content Analytics**: è il core del search engine e permette di raccogliere i documenti del corpus, indicizzarli e annotarli in modo da arricchirli con informazioni provenienti dalla base di conoscenza.
- **WCA Studio**: è il modulo che permette di costruire la base di conoscenza mediante risorse personalizzabili quali dizionari e regole di matching.
- **Service Layer**: il modulo responsabile dell'integrazione tra i servizi di gestione della query utente e il motore di ricerca. Realizza un meccanismo di traduzione da elementi semantici a query al search-engine. Attualmente questo modulo non è legato a particolari domini ma si affida ai risultati dell'analisi restituita da Watson; è perciò estendibile a qualsiasi tipo di dominio.

Si è dimostrato che grazie all'elaborazione della query e dei contenuti testuali dei documenti mediante tecniche di elaborazione del linguaggio naturale è possibile incrementare sensibilmente la rilevanza dei risulati del motore di ricerca. La rilevanza dei documenti non è calcolata limitandosi alla corrispondenza tra le parole della query digitate dall'utente ma valutando anche i concetti appartenenti al dominio legati alle stesse.

5.2 Sviluppi Futuri

Gli sviluppi futuri potrebbero riguardare l'adattamento del progetto a domini ed ambiti differenti da quello finanziario ma appartenenti al mondo enterprise quali quello Bancario, delle Telecomunicazioni e così via. Oltre all'adattamento si potrebbe invece pensare a nuove funzionalità da offrire al cliente.

5.2.1 Adattamento ad altri Domini Enterprise

Come accennato in precedenza, la modularità con cui si è scelto di sviluppare il progetto, permette un adattamento a fronte di un effort non troppo elevato lavorando sui singoli nodi dell'architettura e sulle risorse del WCA Studio che andrebbero sostituite con risorse adatte al dominio target.

- **WCA Studio**: Al fine di adattare le risorse a un nuovo dominio è necessario realizzare dizionari e parsing rule allo scopo di riconoscere gli elementi dello stesso e annotarne i relativi documenti.

- **Node-Red**: Node-Red realizza l'interconnessione tra i vari moduli dell'architettura. Andrebbe adattato solo nel caso in cui si volessero interporre nuovi livelli di servizio tra l'interfaccia del chat-bot e Watson.

- **Conversation**: Il funzionamento rimane intatto. Si adatta il grafo del discorso affinchè si risponda adeguatamente al nuovo amibito (adattando ad esempio le FAQ al dominio corretto) o aggiungendo nuovi intenti da riconoscere se previsti dal dominio.

- **Watson Content Analytics**: Andrebbe adattato scegliendo il crawler adatto a creare il corpus della collezione per il nuovo dominio e come motore di analisi il motore creato, a partire dalle nuove risorse, con WCA Studio.

- **Service Layer**: Il service Layer rimane praticamente intatto; cambierebbe la gestione dell'analisi restituita da WCA qualora si scegliesse un modello differente da quello tassonomico realizzato per il dominio assicurativo del cliente.

5.2.2 Nuove Funzionalità

- **Enanchment mediante feedback utente**: La risposta inline fornita dall'utente mediante bot Telegram a seguito dei risultati restituiti da WCA potrebbero essere raccolti ed elaborati per raffinare l'accuratezza del motore di ricerca. Tale funzionalità, oltre alla raccolta dei feedback, richiederebbe la progettazione e l'implementazione di un meccanismo in grado di alterare l'algoritmo per il calcolo della rilevanza dei documenti nella costituzione del ranking.

- **Store Locator**: Mediante la costruzione di un dizionario **Città** e un dizionario **POI** (Point-of-Interest) è possibile riconoscere all'interno di una query le entità che identificano una particolare Città e il tipo di negozio di cui l'utente esprime il bisogno mediante la sua richiesta. Una volta in possesso di queste informazioni a front-end è possibile restituire il locator centrato sulla Città selezionata con, in evidenza, i negozi che soddisfano la richiesta stessa.

- **Preventivatore**:Si creano delle Parsing Rule che identificano le entità necessarie a riconoscere la volontà dell'utente di richiedere un

preventivo auto. Una Character Rule permette di riconoscere il formato di una targa e riempire il campo targa del preventivo; allo stesso modo si riconosce la data di nascita associata all'utente che richiede il preventivo. Viene automaticamente restituito un preventivo a seguito di una semplice query in linguaggio naturale del tipo "***vorrei un preventivo per l'auto targata EB512MN intestata a Marco Montebello nato il 26/05/1988***".

- **Suggerimento Prodotti**:In modo molto simile alle soluzioni appena presentate si potrebbe creare un dizionario le cui entry associano a particolari parole, che verranno riconosciute all'interno della query utente, il codice associato ad prodotto inerente al concetto digitato e che deve essere consigliato.

Bibliografia

[1] DB-Engines. (2017) Search engine ranking. [Online]. Available: https://db-engines.com/en/ranking/search+engine

[2] B. Liu, *Web data mining: exploring hyperlinks, contents, and usage data*. Springer Science & Business Media, 2007.

[3] M. Melucci, *Information Retrieval. Metodi e modelli per i motori di ricerca: Metodi e modelli per i motori di ricerca*. FrancoAngeli, 2013.

[4] L. Pontieri. (2010, Sep.) Modelli di ir. [Online]. Available: http://staff.icar.cnr.it/pontieri/didattica/SIIR/lezioni/L4%20-%20Modelli%20di%20IR.pdf

[5] A. Singhal, C. Buckley, and M. Mitra, "Pivoted document length normalization," in *ACM SIGIR Forum*, vol. 51, no. 2. ACM, 2017, pp. 176–184.

[6] A. Mussino, *Non tutto, ma di tutto sulla statistica sociale: Appunti per l'analisi di dati di area sociale*, ser. 3S. Nuova Cultura, 2015. [Online]. Available: https://books.google.it/books?id=9BjmCQAAQBAJ

[7] D. Montesi. Information retrieval. [Online]. Available: http://www.cs.unibo.it/~montesi/CBD/06IR.pdf

[8] I. dell'Enciclopedia Italiana. (2011) Treccani.it - vocabolario treccani on line. [Online]. Available: http://www.treccani.it

[9] M. Maggini. Tecnologie per l'elaborazione del linguaggio. [Online]. Available: http://www.dii.unisi.it/~maggini/Teaching/TEL/slides/05%20-%20NLP%20-%20Parole.pdf

[10] The-Apache-Software-Foundation. (2017) Opennlp language detector evaluation. [Online]. Available: https://www.apache.org/dist/opennlp/models/langdetect/1.8.3/langdetect-183.bin.report.txt

[11] R. Kuc and M. Rogozinski, “Elasticsearch server, vol. 1,” 2013.

[12] Node-RED. (2017 (accessed January 7, 2018)) Node-red: Documentation. [Online]. Available: https://nodered.org/docs/"

[13] Telegram. (2017) Bots: An introduction for developers. [Online]. Available: https://core.telegram.org/bots

[14] IBM. (2017, Dec.) Ibm bluemix conversation. [Online]. Available: https://console.bluemix.net/docs/services/conversation/index.html#about

[15] W.-D. J. Zhu, B. Foyle, D. Gagné, V. Gupta, J. Magdalen, A. S. Mundi, T. Nasukawa, M. Paulis, J. Singer, M. Triska *et al.*, *IBM Watson Content Analytics: Discovering Actionable Insight from Your Content*. IBM Redbooks, 2014.

Elenco delle figure

2.1 Architettura generica di un sistema IR 9
2.2 Flow chart di un crawler sequenziale di base [2] 12
2.3 Costruzione del vocabolario di termini a partire dal contenuto dei documenti in un sistema di IR[7] 15
2.4 Generazione degli indici in un sistema di IR, ad ogni elemento del vocabolario vi è associata una lista invertita [7] 16
2.5 Interrogazione di un sistema IR con Inverted Index [7] 17
2.6 Esempio di albero sintattico a seguito dell'operazione di parsing su una frase [9] 23
2.7 esempio di stati di un chart [9] 24
2.8 Predittore per uno stato del chart riferito all'esempio di fig. 2.7 [9] 25
2.9 Scanner per uno stato del chart riferito all'esempio di fig. 2.7 [9] 25
2.10 Completer per uno stato del chart riferito all'esempio di fig. 2.7 [9] 25

3.1 Architettura del sistema realizzato 32
3.2 Flusso Node-Red principale per la realizzazione dell'agente conversazionale previsto dall'architettura del progetto. 34
3.3 Composizione del flusso Node-Red secondario utilizzato dal flow in figura 3.2 per la restituzione dei risultati all'utente e per la gestione di una sua eventuale risposta 37
3.4 Esperienza utente offerta dal Bot su architettura Node-Red . . 39
3.5 Schermata di gestione degli intenti in IBM Bluemix Conversation 40
3.6 Albero di Dialogo e i costrutti if/then dei suoi nodi 41
3.7 Schermata di deployment di un workspace in IBM Bluemix Conversation 41
3.8 Restituzione della risposta ad una FAQ mediante il bot Telegram 44

3.9 Esempio di struttura logica di un documento convertito in formato CAS [15] 46
3.10 Aree e Sottoaree tematiche del portale oggetto di studio . . . 47
3.11 Modello tassonomico utilizzato per l'analisi sintattica del testo custom realizzato mediante WCA Studio 48
3.12 Parte del dizionario Elemento 49
3.13 Architettura di Watson Explorer Content Analytics [15] 49
3.14 Data Model utilizzato da WCA per le sue collezioni [15] 50
3.15 Esempio di struttura logica di un documento convertito in formato CAS [15] 51
3.16 Administration Console di IBM Watson Content Analytics . . . 54
3.17 Risultati per la baseline del motore di ricerca; nessuna elaborazione da parte del service layer che non arricchisce la query sulla base della conoscenza del dominio 68
3.18 Risultati del motore di ricerca a seguito di una query "semantica" arricchita dal service layer implementato. 70

4.1 Estratto dell'output generato dall'esecuzione dello script PHP di test realizzato per la valutazione del motore di ricerca. . . . 75
4.2 Comparazione dei risultati dei test realizzati sia sulla versione Baseline che Taxonomy del motore di ricerca costruito. 76

Elenco delle tabelle

2.1 Un subset del Penn Treebank tagset che prevede più classi dalla grammatica tradizionale. 20

3.1 Struttura di suporto al service layer per la costruzione della query semantica da inviare al motore di ricerca WCA 62

4.1 Risultati del test eseguito sulla Baseline del motore di ricerca. 74

4.2 Risultati del test eseguito sulla Baseline del motore di ricerca. 75

Ringraziamenti

Ringrazio Blue Reply S.r.l per avermi ospitato e permesso di realizzare questo lavoro. In particolare ringrazio Oscar, Gianpiero e tutto il team Cognitive per aver creduto in me e avere condiviso con me la loro conoscenza e avermi fatto crescere professionalmente. Grazie alla mia famiglia, a Giovanna, Graziano e Fabio senza i quali il mio percorso in questi due anni al Politecnico non sarebbe stato lo stesso.