Alma Mater Studiorum · Università di Bologna

SCUOLA DI SCIENZE
Corso di Laurea Magistrale in Informatica

ESTENDERE ED INTEGRARE TECNICHE DI INFORMATION RETRIEVAL E VISUALIZZAZIONE SUNBURST PER LA RICERCA DI FRAMMENTI E DOCUMENTI RILEVANTI

Relatore:
Prof.
ANGELO DI IORIO

Correlatore:
Prof.
FRANCESCO POGGI

Presentata da:
STEFANO NOTARI

Sessione I
Anno Accademico 2023/2024

Our business in this world is not to succeed,
but to continue to fail in good spirits.

Robert Louis Balfour Stevenson

Introduzione

Il problema di come permettere all'utente di cercare un sottoinsieme di oggetti (es. libri, articoli o pagine web) utili per soddisfare il bisogno di conoscenza per la sua ricerca affonda le radici agli inizi della civiltà. Infatti, nelle biblioteche esiste l'ordinamento per nome dell'autore o il raggruppamento di oggetti per la categoria di appartenenza. L'idea di utilizzare i computer per la ricerca di informazioni risale ai primi anni del dopoguerra, quando Vannevar Bush [11] fu ispirato dal primo brevetto di successo ottenuto da Emanuel Goldberg progettato per la ricerca di documenti memorizzati su microfilm (conosciuto anche come "*Statistical Machine*") [49]. Per l'implementazione bisogna aspettare il 1948 quando J. E. Holmstrom [14], citando una prima versione del computer UNIVAC, descrisse una ricerca di informazioni su un computer. L'espandersi della conoscenza e del volume delle opere scritte ha reso praticamente impossibile visualizzare tutti i risultati di ricerca ed ha imposto l'introduzione del concetto di rilevanza per poterne selezionare un sottoinsieme utile da restituire all'utente. Questo ha introdotto un duplice problema: come presentare i risultati all'utente e come sviluppare un algoritmo per il calcolo della rilevanza per la selezione dei k-documenti maggiormente rilevanti per la sua ricerca. Per quanto riguarda le interfacce, la soluzione maggiormente adottata è la visualizzazione dei risultati come semplice lista, fenomeno particolarmente evidente, non solo, nei motori di ricerca ma anche nelle interfacce utilizzate dalle biblioteche digitali. Inoltre, le soluzioni presenti non tengono in considerazione casi d'uso in cui l'utente non sia interessato all'intero documento ma piuttosto ad una sua parte (frammento). Infatti, allo stato attuale le interfacce delle biblioteche digitali, per ogni documento restituito, selezionano un estratto che ritengono sia più rilevante per la ricerca d'utente ma non offrono la possibilità di visualizzare/navigare i frammenti in maniera globale. La mancanza di alternative ha ispirato il lavoro di questa tesi, ovvero proporre un'interfaccia per la visualizzazione dei risultati di ricerca, mantenendo una panoramica globale sugli stessi. Per raggiungere l'obiettivo, come scheletro di partenza è stata utilizzata l'applicazione DocuDipity [44], il cui sviluppo iniziale riguardava il confronto e l'analisi di articoli scientifici, mettendo a disposizione dell'utente una serie di visualizzazioni alternative alla classica vista ipertestuale. Nella versione messa a disposizione, tuttavia, non era stato considerato il task di ricerca per cui, oltre all'implementazione dell'interfaccia proposta, ha richiesto lo sviluppo di un motore di ricerca per il task di Information Retrieval. Per

lo sviluppo del motore di ricerca è stato realizzato un algoritmo ibrido, il quale combina la ricerca lessicale con quella semantica (resa possibile grazie al recente sviluppo di numerosi Large Language Model), migliorando la qualità dei risultati. Invece, l'interfaccia proposta per la visualizzazione dei risultati si basa sulla vista SunBurst, la quale organizza i risultati non in maniera sequenziale ma radiale mantenendo una panoramica globale. L'adattamento dell'applicazione è stato svolto in più fasi: una di progettazione della soluzione, una di refactoring/implementazione della stessa ed, infine, una di valutazione del raggiungimento degli obiettivi. La fase di progettazione ha riguardato lo studio della documentazione dell'applicazione, con particolare attenzione all'architettura e alle tecnologie utilizzate. Prima dello sviluppo, è stato anche necessario approfondire lo stato dell'arte dell'Information Retrieval per studiare i principali algoritmi e tecniche che meglio si adattavano al nostro task. Successivamente, la fase di valutazione è stata suddivisa in due test: uno per l'interfaccia e uno per il motore di ricerca ibrido implementato.
L'elaborato è strutturato nel seguente modo:

- nel primo capitolo verrà presentato lo stato dell'arte relativo alle interfacce per la visualizzazione dei risultati di ricerca ed agli algoritmi di Information Retrieval. Particolare attenzione verrà posta sulla presenza in letteratura di articoli che propongono il SunBurst come tecnica di visualizzazione dei risultati di ricerca

- nel secondo capitolo verrà introdotta l'applicazione DocuDipity e le visualizzazioni alternative proposte. Inoltre, si fornirà un esempio d'uso per esplorare le sue potenzialità

- nel terzo capitolo verrà discussa la soluzione proposta, presentando l'interfaccia alternativa per la visualizzazione dei risultati e l'introduzione del concetto di rilevanza, necessario al fine di fornire un ordinamento dei documenti/articoli

- nel quarto capitolo si approfondirà l'implementazione, con particolare attenzione al motore di ricerca ibrido e alla realizzazione dell'interfaccia proposta

- nel quinto capitolo verranno presentati i risultati dei test effettuati per la valutazione dell'interfaccia e del motore di ricerca ibrido

- nell'ultimo capitolo verranno esposte le conclusioni con i possibili sviluppi futuri

Indice

Capitolo 1

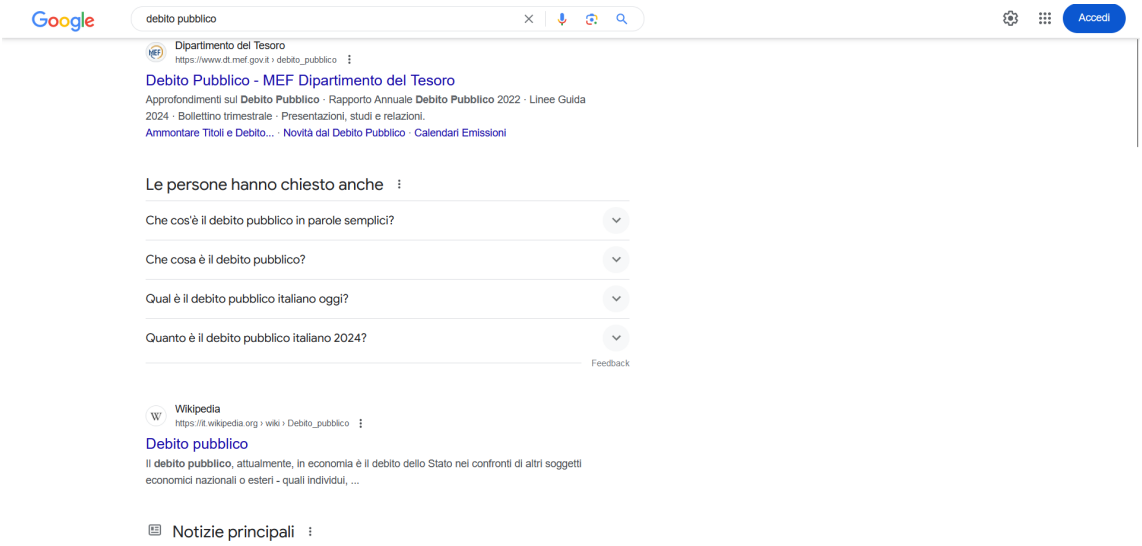
Stato dell'arte

Nella prima parte di questo capitolo approfondiremo gli studi che sono stati effettuati sull'utilizzo, di possibili, interfacce alternative per la visualizzazione dei risultati di ricerca. Allo stato attuale, la tecnica più utilizzata dai maggiori motori di ricerca è quella di presentare i risultati come una lista piatta. In particolare, approfondiremo l'utilizzo del SunBurst in letteratura per verificare se sia già stato applicato alla visualizzazione della struttura di un documento o dei risultati di ricerca.
Successivamente, approfondiremo lo stato dell'arte dell'Information Retrieval per introdurre le principali tecniche al fine di assegnare uno score ai documenti rilevanti per la query d'utente. Il recente progresso di numerosi Large Language Model ha permesso lo sviluppo di un nuovo ramo dell'Information Retrieval non più incentrato sul "*exact match*"[1] ma sulla similarità delle frasi, ovvero il cosidetto Semantic Search.
Quindi, inizialmente verrà esposto lo stato dell'arte riguardante la ricerca lessicale (o Sparse Vector Representation) e successivamente come questo sia stato influenzato dagli ultimi sviluppi della Semantic Search (o Dense Vector Representation).

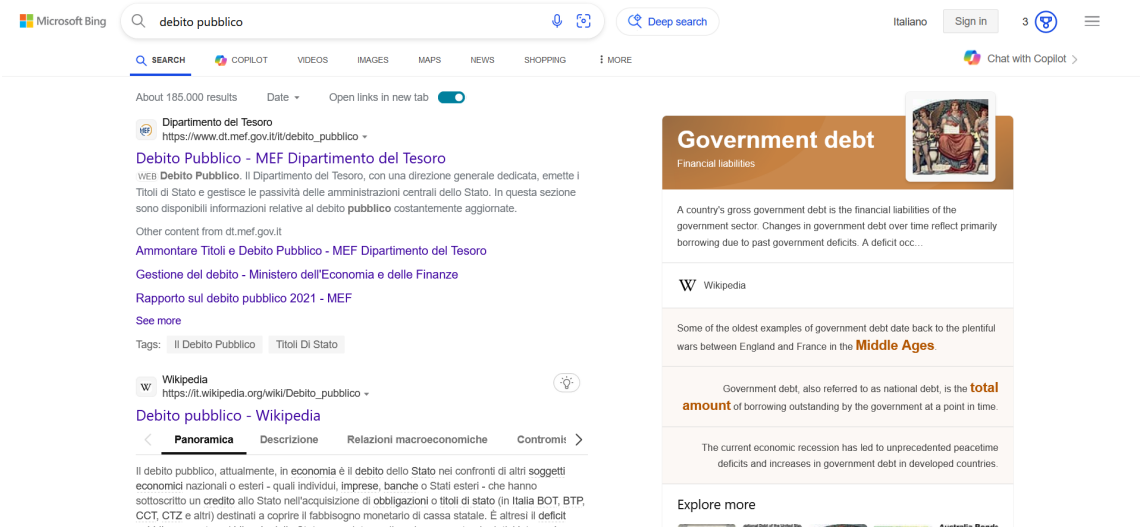
1.1 Visualizzazioni alternative dei risultati di ricerca

Oggi, i principali motori di ricerca utilizzano una SERP (Search Engine Results Page), ovvero la pagina dove sono mostrati i risultati di ricerca all'utente, estremamente simili tra di loro. Questo può essere verificato effettuando una ricerca: in Figura 1.1 sono riportati i risultati di due distinti motori di ricerca commerciali ma la pagina SERP visualizzata è visibilmente molto simile (come i risultati prodotti). Inoltre, data la familiarità che gli utenti hanno con questa modalità di ricerca e visualizzazione dei risultati ha, praticamente, disincentivato una qualunque novità per paura di perdere utenti.

[1] tecnica che prevede l'utilizzo di una parola chiave specifica, esattamente com'è stata definita, all'interno di una pagina web



(a) Risultati di ricerca mostrati da Google per la query “debito pubblico”



(b) Risultati di ricerca mostrati da Bing per la query “debito pubblico”

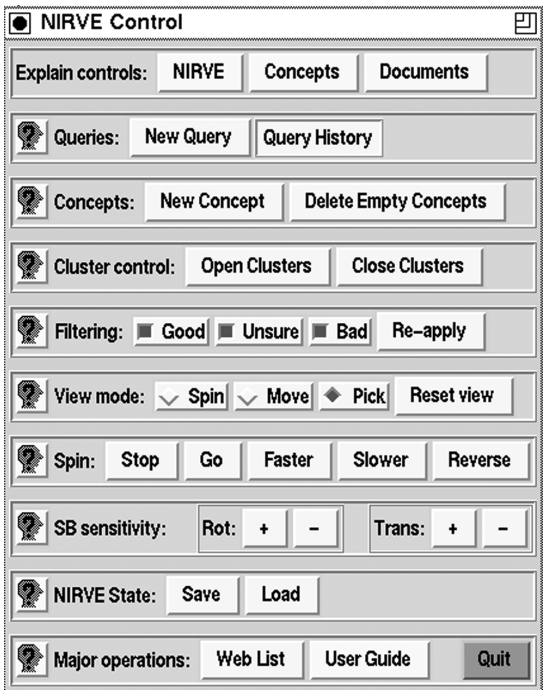
Figura 1.1: Confronto delle SERP tra due motori di ricerca

Ciò non può essere detto per il materiale presente in letteratura in quanto, senza il vincolo della logica di business, ha permesso ai ricercatori un maggiore grado di libertà nell'esplorare nuove soluzioni.
Un primo esempio può essere NIRVE [51], un tool per la visualizzazione dei risultati di ricerca che tenta di combinare la visualizzazione 2D con la 3D (tridimensionale). Infatti, il tool era composto da due componenti principali: una finestra per la rappresentazione dei documenti in 3D ed una finestra per il pannello di controllo.

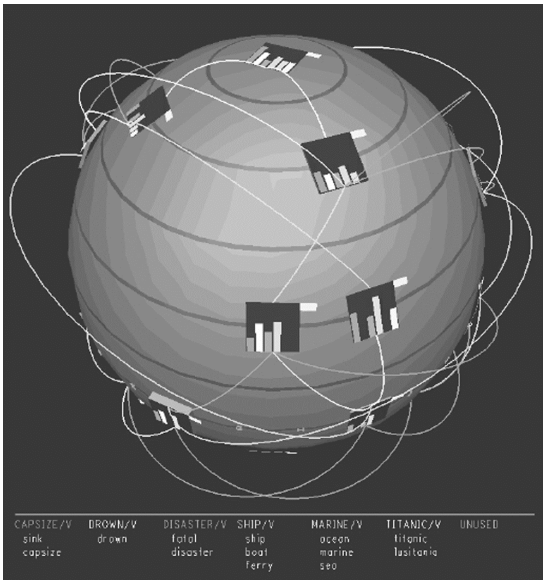
La rappresentazione 3D dei documenti avviene su un globo, Figura 1.2b, in cui sono collocati dei cluster, ovvero un insieme di documenti che hanno un sottoinsieme comune di concetti. La loro posizione sul globo viene così determinata:

- Latitudine: dipende dal numero di concetti (concept) contenuti dal cluster, maggiore il numero di concetti minore sarà la distanza dal "polo nord" del globo
- Longitudine: nessun significato intrinseco, si tenta di avvicinare i cluster con "concepts" simili

A ogni concetto viene assegnato un colore univoco, quando due cluster differiscono per un solo "concept" vengono collegati da un arco con il colore del "concept" di differenza. La finestra con il pannello di controllo, Figura 1.2a, contiene i comandi con i quali è possibile personalizzare alcuni parametri della rappresentazione 3D.
I risultati ottenuti suggeriscono che, per sfruttare appieno i vantaggi dell'interfaccia proposta, è necessario avere una corretta mappatura tra utenti, task ed interfaccia. Inoltre, è emerso che con la vista tridimensionale i tempi di risposta (per l'esecuzione di un task) diminuiscono sensibilmente all'aumentare delle sessioni svolte dall'utente.



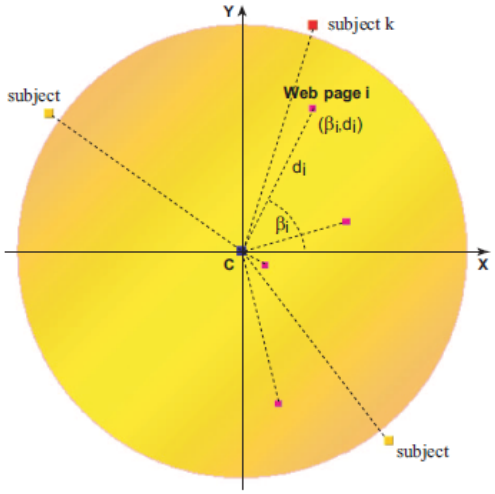
(a) Pannello di controllo di NIRVE



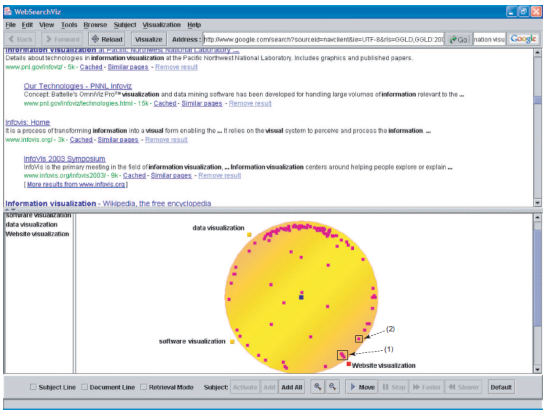
(b) Globo in cui vengono mostrati i risultati di ricerca in 3D

Figura 1.2: Overview dell'applicazione NIRVE per la visualizzazione alternativa dei risultati di ricerca, le immagini sono disponibili nel paper originale (`https://doi.org/10.1145/312624.312634`)

Oltre a proporre un metodo di visualizzazione dei risultati di ricerca, si è anche tentato di fornire una metafora per aiutare l'utente a comprendere in maniera intuitiva i risultati di ricerca [42]. Nello specifico, la metafora adottata è quella del sistema solare: ogni pianeta ed asteroide ha la propria orbita ed una costante velocità di movimento, ogni pianeta o asteroide ruota attorno al sole ed ognuno di essi è attratto reciprocamente dalla forza di gravità. Nel modello proposto, il sole (o punto centrale) è la query dell'utente mentre i concept (anche indicati con il termine subjects) ed i risultati di ricerca sono i pianeti e gli asteroidi. Quando si sposta l'icona di un concept (o subject), questo influenza la rotazione delle pagine web rilevanti per quello specifico concept. Infine, la gravità viene definita come la "*semantic search*" tra la query e i risultati di ricerca. Nella Figura 1.3b viene riportato un esempio di ricerca per mostrare le funzionalità dell'applicazione, mentre nella Figura 1.3a viene definito lo spazio di visualizzazione utilizzato per la metafora.



(a) Spazio di visualizzazione

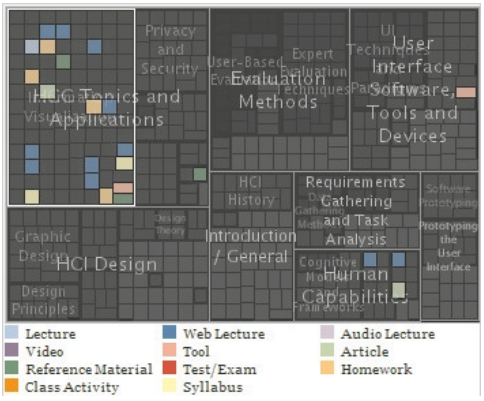


(b) Esempio di visualizzazione risultati di ricerca

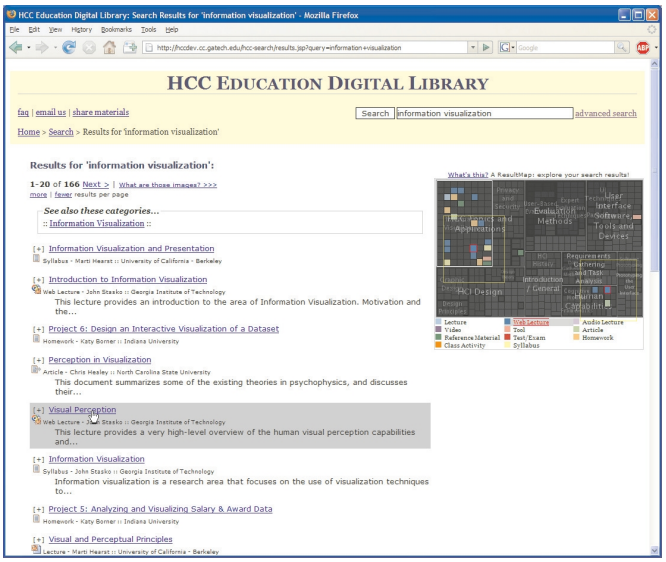
Figura 1.3: Overview di WebSearchViz, le immagini sono disponibili nel paper originale (`https://doi.org/10.1109/TVCG.2006.111`)

Un altro studio ha preso in considerazione i motori di ricerca delle biblioteche digitali [13], in quanto avendo a disposizione un catalogo omogeneo e ricco di metadati è facilmente estendibile con nuove funzionalità.
Per esempio, è stato proposto di aggiungere la feature di visualizzazione dei risultati di ricerca con una TreeMap dove i risultati vengono raggruppati in base alla categoria di appartenenza ed organizzati in un rettangolo mostrato a destra della classica SERP (Figura 1.4b). Inoltre, ogni risultato viene evidenziato con un colore a seconda della sua tipologia (video, libro, articolo, file audio, etc.) come mostrato nella Figura 1.4a.

Ovviamente, è possibile interagire con i nodi della TreeMap per navigare i risultati della ricerca visualizzati come lista.



(a) TreeMap come metodo di visualizzazione alternativo



(b) SERP con l'aggiunta della TreeMap

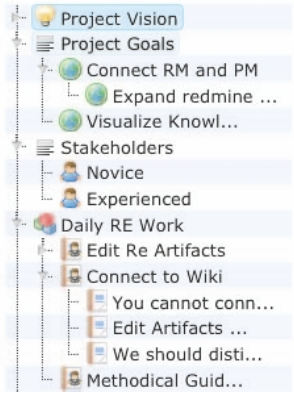
Figura 1.4: Overview dell'interfaccia di visualizzazione proposta con l'utilizzo della TreeMap, le immagini sono disponibili nel paper originale (`https://doi.org/10.1109/TVCG.2009.176`)

Per il lettore interessato, si consiglia la tesi di dottorato di Thomas M. Mann [34], in cui viene trattato in maniera estesa ed approfondita una serie di visualizzazioni alternative e delle loro applicazioni per i risultati di ricerca. Tuttavia, si sottolinea che nel lavoro non è stata trattata esplicitamente la vista SunBurst e, per tale ragione, la prossima sezione sarà dedicata esclusivamente a questa particolare vista.

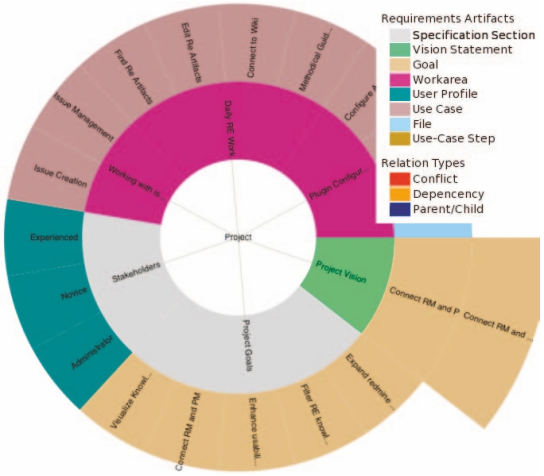
1.2 SunBurst: Letteratura

Il SunBurst è una tecnica di visualizzazione che permette di rappresentare dati gerarchici su livelli circolari (dette anche corone circolari), il cui centro rappresenta l'elemento radice della collezione dati. Nello specifico, una corona circolare può essere composta da più segmenti che rappresentano gli elementi della collezione e che hanno una relazione gerarchica con il segmento della corona cirolcare esterna.
Per questa sua caratteristica è diventato un metodo di visualizzazione molto studiato e applicato in svariati ambiti.
In questo articolo [36] gli autori esplorano le potenzialità del SunBurst per rappresentare le dipendenze degli artefatti nell'Ingegneria del software. Invece di utilizzare una classica

vista ad albero (come mostrato nella Figura 1.5a) o diagramma uml, si rappresentano gli artefatti in maniera circolare, dove il centro rappresenta il progetto ed ogni livello della corona circolare rispetta la gerarchia della struttura dati ad albero. Uno dei principali vantaggi risiede nella maggiore leggibilità e nella vista globale. Nella Figura 1.5 è possibile vedere un confronto tra la rappresentazione degli artefatti come struttura dati ad albero e SunBurst.



(a) Artefatti rappresentati con una struttura ad albero



(b) Visualizzazione alternativa proposta

Figura 1.5: Confronto tra modi di visualizzare gli artefatti, le immagini sono disponibili nel paper originale (`http://dx.doi.org/10.1109/MARK.2011.6046557`)

Un altro utilizzo è l'analisi del comportamento dell'utente quando naviga un sito, infatti la maggior parte dei siti hanno numerosi business-goal legati al comportamento dell'utente (basti pensare al cosiddetto "*user engagement*" con il contenuto del sito). In questo articolo [48], un ricercatore di Google vuole approfondire i modi con i quali gli utenti scoprono e navigano i video di YouTube con l'obiettivo di massimizzare l'engagement degli stessi con i video. Per far ciò il ricercatore propone di applicare la vista SunBurst, dove per semplificare categorizza ogni URL in un insieme pre-definito di pagine. Tuttavia, tali dati sono confidenziali quindi per mostrare un esempio di SunBurst per tale analisi utilizza i dati di un fittizio sito di e-commerce. Tale scelta viene giustificata dal fatto che, similmente a YouTube, anche un sito di e-commerce è interessato allo "*user engagement*" per massimizzare il numero di prodotti venduti.
Nella Figura 1.6 è possibile analizzare i punti di ingresso del sito, la maggior parte degli utenti entra dalla "*home page*" del sito mentre altri entrano direttamente dalla pagina del prodotto. Salendo di livello, è possibile seguire il comportamento dell'utente e la sua iterazione con il menu o con i link presenti. Questo aiuta a visualizzare i pattern degli utenti di un sito su un unico grafico, di cui è possibile fare uno screenshot, e di

rispondere in maniera agevole ad un'importante domanda (sia per gli sviluppatori che per i manager): come interagiscono gli utenti con il nostro sito?

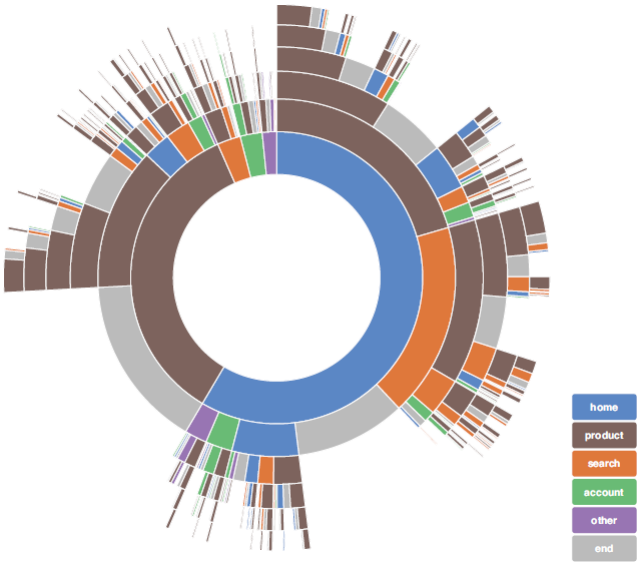


Figura 1.6: Visualizzazione SunBurst per analizzare i path degli utenti in un sito, l'immagine è disponibile nel paper originale (`https://doi.org/10.1109/MCG.2014.63`)

Infine, il SunBurst può essere utilizzato anche per i task di public opinion analysis. In questo articolo [31] viene utilizzato per analizzare l'opinione pubblica relativa ad eventi tenuti nella provincia di Shaanxi (Cina): nel primo layer compare il nome dell'evento, nel secondo layer compaiono gli aggettivi che descrivono la percezione che gli abitanti hanno per quell'evento. Nella Figura 1.7 viene mostrato il risultato ottenuto.

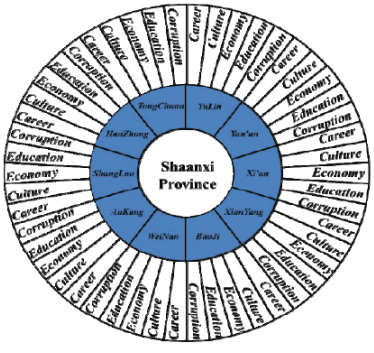


Figura 1.7: SunBurst per la visualizzazione di task relativi alla public opinion analysis, immagine disponibile nel paper originale (`https://doi.org/10.1109/BMEI.2015.7401618`)

1.3 Information Retrieval

Nella sezione precedente sono stati esposti alcuni modi per visualizzare i risultati di ricerca, tuttavia per rendere possibile ciò è necessario un algoritmo che data la query d'utente selezioni un sottoinsieme di risorse rilevanti per la sua ricerca.
Questo task è conosciuto con il nome di Information Retrieval, il quale identifica le risorse che sono rilevanti per soddisfare il cosidetto "*user information needs*". In altre parole, si tratta di un algoritmo che data una query ed un insieme di risorse (ad esempio documenti o pagine web) assegna, per ogni risorsa, un numero reale in modo tale da ordinare i documenti ed evidenziare le top-k risorse da ritornare all'utente. Per ottenere questa lista di risorse esistono due strategie principali: la ricerca lessicale (Sparse Vector Representation) e la ricerca semantica (Dense Vector Representation).
Quest'ultima si è diffusa nell'ultimo periodo grazie all'incredibile sviluppo dei Large Language Model: si consideri il modello GPT-3 [10] utilizzato per il chatbot chatGPT, oppure la versione sviluppata da Google Gemini [54] (precedentemente conosciuto come BARD, basato sul modello BERT).
In Machine Learning un modello viene definito come un tipo di "*modello matematico*" che, dopo essere "addestrato" su un dato dataset, può essere utilizzato per fare predizioni o classificazioni su nuovi dati (non visti nella fase di addestramento).
Lo scoglio principale per l'utilizzo di questi modelli è dato dall'enorme potenza di calcolo richiesta per addestrare e mantenere il modello, basti pensare che per il modello GPT3 si stima un utilizzo di circa 350GB di RAM ed un costo complessivo intorno ai 12 milioni di dollari [59].
Per superare questo scoglio, sono stati messi a disposizione della community dei modelli già addestrati, su specifici dataset, abbattendo la barriera iniziale del costo e del tempo di addestramento.
Di seguito verrà approfondita la ricerca lessicale, mostrando i passaggi necessari per costruire un motore di ricerca partendo da una generica collezione di documenti e, successivamente, si descriveranno brevemente alcune sue limitazioni e si approfondirà la ricerca semantica come risposta ai problemi citati.

1.3.1 Sparse Vector Representation (ricerca lessicale)

Il principale problema per il task di Information Retrieval è l'ambiguità del linguaggio naturale, in quanto molte parole possono assumere significati diversi in base al contesto in cui vengono usate oppure possono esserci parole che non sono rilevanti per la query. Ad esempio, le coniugazioni dei verbi non aggiungono informazioni utili per il task e, anzi, rischiano di far perdere possibili match in quanto la query potrebbe contenere un verbo coniugato in maniera discorde nell'occorrenza del documento.
Il problema di come rappresentare (ed estrarre) le informazioni contenute in una collezione di documenti è comune a numerosi task di NLP (Natural Language Processing) e, per

tale ragione, l'approccio nello stato dell'arte tende ad essere concorde e standardizzato [57].
In Figura 1.8 vengono mostrate le fasi per la normalizzazione del testo estratto (Clean Text, Lowercase), necessarie ad ottenere un insieme omogeneo di testo senza dettagli sulla sua formattazione, e la generazione dei token per rappresentare le informazioni contenute in ciascun documento (Tokenization, Lemmatization, Stop words and punctuation e Unigrams & Bigrams). L'importanza dell'ultima fase consiste nel fatto che gli algoritmi di Information Retrieval, per poter assegnare un punteggio di rilevanza a ciascun documento, hanno bisogno di tenere traccia del numero di occorrenze per ciascuna parola della query all'interno del documento, per poi utilizzare questo dato in una formula per calcolare il punteggio di rilevanza.

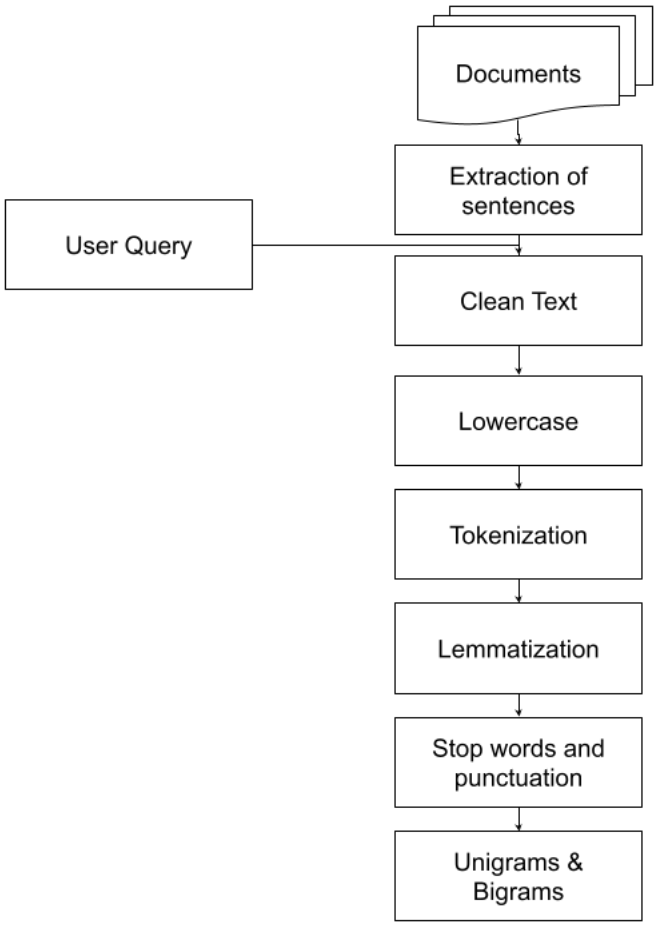


Figura 1.8: Overview delle fasi del text-preprocessing dello Sparse Vector Representation

Estrazione del testo dal documento

Il primo passo prevede l'estrazione del testo dalla base di dati o dal formato con cui il documento è stato salvato. Nel caso di "sentiment analysis"[2], non si avrà un documento ma una serie di post/tweet memorizzati in qualche modo e in base all'applicazione si dovranno utilizzare le API messe a disposizione degli sviluppatori per recuperare il contenuto dei post/tweet desiderati. Per i casi in cui si lavora con articoli scientifici oppure libri, invece, si avrà molto probabilmente il documento memorizzato in un formato di tipo XML (o HTML) e si dovrà utilizzare xpath (o DOM selector) per estrarre il contenuto. Da notare che l'obiettivo di questo task consiste semplicemente nell'estrarre il testo ed i relativi metadati per poterlo identificare univocamente.
Come esempio, supponiamo di avere uno script di "*Web Scraping*"[3] per l'estrazione del contenuto da pagine di Wikipedia e di avere il seguente frammento:

> \tLa Rivoluzione francese fu un periodo di sconvolgimento sociale, politico e culturale estremo, e prevalentemente violento, avvenuto in Francia tra il 1789 e il 1799, poi allargatosi in Europa con le guerre rivoluzionarie francesi e le guerre napoleoniche.\n In storiografia è lo spartiacque tra età moderna ed età contemporanea.

Clean Text

Durante la fase di estrazione può capitare che il testo estratto contenga dei caratteri o formattazioni visive che per il task sono completamente superflue: come gli spazi ripetuti oppure multipli a capo. Lo scopo di questa fase è di rimuovere tutte le informazioni inutili per mantenere una rappresentazione uniforme ed omogenea delle frasi estratte. Per raggiungere lo scopo, possono essere utilizzate alcune funzioni di libreria comuni a molti linguaggi oppure utilizzare "*regular expression*" (regex) per scrivere regole per la rimozione dei caratteri (o ripetizione di caratteri) indesiderati.
Applicando questo processo alla frase precedentemente estratta, otteniamo il seguente risultato:

> La Rivoluzione francese fu un periodo di sconvolgimento sociale, politico e culturale estremo, e prevalentemente violento, avvenuto in Francia tra il 1789 e il 1799, poi allargatosi in Europa con le guerre rivoluzionarie francesi e le guerre napoleoniche. In storiografia è lo spartiacque tra età moderna ed età contemporanea.

[2]Processo di analisi del testo digitale per determinare se il tono emotivo del messaggio è positivo, negativo o neutro

[3]Tecnica di estrazione di dati da un sito web per mezzo di programmi software

Lowercase

I task di Information Retrieval richiedono un input da utente, pertanto, mantenere la distinzione tra uppercase e lowercase si rileverebbe controproducente in quanto, durante la scrittura, difficilmente un utente rispetta la scrittura di luoghi o nomi con le iniziali maiuscole. Tuttavia, anche se l'utente scrivesse le query con caratteri sia maiuscoli sia minuscoli rimarrebbe l'incognita dell'errore umano. Questo secondo caso è ancora più subdolo, perché rende difficile capire se la query contiene un carattere maiuscolo per errore oppure no. Per queste ragioni, avere le query case-sensitive comporterebbe enormi ripercussioni sui risultati ottenuti, pertanto, è consigliato di lavorare con stringhe aventi carattere minuscolo.
Dopo questa fase, applicata alla nostra stringa di esempio, otteniamo la seguente stringa:

> la rivoluzione francese fu un periodo di sconvolgimento sociale, politico e culturale estremo, e prevalentemente violento, avvenuto in francia tra il 1789 e il 1799, poi allargatosi in europa con le guerre rivoluzionarie francesi e le guerre napoleoniche. in storiografia è lo spartiacque tra età moderna ed età contemporanea.

Tokenization

Ogni frase deve essere rappresentata come una lista di token (parole). Ad ogni modo, non è sufficiente utilizzare lo spazio come metodo di separazione, in quanto non terrebbe conto della punteggiatura o di specifici casi di lingue straniere (basti pensare all'arabo oppure alle lingue con sinogrammi come il cinese). La peculiarità di questa fase è data dalla sua forte dipendenza con la lingua considerata, infatti, per separare correttamente le parole è necessario costruire regole fortemente influenzate dall'alfabeto utilizzato e dalla sua grammatica. Per questo motivo, esistono per lingue differenti diversi automi (o regular expression) per questo task [58, 38].
Applicando la fase di Tokenization alla stringa di esempio, otteniamo la seguente lista di token:

> ['la', 'rivoluzione', 'francese', 'fu', 'un', 'periodo', 'di', 'sconvolgimento', 'sociale', ',', 'politico', 'e', 'culturale', 'estremo', ',', 'e', 'prevalentemente', 'violento', ',', 'avvenuto', 'in', 'francia', 'tra', 'il', '1789', 'e', 'il', '1799', ',', 'poi', 'allargatosi', 'in', 'europa', 'con', 'le', 'guerre', 'rivoluzionarie', 'francesi', 'e', 'le', 'guerre', 'napoleoniche', '.', 'in', 'storiografia', 'è', 'lo', 'spartiacque', 'tra', 'età', 'moderna', 'ed', 'età', 'contemporanea', '.']

Lemmatization vs Stemming

In linguistica è comune la flessione, un meccanismo per la creazione di parole in cui la stessa viene modificata per esprimere varie categorie grammaticali (il genere, il numero,

la persona etc.). Per i task di Information Retrieval mantenere questa ricca presentazione è controproducente per due ragioni:

1. diminuzione di match: mantenere le parole flesse diminuisce la possibilità di match poiché l'utente nella query può utilizzare una parola flessa non contenuta nel documento o viceversa

2. inefficienze nel salvataggio: si mantengono in memoria un numero di parole che difficilmente verranno utilizzate

Per risolvere questo problema esistono gli algoritmi di stemming e lemmatization.
La tecnica di stemming consiste nel riportare una parola alla sua forma base rimuovendo suffissi e forme flesse, esponendo a due tipi di errore:

1. under stemming: quando due parole appartenenti allo stesso gruppo concettuale sono ridotte a due forme di base distinte (stem). Per esempio, le parole "*data*" e "*datum*", entrambe dovrebbero essere ridotte allo stem "*dat*" ma la seconda, per alcuni algoritmi, viene ridotta a "*datu*"

2. over stemming: quando due parole che dovrebbero essere ridotte a due stem diversi, vengono ridotte alla stessa radice (stem). Per esempio, le parole "*new*" e "*news*", la seconda potrebbe essere ridotta a "*new*" in quanto l'algoritmo di stemming, per errore, riconosce la parola come plurare e rimuove la lettera s

Gli algoritmi di stemming sono classificati in tre famiglie [25]:

1. truncating: metodi incentrati sulla rimozione dei suffissi o prefissi di una parola

2. statistical: metodi basati su analisi e tecniche statistiche

3. mixed: come suggerisce il nome, sono metodi che utilizzano un approccio ibrido delle due famiglie precedenti

Le tecniche di lemmatizzazione raggruppano le forme flesse di una parola in modo da poterle analizzare come un singolo oggetto, identificate dal lemma (forma canonica).
La principale differenza tra lemmatizzazione e stemming consiste nel fatto che lo stemming prende solo in considerazione la parola ed ignora il contesto in cui viene utilizzata. Quindi, per poter utilizzare un algoritmo di lemmatizzazione, molte implementazioni fanno affidamento ad un vocabolario lessicale, come ad esempio WordNet [40], in cui per ogni parola vengono considerate le possibili forme flesse (aggettivo, verbo, nome) e in base al ruolo che la parola svolge viene riportata ad una forma flessa differente.
Diversi studi in letteratura [27, 2] hanno dimostrato che in generale è preferibile utilizzare un algoritmo di lemmatizzazione.
Applicando la lemmatizzazione ai precedenti token, otteniamo:

['il', 'rivoluzione', 'francese', 'essere', 'uno', 'periodo', 'di', 'sconvolgimento', 'sociale', ',', 'politico', 'e', 'culturale', 'estremo', ',', 'e', 'prevalentemente', 'violento', ',', 'avvenire', 'in', 'Francia', 'tra', 'il', '1789', 'e', 'il', '1799', ',', 'poi', 'allargatosi', 'in', 'Europa', 'con', 'il', 'guerra', 'rivoluzionaria', 'francese', 'e', 'il', 'guerra', 'napoleonico', '.', 'in', 'storiografia', 'essere', 'il', 'spartiacqua', 'tra', 'età', 'moderno', 'e', 'età', 'contemporaneo', '.']

Stop Words and puntaction

In Information Retrieval con il termine "*stop words*" si indicano tutte quelle parole che aggiungono poche informazioni e che spesso sono irrilevanti per le query dell'utente. Furono scoperte da Hans Peter Luhn nel 1958 [32].
In particolare, le seguenti categorie rientrano nella definizione di stop words [26]:

- articoli (determinativi e indeterminativi)
- congiunzioni
- preposizioni

La rimozione delle stop words può avvenire attraverso il filtraggio dei token contro una lista precompilata delle stop words (metodo classico) oppure avvenire senza una lista precompilata, analizzando la frequenza delle parole all'interno del documento e rimuovendo quelle più frequenti (basata sulla legge di Zipf [61]).
Applicando la rimozione delle stop words e della punteggiatura all'array di token, otteniamo:

['rivoluzione', 'francese', 'essere', 'periodo', 'sconvolgimento', 'sociale', 'politico', 'culturale', 'estremo', 'prevalentemente', 'violento', 'avvenire', 'Francia', '1789', '1799', 'poi', 'allargatosi', 'Europa', 'guerra', 'rivoluzionaria', 'francese', 'guerra', 'napoleonico', 'storiografia', 'essere', 'spartiacqua', 'età', 'moderno', 'età', 'contemporaneo']

Bigrams

Dopo l'ultima operazione, per ogni documento si ha una lista di token, ovvero una lista di parole che rappresentano il suo contenuto. Siccome, tale lista è composta da singole parole può anche essere chiamata lista di unigram, dove unigram fa riferimento al singolo token/parola.
Limitarsi a rappresentare i documenti come array di unigram non è sufficiente a catturare interamente i concetti presenti in essi. Infatti, per molte lingue è comune utilizzare una coppia di parole (token o unigram) per definire un nuovo concetto che differisce dal significato originale delle due parole. Ad esempio, si pensi ai token "*rivoluzione*" e

"*francese*", presi singolarmente hanno un significato ma considerati insieme indicano uno specifico evento storico. O ancora, si pensi alle parole "*debito*" e "*pubblico*" che unendole indicano l'indebitamento di uno stato ma prese singolarmente fanno riferimento a due concetti ben diversi.
Una soluzione "*naïf*" sarebbe quella di generare una lista di bigrams dalla lista di unigram: per ogni coppia adiacente di unigram viene generato un bigram, soluzione non ottimale poiché genera un numero di bigram lineare al numero di coppie di unigram e, quindi, inefficiente sia dal punto di vista della memoria occupata sia dal punto di vista della correttezza in quanto la maggior parte dei bigram generati sarebbero spazzatura.
Una soluzione più raffinata è l'approccio data-driven, che consiste nell'analizzare il numero di occorrenze di una coppia di unigram e che può essere implementato attraverso l'utilizzo "*Distributed Representations of Words and Phrases and their Compositionality*" [39] o "*Normalized (Pointwise) Mutual Information in Collocation Extraction*" [8].
La principale differenza tra i due metodi consiste nel fatto che il primo è stato sviluppato per migliorare la rappresentazione vettoriale, ponendo particolare attenzione sulle cosiddette "*espressioni idiomatiche*"[4]. Il secondo, invece, si basa sull'analisi delle occorrenze, ovvero, date due parole calcola la probabilità che compaiono insieme attraverso l'analisi della frequenza nella collezione di documenti.
Nel nostro caso non utilizzeremo l'embedding della frase, almeno per l'implementazione dell'algoritmo dello Sparse Vector Representation, perciò è stato scelto il secondo approccio.
La formula "*npmi*" è la seguente: $\frac{\ln p(word_a, word_b)}{(p(word_a) * p(word_b))} / -\ln p(word_a, word_b)$, dove $p(word) = \frac{word\_count}{corpus\_word\_count}$.
Infine, è necessario individuare il corretto valore di threshold da impostare per considerare due parole come bigram. Questo, è possibile attraverso un approccio di tipo empirico, ovvero, dato il corpus basta individuare un insieme di token per analizzare quali vengono correttamente conosciuti come bigram e quali no.
I bigram riconosciuti dalla sequenza di token della frase di esempio sono i seguenti:

['rivoluzione_francese', 'sociale_politico']

BM25

Riassumendo: per ogni documento (appartenente alla collezione di documenti iniziale) si estrae il testo e si applicano le varie fasi di text-preprocessing, precedentemente esposte, per ottenere un array di token. L'insieme dei token che compaiono nella collezione di documenti viene definito "*vocabolario*". Per il calcolo dello score di rilevanza, l'algoritmo di ricerca ha bisogno di conoscere per ciascun token del vocabolario, il numero di occorrenze per ogni documento della collezione. Infatti, per tale ragione, viene costruita una matrice avente come riga l'identificativo del documento e come colonna i

[4]Espressione complessa di una lingua non interpretabile letteralmente

token del vocabolario. In questo modo, la cella (i, j) conterrà il numero di volte che la j-esima parola compare nel i-esimo documento. Per ragioni di efficienza, non è possibile rappresentare esplicitamente l'intera matrice, poiché la maggior parte dei valori sarebbe zero e, pertanto per ragioni di efficienza, vengono memorizzati solo le entry con valori strettamente maggiori di zero. Successivamente, esistono numerosi algoritmi per assegnare ad ogni documento un punteggio in base al numero di occorrenze di token della query, quello maggiormente utilizzato è il BM25 [47].
Data una query Q, contenente i token (da notare la generalità, potrebbero essere sia unigram sia bigram) $q_1, q_2, ..., q_n$, il punteggio di BM25 di un documento D è dato dalla seguente formula:

$$score(D, Q) = \sum_{i=1}^{n} IDF(q_i) \frac{f(q_i, D)(k+1)}{f(q_i, D) + k(1 - b + b * \frac{|D|}{avgdl})} \tag{1.1}$$

Dove $f(q_i, D)$ è il numero di volte che il token q_i compare nel documento, è sufficiente recuperare questo valore dalla matrice precedentemente calcolata, $|D|$ è la lunghezza del documento inteso come numero di token e $avgdl$ è la lunghezza media della collezione di documenti. Mentre k e b sono variabili libere che in assenza di avanzate ottimizzazioni vengono scelte nel range $k \in [1.2, 2.0]$ e $b = 0.75$ [30, 53, 55].
Infine, $IDF(q_i)$ misura l'Inverse Document Frequency per il token q_i. In Information Retrieval è una misura che penalizza i token (parole) che compaiono spesso nel corpus fornendo maggiore importanza alle parole rare. Da notare, che nella procedura di text-preprocessing appena esposta, le stop words vengono rimosse nella Sezione 1.3.1, senza rendere superflua questa misura. In caso contrario, tutti i token avrebbero lo stesso peso degradando la qualità dei risultati.

1.3.2 Dense Vector Representation (ricerca semantica)

Una delle principali limitazioni dello Sparse Vector Representation consiste nella sua rigidità in quanto considera solo i match esatti (exact match) senza tenere conto di possibili sinonimi delle parole. Per superare queste limitazioni esistono alcune tecniche che prevedono l'espansione della query con altri termini relativi alla ricerca [1]. Tuttavia, la pubblicazione del paper di BERT nel 2018 [15] ha messo a disposizione un generico Large Language Model, che data una frase ne restituisce il suo "*embeddings*". La parola "*embeddings*", nel contesto di Machine Learning, indica la rappresentazione di oggetti, per esempio immagini/video o testo, come forma matematica. Questa traduzione è fondamentale, in quanto con la rappresentazione originaria non è possibile utilizzare alcun modello di Machine Learning.
Una volta ottenuto l'embedding della frase, è possibile personalizzare l'architettura aggiungendo decoder o altri layer da applicare all'output, in base al task da svolgere.
Particolare attenzione verrà posta al task di Sentence Similarity, ovvero date due fra-

si assegnare un punteggio in base a quanto le due frasi siano, effettivamente, simili. Nell'ambito della ricerca semantica si distingue tra:

- ricerca simmetrica: quando la lunghezza della query è simile alla lunghezza degli elementi del corpus. Rientrano in questa categoria i task di "*question answer*", poiché la lunghezza (data dal numero di caratteri) delle query ed entry del corpus sono simili

- ricerca asimmetrica: di solito, la lunghezza della query è molto minore della lunghezza degli elementi del corpus. Rientrano in questa categoria i task di Information Retrieval classici dei motori di ricerca

L'importanza di questa distinzione, consiste nel fatto che esistono specifici dataset per le due categorie e quindi è fondamentale conoscere in quale categoria il proprio task rientra per scegliere il modello allenato sul corretto dataset (pensato per quello specifico task).

Introduzione a BERT e dei suoi limiti

BERT, Bidirectional Encoder Representations from Transformers, è basato sull'architettura di transformers per l'elaborazione del linguaggio naturale (Natural Language Processing).
Un Transformer [56] utilizza il meccanismo di self-attention per imparare la relazione delle parole in un testo. Ad esempio: data la frase "*The cat didn't cross the street because it was too scared*", il pronome personale "*it*" a chi è riferito? Questa può sembrare una domanda banale per un umano ma non per un algoritmo e lo scopo del meccanismo di self-attention è esattamente questo, collegare la parola "*it*" a "*cat*".
Per catturare questo, l'autore utilizza l'astrazione vettoriale Q, K e V, dove Q astrae la query dell'utente, K come l'insieme delle possibili chiavi (ad esempio, per la ricerca di un video dei possibili valori possono essere la descrizione del video o il titolo del video) e V come i migliori risultati.
La formula utilizzata è la seguente: $MultiHead(Q, K, V) = Concat(head_1, ..., head_h)W$ dove $head_i = Attention(XW_i^Q, XW_i^K, YW_i^V)$.
I valori delle matrici W_i^Q, W_i^K e W_i^V vengono costruiti durante la fase di addestramento del modello con il meccanismo della backpropagation mentre i vettori Y e X dipendono dal tipo di meccanismo di attenzione utilizzato (self-attention o cross-attention). Ad esempio: per il meccanismo di self-attention dell'encoder dipendono dall'output dell'encoder precedente, quindi si ha $X = Y$, invece per il meccanismo di cross-attention, X rappresenta l'output dell'ultimo encoder e Y l'output del precedente decoder.
I transformer utilizzano due distinti meccanismi:

- encoder: si occupa di leggere l'input, nello specifico mappa una sequenza di simboli in una rappresentazione continua

- decoder: data la rappresentazione continua, il decoder genera come output una sequenza di simboli

A differenza dei modelli direzionali, che leggono il testo dell'input sequenzialmente (da sinistra a destra o da destra a sinistra), l'encoder legge l'intera sequenza di parole allo stesso tempo, anche se più precisamente, dovrebbe essere considerato non direzionale. Questa caratteristica gli permette di imparare il contesto in cui una parola viene utilizzata in base alle parole vicine ad essa, in quanto cambiando la posizione di una parola in una frase, cambia anche la sua rappresentazione nell'embedding. Questo comportamento viene mostrato nella Figura 1.9, in cui l'input embedding viene calcolato come la somma tra l'identificativo del singolo token, l'identificativo del segmento a cui la frase appartiene (in quanto al modello possono essere date due frasi, separate dal token $[SEP]$) e dalla posizione del token all'interno della sequenza.

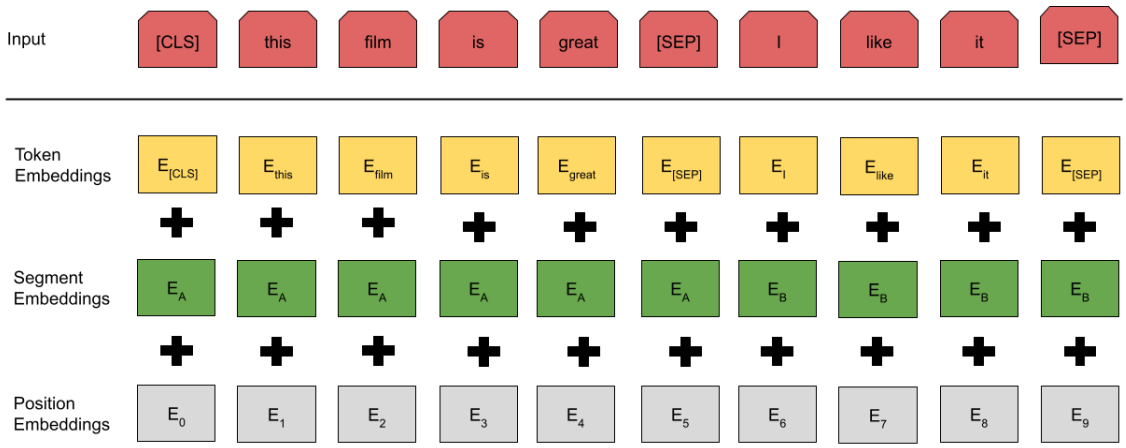


Figura 1.9: Come l'embedding viene calcolato da BERT

Da notare che BERT utilizza solamente uno stack di encoder in quanto il pre-training è stato svolto su task generici in cui l'utilizzo del decoder era superfluo. Una volta allenato il modello è possibile personalizzarlo, partendo dall'embedding generato dall'input, per risolvere il task specifico. Durante la fase di pre-training, BERT è stato allenato con il task Masked Language Modeling (MLM) e Next Sentence Prediction (NSP). Prima di passare una sequenza di parole a BERT, per il task di MLM, il 15% di queste parole viene sostituita con il token $[MASK]$. Il compito del modello è quello di predire il valore originale delle parole mascherate, utilizzando il contesto dato dalle parole della sequenza che non sono state mascherate. Invece, nel secondo task di training, ovvero Next Sentence Prediction, il modello riceve una coppia di frasi come input e deve imparare a predire se la seconda frase nella coppia è la successiva della prima nella sequenza originale.
Nella Figura 1.10 sono riportate le due versioni disponibili per BERT, che consistono in:

1. BERT Base: 12 layer (blocchi di encoder), 768 la dimensione dell'encoding (output) e 110 milioni di parametri
2. BERT Large: 16 layer (blocchi di encoder), 1024 la dimensione dell'encoding (output) e 340 milioni di parametri

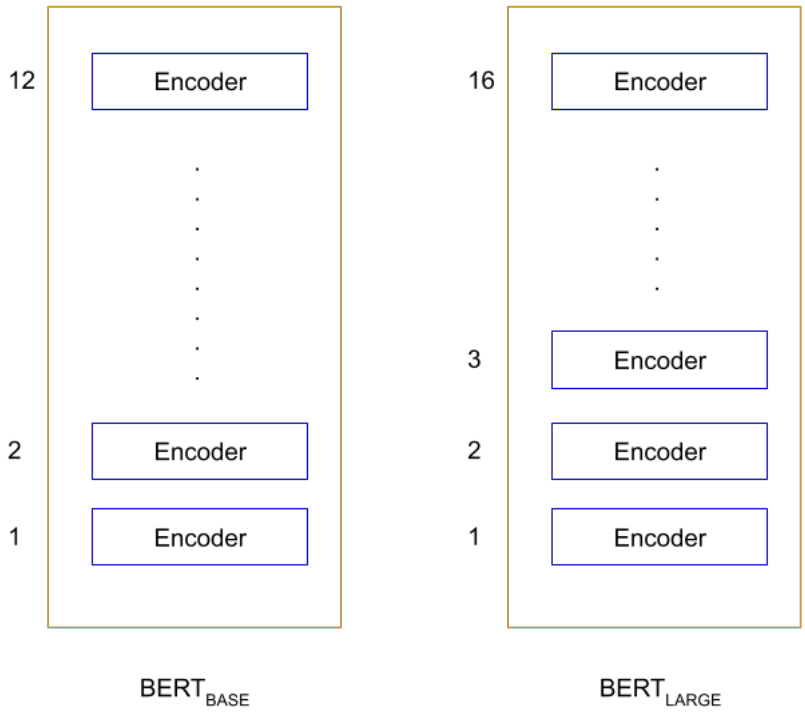


Figura 1.10: Architettura di BERT

Come SBERT supera le limitazione di BERT

Una grande limitazione del modello BERT è data dall'obbligo di dover fornire in input una coppia di frasi (separate dal token $[SEP]$) che emerge, sopratutto, nei task di Similarity Search (o Semantic Search), in quanto si ha una collezione di frasi (o documenti) da confrontare tra di loro per determinare la coppia più simile. Basti pensare che, già con una collezione di circa 10 000 frasi, il numero di iterazioni per trovare la coppia con punteggio maggiore richiede $n(n-1)$ iterazioni, ovvero 49 995 000 iterazioni. Che eseguite su una V100 GPU richiede circa 65 ore.
Per superare questo ostacolo SBERT [46] introduce il concetto di "*Siamese Network*", permettendo di passare frasi indipendenti allo stesso modello BERT per ottenere, per

ciascuna frase, la relativa rappresentazione. Nella Figura 1.11 è possibile vedere il confronto tra l'architettura siamese e una non-siamese: ciò permette di effettuare l'encoding di una collezione di frasi in maniera indipendente e una volta ottenuto l'embedding per ciscuna frase, applicare un qualunque algoritmo per misurare la similarità.
Nello specifico, questo viene effettuato passando una singola frase al modello e aggiungendo un pooling layer all'output di BERT. Nell'articolo vengono menzionate diverse strategie di pooling, la mean/max pooling applicata sull'output oppure utilizzare direttamente l'output del $[CLS]$ token. La strategia adottata di default è quella del mean pooling sull'output.

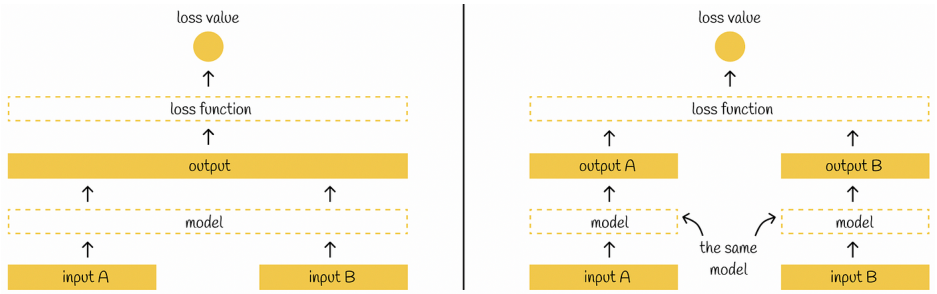


Figura 1.11: Architettura di SBERT, immagine disponibile al seguente indirizzo `https://towardsdatascience.com/sbert-deb3d4aef8a4`

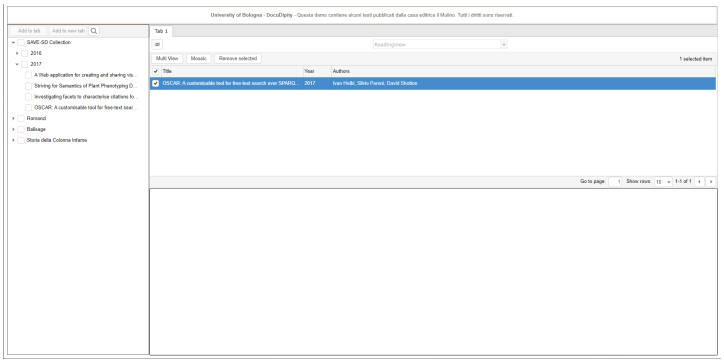
Capitolo 2

Background: DocuDipity

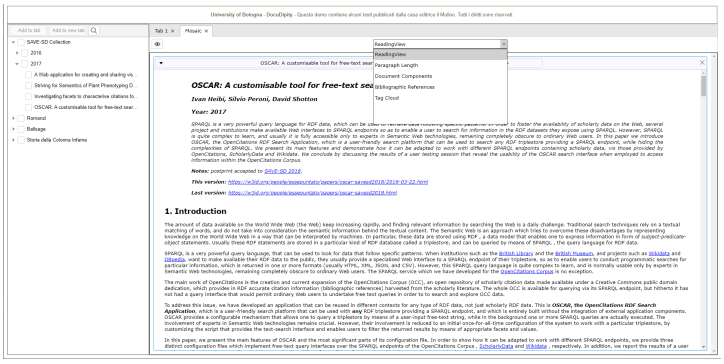
L'applicazione DocuDipity è stata, inizialmente, implementata dal DASPLab (Digital and Semantic Publishing Lab), un gruppo di ricerca dell'Università di Bologna, con l'obiettivo di sviluppare un web tool per l'analisi e il confronto di articoli scientifici [44]. L'analisi consiste nel consultare il documento/articolo utilizzando diverse visualizzazioni dello stesso, con lo scopo di aiutare l'utente a far emergere caratteristiche che con la visualizzazione ipertestuale passerebbero inosservate. Nella Sezione 2.1 vengono introdotti i concetti principali dell'applicazione e le visualizzazioni implementate, mentre nella Sezione 2.2 verrà presentato un esempio d'uso per mostrare le potenzialità dell'applicazione.

2.1 Docudipity: Introduzione e concetti principali

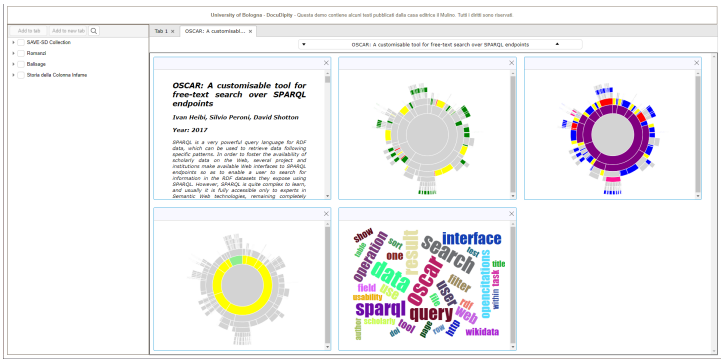
Il principio su cui si basa Docudipity è il cosiddetto "*Overview first, zoom and filter, then details on demand*" [3] , principio cardine della disciplina di Visual Analysis.
L'idea è quella di avere a disposizione un'interfaccia in grado di dare una panoramica complessiva dei dati a disposizione (*overview*), ridurre la complessità della visualizzazione attraverso la rimozione delle informazioni superflue e lasciare in evidenza quelle maggiormente rilevanti (*zoom and filter*). Infine, permettere all'utente, in un qualunque momento, di selezionare item specifici o gruppi di interesse ed ottenere informazioni aggiuntive (*details on demand*). Questo principio, inizialmente, è stato applicato in Docudipity per fare analisi su un insieme di articoli scientifici, al fine di scoprire nuove caratteristiche e renderle evidenti agli utenti.
Il nome Docudipity ha origine proprio dal termine che fa riferimento alla scoperta di qualcosa di inaspettato mentre si sta cercando altro, ovvero la serendipità (dall'inglese *serendipity*).
Nella Figura 2.1a è mostrata la pagina iniziale dell'applicazione, composta da una barra laterale in cui sono presenti i documenti/articoli disponibili, e la tab in cui vengono



(a) Pagina iniziale di Docudipity



(b) Docudipity - Mosaic



(c) Docudipity - MultiView

Figura 2.1: Docudipity confronto tra mosaic view e MultiView

visualizzati i documenti/articoli selezionati per l'analisi. Successivamente, è possibile scegliere due modi per la visualizzazione dei documenti selezionati:

1. MultiView, Figura 2.1c: vista disponibile, solamente, per singolo documento, si apre una nuova finestra con le viste alternative disponibili

2. Mosaic, Figura 2.1b: disponibile sia per singolo documento che per molteplici do-

cumenti, si apre una nuova finestra in cui è possibile cambiare la vista da utilizzare sul sottoinsieme di documenti selezionati. Inoltre, è possibile usare le frecce per navigare la lista dei documenti selezionati

2.1.1 HyperText View - RASH

La vista ipertestuale è basata su una versione semplificata di HTML, RASH [43], che mette a disposizione un insieme di strumenti per la validazione e visualizzazione di documenti scritti in questo formato. Per maggiori dettagli tecnici relativi a RASH e alla teoria su cui si basa, si rimanda alla Sezione 4.3.1 del Capitolo 4.
Per ogni documento/articolo viene riportata la "*Table Of Content*" con link navigabili, inoltre ogni riferimento bibliografico presente nell'articolo viene gestito per evidenziare, quando cliccato da utente, la relativa entry nella bibliografia e viceversa. Lo stile applicato agli articoli è quello utilizzato da RASH, il quale ha messo a disposizione le regole di stile per uniformare la rappresentazione di documenti che seguono il suo standard e una suite di script per la gestione della numerazione delle sezioni e sottosezioni. Questo permette di avere uno stile uniformato ed evitare l'utilizzo di molteplici CSS.
La vista ipertestuale ha il vantaggio di avere una buona rappresentazione del dettaglio dei documenti ma tende a offrire una scarsa panoramica del documento nel suo insieme, in quanto utilizzando una visualizzazione sequenziale con documenti di grandi dimensioni è possibile mostrare solo una parte del documento perdendo di vista la sua struttura.

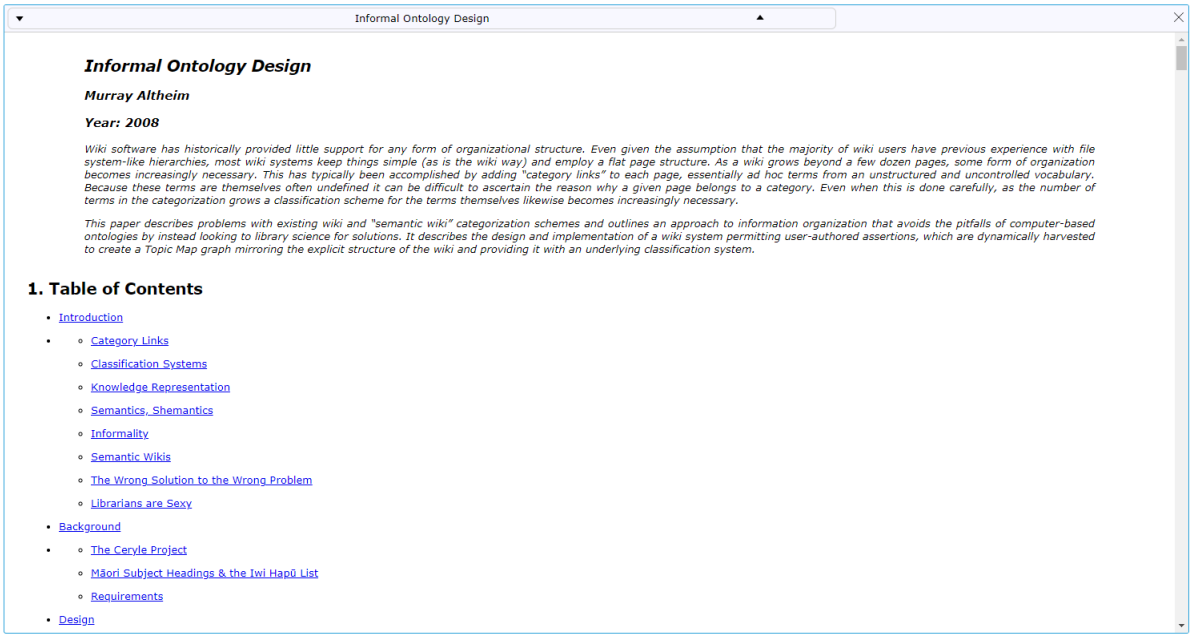


Figura 2.2: Vista ipertestuale di un articolo scientifico

2.1.2 SunBurst View

In Docudipity la visualizzazione SunBurst viene utilizzata per mostrare la struttura di un documento. Il centro rappresenta l'intero documento, il primo livello rappresenta gli elementi in cui il documento è diviso (ad esempio capitoli/sezioni, introduzione/abstract, bibliografia etc.). Successivamente, salendo di livello vengono rappresentati il contenuto di ciascun elemento precedente, come ad esempio sottosezioni, paragrafi, figure, tabelle e/o liste. Fino ad arrivare all'ultimo livello che, in generale, rappresenta un blocco di testo. La grandezza di ciascun elemento del SunBurst dipende dal numero di caratteri che contiene. A differenza della vista ipertestuale, che offre una buona rappresentazione del dettaglio di un documento, con SunBurst viene favorita una panoramica generale del documento, che attraverso l'utilizzo di colori e tooltip suggeriscono all'utente il ruolo dei vari segmenti (viola per le sezioni, blue per i blocchi di testo e rosso per le liste). Nella Figura 2.3 viene riportato un esempio di visualizzazione SunBurst per un documento della collezione Balisage[1].

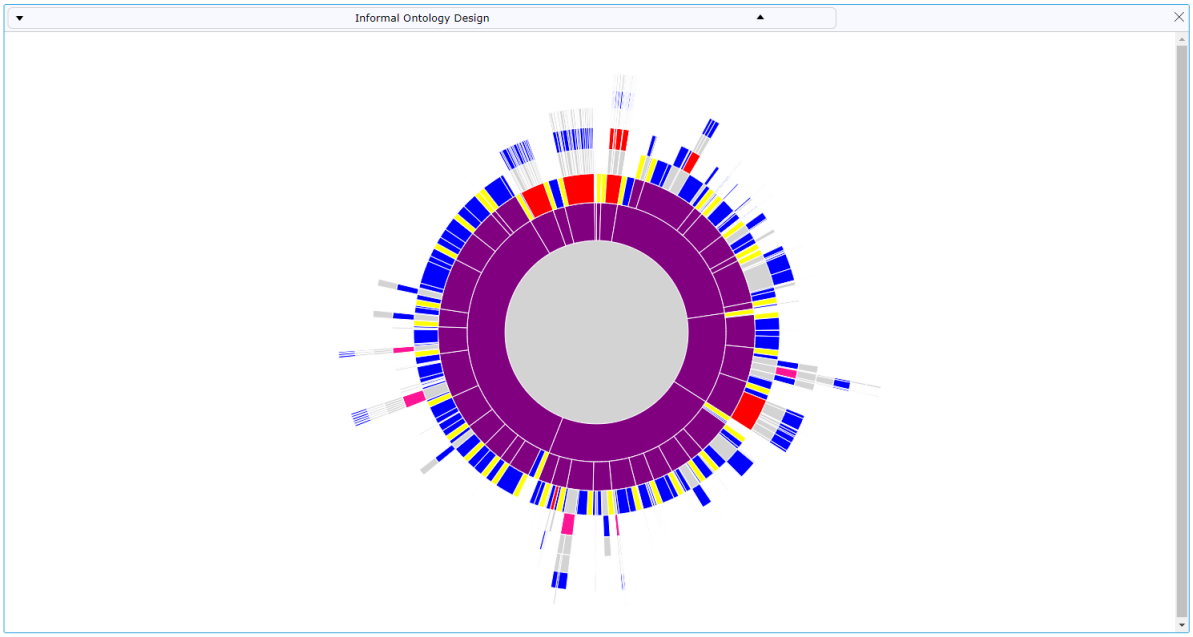


Figura 2.3: Visualizzazione SunBurst per un documento della collezione Balisage

[1] https://www.balisage.net/index.html

2.1.3 TagCloud View

La TagCloud View è generata a partire dalla vista ipertestuale, dalla quale si estraggono i blocchi di testo e si applicano le tecniche di text preprocessing viste nella Sezione 1.3.1 dello Stato dell'Arte dell'Information Retrieval. In questo modo, si ha per ogni parola la sua rappresentazione canonica, riducendo la dimensione del vocabolario e aumentando la correttezza della vista, in caso contrario si avrebbe il rischio di rappresentare un grafico contenente solamente stop words e/o diverse coniugazioni di una stessa parola. Il vantaggio di questa vista consiste nella possibilità di avere visualizzate le parole più ricorrenti per far emergere la categoria o l'argomento trattato nel documento.

Figura 2.4: Visualizzazione TagCloud per un documento della collezione di Balisage

2.1.4 Coordinated View

Una delle funzionalità implementate e messe a disposizione da Docudipity riguarda la sincronizzazione delle visualizzazioni. Ovvero, durante l'analisi di un documento, quando l'utente seleziona un elemento del SunBurst, la visualizzazione ipertestuale (dello stesso documento) si sincronizza evidenziando il medesimo elemento. Inoltre, quando si posiziona il cursore sopra ad un frammento di testo nella visualizzazione ipertestuale, nel SunBurst viene evidenziato sia il frammento di testo che gli elementi gerarchicamente correlati a tale frammento. Nella Figura 2.5 viene mostrato un esempio di sincronizzazione relativo alla vista ipertestuale quando si seleziona un elemento del SunBurst.
I benefici di tale funzionalità sono la possibilità di combinare i principali vantaggi delle due visualizzazioni, cioè la capacità del SunBurst di rappresentare in uno spazio relativamente ristretto l'organizzazione gerarchica di un documento, anche di grandi dimensioni, e il dettaglio offerto dalla visualizzazione ipertestuale.

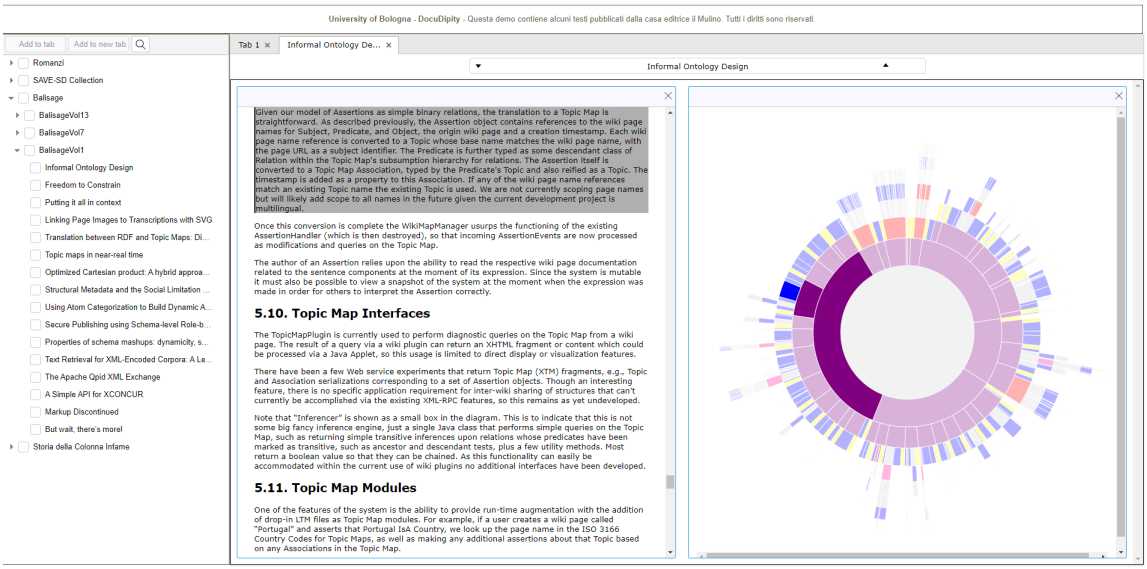


Figura 2.5: Sincronizzazione vista ipertestuale con SunBurst

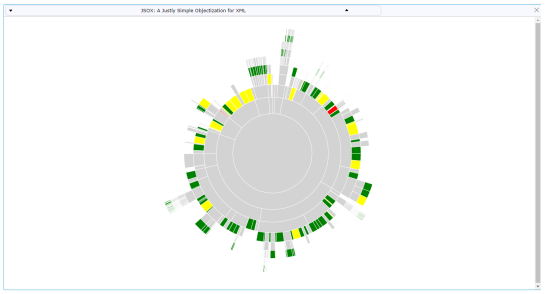
2.2 Docudipity, un esempio d'uso: Stili di scrittura

In questa Sezione mostreremo le potenzialità offerte dall'applicazione Docudipity per analizzare documenti scientifici. Per tale analisi, sono stati utilizzati alcuni articoli scientifici presentati al convegno Balisage Markup Conference che, dal 2008, mette a disposizione in formato XML gli articoli presentati. L'analisi presa in considerazione è quella sullo stile e sul linguaggio utilizzato da/dagli autori dell'articolo. L'idea di base è che esistono differenti stili di scrittura che possono variare da autore ad autore, che vengono nascosti dalla visualizzazione dettagliata ipertestuale ma che con l'utilizzo della vista SunBurst si possono, con maggior facilità, scoprire ed evidenziare. Per esempio, considerando la lunghezza dei singoli paragrafi di un documento, si colorano di giallo i paragrafi che hanno una lunghezza simile alla media e di rosso/verde i paragrafi che hanno una lunghezza superiore/inferiore alla media. L'applicazione di tale analisi sulla collezione di documenti, precedentemente descritta, permette di evidenziare:

- nel caso di articoli con un solo autore lo stile adottato ovvero chi predilige uno stile conciso e con paragrafi corti (Figura 2.6b) e chi, invece, tende ad essere più prolisso e sviluppare il pensiero con frasi lunghe (Figura 2.6c)

- nel caso di articoli con più autori di individuare le parti alle quali un autore ha contribuito, come mostrato nella Figura 2.6a, in quanto emergono i differenti stili dell'autore

(a) Articolo scritto da più autori con differenti lunghezze dei paragrafi



(b) Articolo costituito principalmente da paragrafi corti (in verde)

(c) Articolo costituito principalmente da paragrafi lunghi (in rosso)

Figura 2.6: rappresentazione stili autori

Capitolo 3

Strategie per migliorare la ricerca: un'estensione di DocuDipity

In questo Capitolo verrà approfondito il contesto d'uso dell'applicazione DocuDipity, con particolare attenzione rivolta alla base di dati utilizzata e agli utenti per cui tale strumento è stato progettato. Nella Sezione 3.1 verranno introdotti e discussi i casi d'uso. Nella Sezione 3.2 si approfondirà la soluzione proposta per ciascun caso d'uso, precedentemente discusso, introducendo e analizzando ad alto livello i componenti necessari. Infine, nella Sezione 3.3 verrà discusso il componente relativo all'interfaccia mentre nella Sezione 3.4 il componente relativo al motore di ricerca.

3.1 Contesto e Problema

Il sistema DocuDipity è stato sviluppato in un contesto accademico con l'obiettivo di contenere, come basi di dati, un insieme di articoli scientifici e/o libri. Questa caratteristica lo accomuna ad una "*Biblioteca Digitale*", ovvero la presenza di un numero limitato di documenti e la disponibilità di numerosi meta-dati per ogni oggetto contenuto nella banca dati. Infatti, le collezioni di documenti utilizzate per la fase di valutazione, Capitolo 5, sono le seguenti:

- collezione di libri pubblicati dalla casa editrice "*Il Mulino*", il cui argomento verte su una delle seguenti categorie "*Economia*", "*Scienza Politica*" e "*Sociologia*"
- collezione di articoli scientifici presentati nelle conferenze Balisage[1] e SAVE-SD[2], dove il raggruppamento viene effettuato in base all'anno di pubblicazione degli articoli e non in base ad un argomento specifico

[1] `https://www.balisage.net/index.html`
[2] `https://save-sd.github.io/`

Sia i libri sia gli articoli scientifici sono documenti gerarchici, ovvero sono divisi per capitoli/sezioni e tutti gli elementi seguono questa rappresentazione, ciò ha permesso di integrarli nel sistema e di visualizzarne la struttura nelle viste alternative proposte. Data la presenza nella banca dati di articoli scientifici e libri di interesse principalmente accademico, il target degli utenti è stato individuato da studenti, ricercatori, professori o, semplicemente, studiosi che vogliono approfondire una specifica tematica.
Un primo caso d'uso riguarda la navigazione del documento con particolare attenzione alle parti risultate rilevanti per una ricerca d'utente. Per esempio, prendiamo in considerazione uno studioso di Economia che vuole approfondire un argomento incentrato sull'indebitamento dell'Italia, molto probabilmente non sarà interessato ad approfondire o leggere tutti i libri che trattano questo tema ma solamente le sezioni o sottosezioni che trattano il tema in modo più specifico. Nel contesto delle "*Biblioteche/Librerie Digitali*" per risolvere questo caso d'uso, per ogni libro restituito dalla ricerca, viene proposto un estratto ovvero un passaggio significativo del libro che si presuppone sia di interesse per la ricerca d'utente. Tuttavia, questa soluzione risulta limitante in quanto è possibile mostrare solamente un estratto alla volta, con conseguente perdita di vista del risultato nel suo complesso da parte dell'utente di estratti che potrebbero essere inerenti alla sua ricerca. Un altro caso d'uso riguarda la selezione di interi documenti quando di dimensione più contenuta rispetto a libri, come possono essere gli articoli scientifici. Consideriamo l'esempio di un ricercatore o dottorato interessato ad approfondire uno specifico tema per lo sviluppo del suo elaborato, come potrebbe essere l'argomento relativo al "*Semantic Web*". In questo caso, il suo interesse riguarda l'intero articolo scientifico (non una sua parte) e per aiutare l'utente è necessario implementare un meccanismo di ricerca che permetta di selezionare o interi documenti oppure solo le parti maggiormente rilevanti.

3.2 DocuDipity per ambiente di ricerca

Nella Figura 3.1, viene mostrata la finestra di ricerca dell'applicazione come soluzione proposta per i casi d'uso discussi nella Sezione precedente. Nello specifico, la soluzione si divide in due parti principali: la prima legata all'interfaccia, ovvero come presentare i documenti risultati rilevanti, e la seconda legata al calcolo dello score di rilevanza per determinare il sottoinsieme di documenti rilevanti.
Per quanto riguarda l'interfaccia, viene proposto l'utilizzo di una vista radiale come il SunBurst. I vantaggi dell'utilizzo di una vista radiale consistono nella possibilità di mostrare all'utente, non solo i risultati di ricerca, ma anche l'insieme di frammenti di un documento risultati rilevanti permettendo di superare la limitazione dell'estratto proposto dalle "*Biblioteche Digitali*". Inoltre, aiuta l'utente a navigare il documento in maniera efficiente verso le parti risultate rilevanti per la sua ricerca, superando la limitazione dello scorrimento manuale. La soluzione relativa all'interfaccia e all'utilizzo del SunBurst viene approfondita nella Sezione 3.3.

Infine, per ottenere i documenti rilevanti per la ricerca d'utente è necessario implementare un motore di ricerca che, data una query, assegni un punteggio a ciascun documento della collezione. Tuttavia, classicamente gli algoritmi di Information Retrieval sono stati sviluppati per lavorare a livello di documento e non di una sua parte (frammento). Nella Sezione 3.4 viene mostrato come è stata superata tale limitazione e come vengono calcolate le rilevanze per i casi d'uso esposti nella Sezione precedente, con particolare attenzione alla rilevanza locale per selezionare i documenti con il frammento maggiormente rilevante e alla rilevanza globale per selezionare i documenti maggiormente rilevanti considerando l'insieme dei frammenti dello stesso.

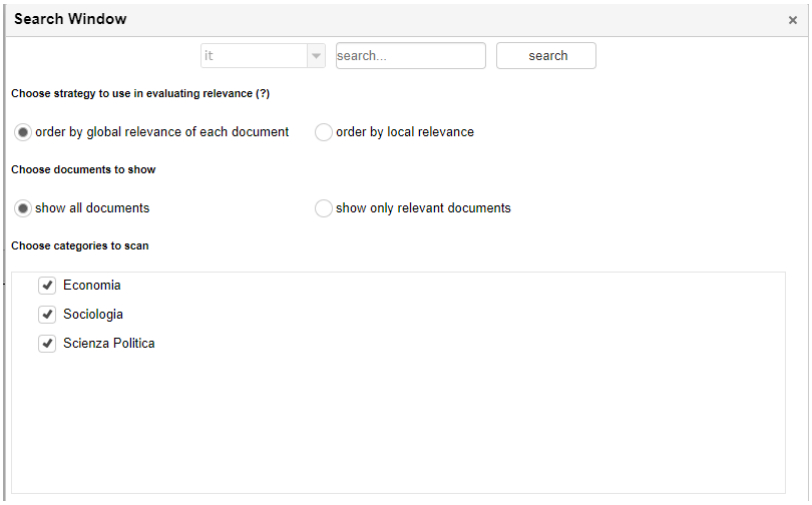


Figura 3.1: Maschera di ricerca

3.3 Interfaccia: SunBurst

La tecnica di visualizzazione SunBurst è stata utilizzata per risolvere due problemi: il primo riguarda la visualizzazione dei risultati di ricerca mentre il secondo riguarda come mostrare i frammenti rilevanti del documento. Il primo SunBurst rappresenta la struttura dell'insieme dei documenti/articoli ed evidenzia il sottoinsieme di essi, risultati rilevanti per la ricerca d'utente. Successivamente, cliccando sul singolo documento è possibile utilizzare la combinazione della "*reading view*" e "*SunBurst view*" del documento per analizzare i frammenti risultati rilevanti. Questo è reso possibile grazie al punteggio assegnato ad ogni frammento, la cui tonalità dipende dalla sua rilevanza. Tuttavia, per la visualizzazione dei risultati di ricerca non sempre basta il punteggio del singolo frammento ma, a volte, è necessario il punteggio complessivo del documento. Per tale motivo, è necessario definire diversi concetti di rilevanza, approfonditi nella prossima sezione.

3.3.1 SunBurst per visualizzare i risultati di ricerca

Classicamente, per una qualsiasi implementazione di un motore di ricerca (almeno per quelli strettamente commerciali) è dato per scontato che la presentazione dei risultati debba avvenire attraverso una paginazione di una lista di link. Una delle principali conseguenze di questo modo di presentare i risultati è il cosiddetto "*Golden Triangle*" [24], ovvero, un pattern seguito dagli utenti che consiste nel concentrare la propria "attenzione" solo su un'area specifica dello schermo (che ricorda, appunto un triangolo). Questo fatto è stato, ampiamente, sfruttato dai motori di ricerca (come ad esempio Google o Bing) per vendere tale spazio alle imprese interessate ad essere messe in evidenza per certe query degli utenti [12].
L'interfaccia proposta non raggruppa i risultati in maniera lineare ma circolare, ovvero organizza i documenti per argomento di appartenenza. In questa implementazione, ogni documento/articolo appartiene ad un solo argomento ed ogni argomento è associato ad un colore univoco per rendere evidente, a colpo d'occhio, l'associazione colore-argomento. Quindi, il centro del SunBurst rappresenta l'insieme di tutti gli argomenti, ogni livello rappresenta la relazione tra argomento e sotto-argomento ed, infine, l'ultimo livello mostra i documenti/articoli. Ogni sotto-argomento viene associato con il medesimo colore all'argomento padre. Inoltre, i documenti/articoli vengono visualizzati in ordine decrescente e la relativa intensità del colore è influenzata dal punteggio di rilevanza assegnatogli, per cui più un documento è rilevante maggiore sarà la sua tonalità. Se un documento risulta non rilevante, viene colorato di grigio. Il vantaggio di questa soluzione è la possibilità di avere una panoramica generale dei risultati permettendo di effettuare una prima analisi sulla distribuzione dei risultati di ricerca per argomento.

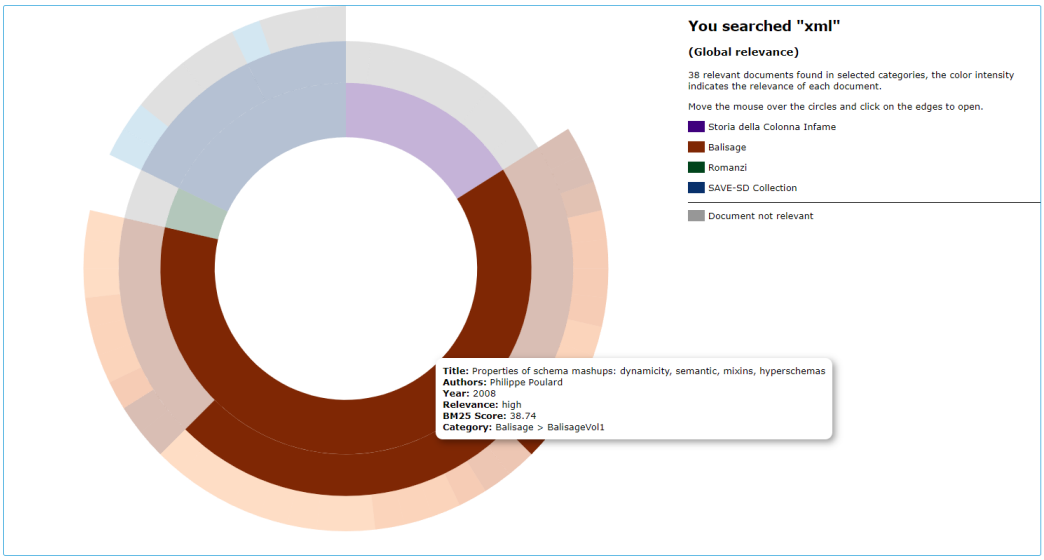


Figura 3.2: Applicazione del SunBurst ai risultati di ricerca

3.3.2 SunBurst per evidenziare frammenti rilevanti

Una prima soluzione per evidenziare i frammenti di un documento, consiste nel riutilizzare la vista del SunBurst per visualizzarne la struttura, con l'aggiunta del punteggio dello score per i blocchi di testo risultati rilevanti per la query. In questo modo, durante la visualizzazione i segmenti vengono evidenziati in base al colore dell'argomento di appartenenza del documento mentre la tonalità in base al punteggio del frammento.
Affacciando questa vista con la vista ipertestuale, si ottiene una panoramica del documento ed offre un modo efficiente e veloce per la navigazione del documento verso i frammenti rilevanti. Inoltre, permette di sfruttare le funzionalità implementate da DocuDipity, ovvero sincronizzazione della vista, in quanto, cliccando su un elemento del SunBurst la vista ipertestuale si sincronizza sul relativo elemento e viceversa. Quando si evidenzia un elemento nella vista ipertestuale, il corrispettivo elemento nella vista SunBusrt viene evidenziato, comprendendo gli elementi gerarchicamente correlati.
In Figura 3.3 è riportato un esempio per il caso d'uso relativo alla navigazione dei frammenti rilevanti, infatti utilizzando la colorazione per evidenziare i frammenti rilevanti (sia nella vista ipertestuale sia nel SunBurst) ed attraverso la tecnica "*Coordinated View*", l'utente è in grado di navigare il documento direttamente nei frammenti che sono rilevanti per la sua ricerca ignorando il resto.

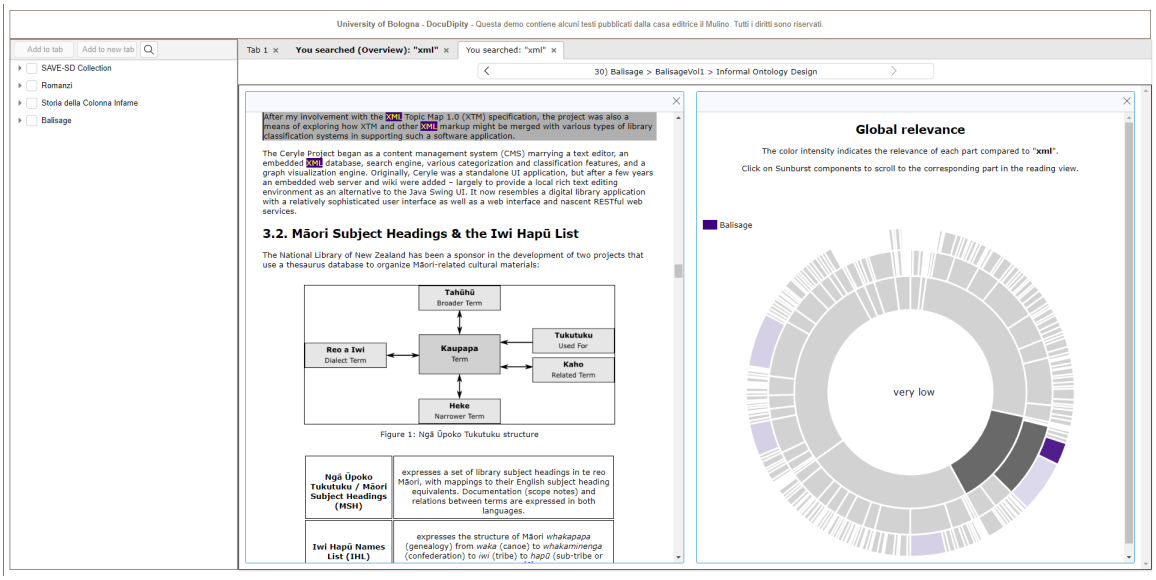
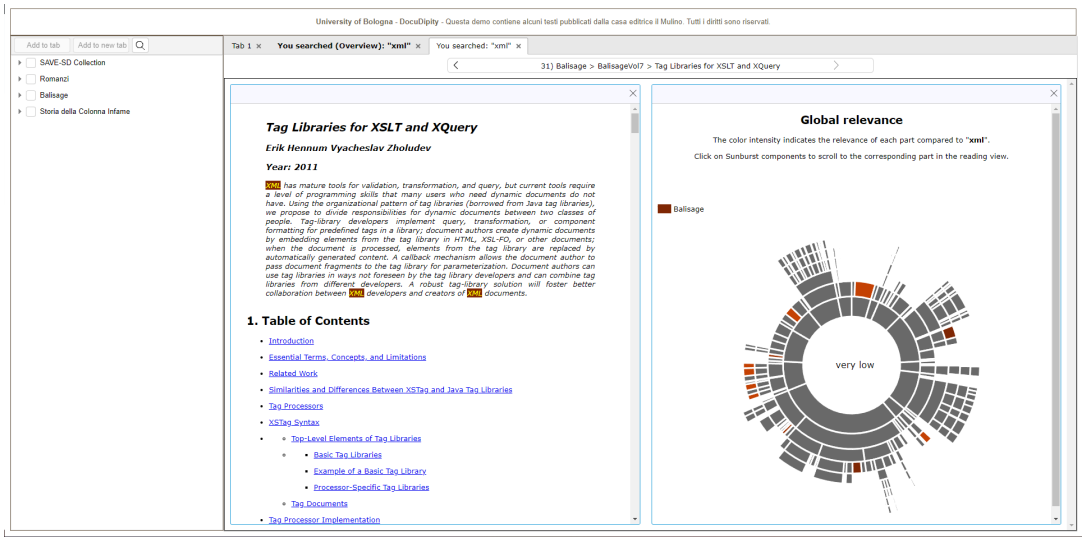


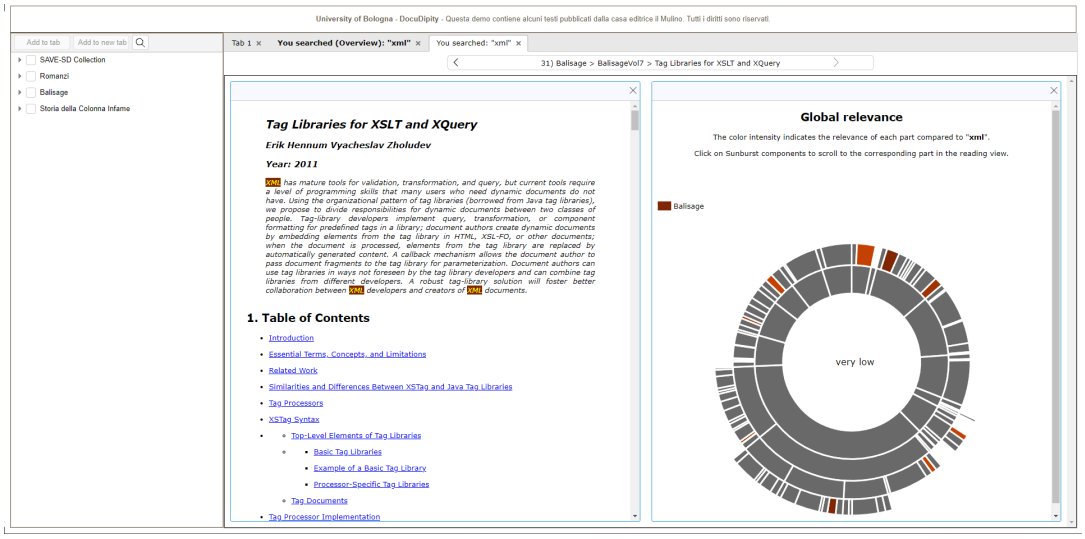
Figura 3.3: Caso d'uso: DocuDipity per navigare i frammenti rilevanti

Una limitazione di questa soluzione è data dalla visualizzazione dell'intera struttura del documento, poiché risulta essere controproducente per lo scopo di questa vista. Infatti, la rappresentazione iniettiva del livello gerarchico di tutti i tag del documento porta a perdere il "*focus*" incentrato sui blocchi di testo, in quanto questi ultimi possono

essere contenuti da molteplici container riducendo lo spazio per la visualizzazione.
Per tale motivo, si è optato per la derivazione di un nuovo documento RASH in cui i container vengono collassati per evidenziare i blocchi di testo (Figura 3.4). Questa operazione ha il vantaggio di una maggiore visibilità dei frammenti di testo e nella struttura semplificata, mentre lo svantaggio consiste nel fatto che non tutti gli elementi del documento RASH hanno un corrispettivo elemento nella rappresentazione SunBurst. Tenendo presente lo scopo del task, si è optato per questa soluzione.



(a) Vista delle similarità utilizzando il SunBurst



(b) Vista delle similarità utilizzando la versione compatta del SunBurst

Figura 3.4: Confronto tra il SunBurst e il SunBurst compatto per la vista delle similarità

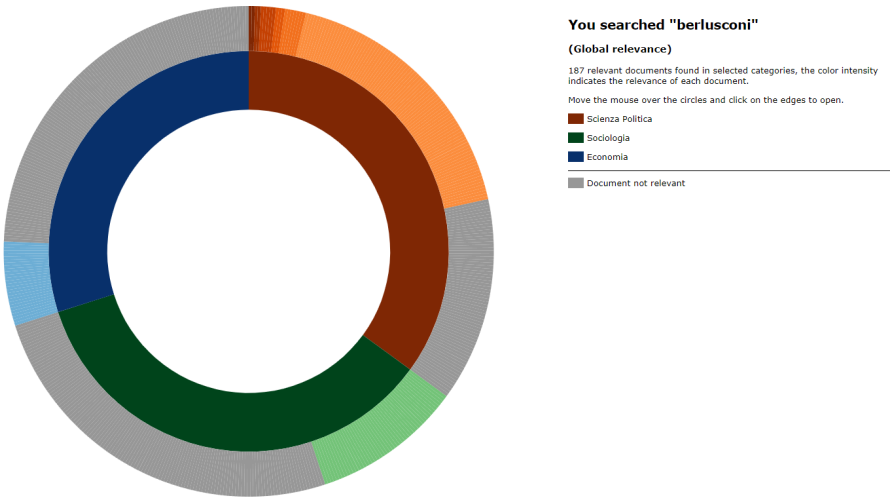
3.4 Information Retrieval: Introduzione al concetto di Rilevanza

Il termine "*frammento*" indica l'insieme dei blocchi di testo di un documento, come ad esempio paragrafi, titoli. Gli algoritmi di Information Retrieval calcolano un punteggio di rilevanza a livello di documento, non per un suo frammento. La conseguenza di questo consiste nell'impossibilità di utilizzare il SunBurst per la navigazione dei frammenti di un documento, in quanto non si ha la possibilità di assegnare un punteggio alla parte del documento che risulta rilevante. Per superare queste limitazioni, il punteggio di rilevanza è stato calcolato a livello di "*frammento*" e non del documento, quindi per ogni documento appartenente alla collezione si estraggono i blocchi di testo con i relativi identificativi in aggiunta a quello del documento, ai quali viene applicato l'algoritmo di Information Retrieval, per la cui implementazione si rimanda alla Sezione 4.2 del Capitolo 4. Questo permette di evidenziare nel SunBurst i singoli frammenti ma introduce il problema di come derivare il punteggio di rilevanza del documento da utilizzare nella selezione del sottoinsieme dei k-documenti rilevanti per la ricerca dell'utente. Una soluzione consiste nel calcolare una "*rilevanza locale*", ovvero il punteggio del documento viene calcolato come il massimo score dei suoi frammenti, permettendo di evidenziare solamente i documenti che contengono un blocco di testo altamente rilevante per la query, penalizzando i documenti che hanno un numero maggiore di frammenti (ma con un punteggio di singoli frammenti minori). Tuttavia, l'efficacia della tecnica per derivare lo score del documento, dai frammenti, dipende anche dal task dell'utente. Per esempio, se l'utente è interessato alla ricerca di una definizione, la "*rilevanza locale*" si adatta bene, in quanto si concentra sui singoli frammenti. Però esistono altri task per i quali questo approccio risulta controproducente, come ad esempio per i task in cui si cerca un documento che tratta un argomento specifico quindi non interessa il singolo frammento ma il documento nel suo complesso. Infatti, si introduce il concetto di "*rilevanza globale*", ovvero partendo dalla rilevanza dei frammenti di uno stesso documento, deriva la rilevanza a livello di documento, introducendo il problema di come calcolare/derivare tale punteggio. Durante la fase di implementazione, sono stati provati alcuni approcci, come il punteggio medio dei frammenti e la somma dei singoli punteggi, e dopo alcuni esperimenti, è stato scelto il metodo che considera la somma del punteggio dei frammenti del documento.

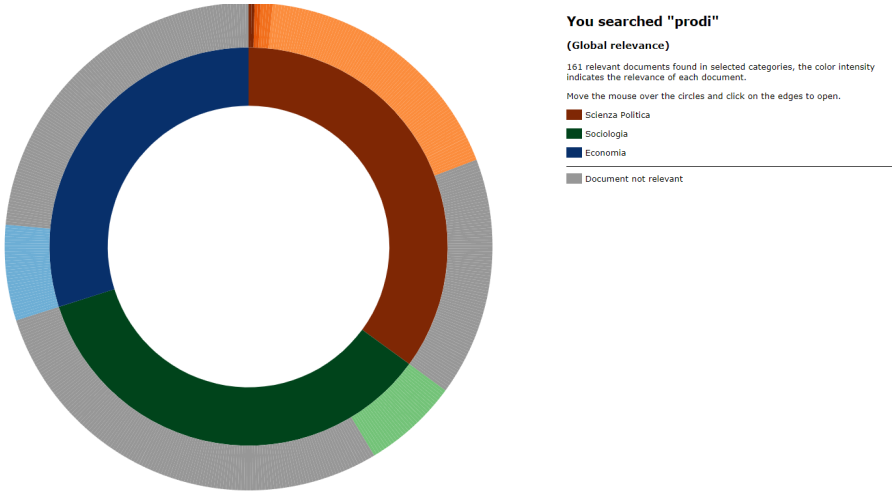
3.4.1 Esempio d'uso: rilevanza globale

Prendiamo come esempio il seguente caso d'uso: "*Analizzare per le categorie di Economia, Scienza Politica e Sociologia chi tra Berlusconi e Prodi è maggiormente citato*". Questo potrebbe essere un caso d'uso comune a molti ricercatori/studenti che devono approfondire l'impatto di leader politici su vari argomenti e/o confrontarne la popolarità.

Qui, la ricerca deve utilizzare la strategia globale in quanto siamo interessati all'insieme dei paragrafi in cui si potrebbe citare uno dei due leader e non ad un singolo paragrafo. Inoltre, per la presentazione dei risultati siamo interessati a mantenere anche i documenti non rilevanti per avere un confronto grafico tra le due ricerche. In Figura 3.5 sono riportati i risultati di ricerca, dove i riferimenti ai due leader sono particolarmente bilanciati anche se Berlusconi ha un numero di occorrenze maggiore rispetto a Prodi. Per entrambi, i libri che li citano più frequentemente sono quelli di "*Scienza Politica*" mentre per le restanti due categorie sono citati marginalmente.



(a) Risultati di ricerca per la keyword "Berlusconi"



(b) Risultati di ricerca per la keyword "Prodi"

Figura 3.5: Confronto quantitativo dei libri che contengono un riferimento a Berlusconi o Prodi, organizzati per categoria

Capitolo 4

Implementazione

Dopo aver discusso ad alto livello della soluzione proposta, in questo capitolo si approfondiranno i dettagli implementativi per la realizzazione dell'interfaccia e del meccanismo di rilevanza. Prima di entrare nella fase implementativa vera e propria, è doveroso presentare l'architettura dell'applicazione DocuDipity nella Sezione 4.1, necessaria per introdurre i componenti utilizzati e le, eventuali, dipendenze. Successivamente, si discuterà del motore di ricerca ibrido, della sua implementazione e delle relative tecnologie utilizzate (Sezione 4.2). Particolare attenzione verrà posta alla combinazione della tecnica dello Sparse Vector Representation (o ricerca lessicale) con la Dense Vector Representation (o ricerca semantica). Infine, nella Sezione 4.3 verrà approfondita la pipeline della generazione delle viste e delle tecnologie utilizzate per la loro realizzazione ed nella Sezione 4.4 verrà approfondita l'implementazione del meccanismo di evidenziazione dei blocchi risultati rilevanti nel SunBurst.

4.1 Architettura

Nella Figura 4.1 viene riportata l'architettura dell'applicazione, i cui componenti sono stati dockerizzati e orchestrati tramite la funzionalità di docker-compose [18]. Nello specifico:

- NGINX[1]: entry-point per le richieste http, si occupa di instradare le richieste verso il corretto container attraverso il "*pattern matching*" del path delle stesse. Inoltre, implementa il meccanismo del caching per evitare di instradare richieste fatte più volte
- Docudipity API: applicazione sviluppata in Node.JS [20], sfruttando il framework express [19], implementa le API per la gestione delle risorse necessarie al corretto

[1] `https://nginx.org/en/`

funzionamento. Maggiori dettagli si possono trovare nella tesi di Stefano Notari [17]

- MongoDB [41]: database non-relazionale per il salvataggio dei dati persistenti utilizzati dall'applicazione, ad esempio account e metadatati dei documenti
- Docudipity Frontend: implementa l'interfaccia dell'applicazione e mette a disposizione dell'utente una serie di viste alternative per l'analisi dei documenti. Per un approfondimento, si rimanda alla tesi di Kevin Maiani [16]
- Python Services: servizio che implementa la ricerca dei documenti, in particolare incentrato sul text-preprocessing dei documenti/query e sul calcolo dello score di rilevanza. I dettagli implementativi relativi al motore di ricerca e al text-preprocessing verranno affrontati nella Sezione 4.2
- Simple Docudipity: web app sviluppata in React [37] per implementare la classica vista lineare dei risultati di ricerca. Necessaria, per il test con gli utenti, al fine di avere un confronto tra l'interfaccia proposta ed una classica
- Resources: "*bind mount*"[2] tra una cartella della macchina host e il container, permette di rendere persistenti i file inseriti anche dopo la cancellazione del container

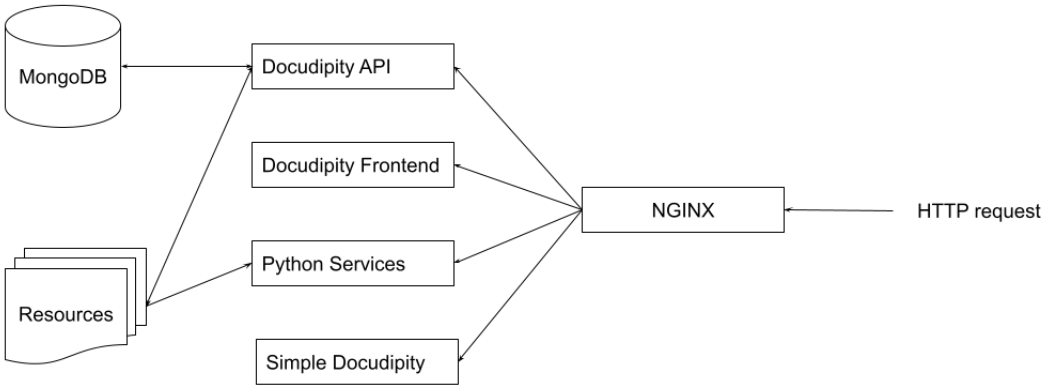


Figura 4.1: Architettura di DocuDipity

4.2 Search Engine: BM25 + SBERT

Il motore di ricerca è stato implementato attraverso il framework Flask di Python [22], per rendere accessibili le API ai frontend, descritti nella precedente sezione. Le

[2]data una directory, viene replicata in un punto differente

API messe a disposizione sono due: "*getBlockScore*" e "*getDocumentScore*", entrambe si aspettano come querystring la query dell'utente, la lingua in cui la query è stata scritta ed il tipo di rilevanza desiderata (locale o globale). La differenza tra le due API consiste nel fatto che la seconda restituisce la lista dei documenti rilevanti mentre la prima restituisce il JSON del SunBurst (versione compatta) con l'aggiunta dello score ai frammenti risultati rilevanti (per il documento indicato dalla querystring attraverso il campo "*expressionId*"). In questo modo, permette al frontend di colorare i segmenti in base al punteggio di rilevanza. Da notare che l'implementazione delle API è stata realizzata in modo da astrarre dal singolo algoritmo di Information Retrieval utilizzato. Infatti, è stata definitita la classe astratta "*BaseSimilarities*" che definisce i metodi che ogni singolo algoritmo di ricerca deve implementare, Figura 4.2. Questo permette alla classe "*HybridSearch*", che utilizza le istanze di "*BM25*" e "*SBERT*", di calcolare lo score finale e di adoperare in maniera agnostica una qualunque implementazione di un algoritmo di Information Retrieval, grazie alla definizione delle funzioni necessarie nella classe astratta e quindi di avere un template da seguire. Per il ramo dei Sparse Vector Representation è stato implementato l'algoritmo BM25, approfondito nella Sezione 4.2.1, invece per il ramo della ricerca semantica (o Dense Vector Representation) è stato utilizzato SBERT, i cui dettagli implementativi verranno approfonditi nella Sezione 4.2.2. Infine, si discute della tecnica utilizzata per generare il punteggio finale, combinando il punteggio di rilevanza della tecnica Sparse Vector Representation e Dense Vector Representation nella Sezione 4.2.3.

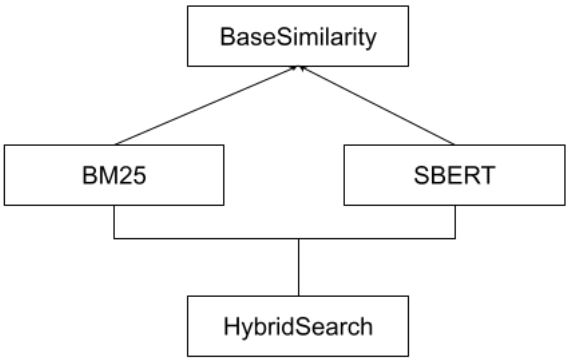


Figura 4.2: Diagramma UML dell'HybridSearch Engine

4.2.1 BM25

La parte di estrazione del testo viene effettuata dalla libreria lxml [4], data la rappresentazione RASH di un documento viene caricato il relativo albero e attraverso l'utilizzo di query xpath vengono estratti i nodi che contengono il testo. Nella Figura 4.1 è ripor-

tata l'espressione utilizzata, la quale estrae i nodi che hanno un pattern di tipo block (maggiori dettagli si possono trovare nella Sezione 4.3.1, ai fini di questa sezione basta sapere che sono i tag HTML che contengono testo) escludendo il testo di immagini o tabelle.

```
with open(path.join(self._DOCUMENT_COLLECTION_PATH, document), 'r',
    encoding='UTF-8') as rash:
    rashHtml = rash.read()
    parser = ET.HTMLParser(remove_comments=True, encoding='UTF-8')
    tree = ET.fromstring(rashHtml.encode(), parser)
    # extract from the tree all the block that may contain text
    tagBlockArray = tree.xpath("
        //*[@data-docudipity-pattern='block' and not(ancestor::figure)]
    ")
```

Listing 4.1: Xpath per l'estrazione dei blocchi di testo

Una volta estratto il testo, è applicata una funzione per la pulizia del testo, come mostra la Figura 4.2 la funzione "*rstrip*" rimuove il carattere spazio, eventualmente ripetuto, sia all'inizio sia alla fine della stringa passata in input, dopo di che vengono rimossi i caratteri per la nuova linea (\n), per il tab (\t) e, infine, gli spazi ripetuti all'interno della stringa. Durante l'implementazione è emerso un problema relativo alla tokenizzazione per la lingua italiana, la libreria nltk [5] non gestiva correttamente l'apostrofo delle parole, in quanto produceva in output un singolo token come l'unione tra articolo, il carattere dell'apostrofo e la parola. Questo risultava problematico in quanto la fase di tokenizzazione deve produrre token che contengono una singola parola, per tale ragione come patch il carattere dell'apostrofo viene sostituito con uno spazio.

```
def _normalizeText(self, text: str, language: str = "en") -> str:
    text = text.rstrip()
    # remove the \n character
    text = re.sub('\n', '', text)
    # remove the \t character
    text = re.sub('\t', '', text)
    # remove the duplicated spaces
    text = re.sub(' +', ' ', text)
    # if the language is italian replace the ' character with a space
    if(language == "it"):
        text = re.sub("'", ' ', text)
    return text
```

Listing 4.2: Funzione per la pulizia delle stringhe estratte

La parte relativa alla generazione dei token e della lemmatizzazione è stata realizzata attraverso la libreria nltk [5], come mostra la Figura 4.3. Tuttavia, per la parte di lemmatizzazione era stato originariamente utilizzato WordNet che, al momento non supporta la lingua italiana e per tale motivo è stata aggiunta la libreria spacy [23] per la lemmatizzazione dell'italiano.

```
def __generateTokens(self, text, language):
    customPunctuation = string.punctuation + "’"
    # each sentence must be forced to the lowercase
    text = text.lower()
    # tokenize the sentence
    tokens = word_tokenize(text,
        language=self.__SUPPORTED_LANGUAGE[language])
    # lemmatize
    if language == ’en’:
        lemma = WordNetLemmatizer()
        tokens = [ lemma.lemmatize(word, "v") for word in tokens ]
        tokens = [ lemma.lemmatize(word, "n") for word in tokens ]
    else:
        # where nlp is spacy.load("it_core_news_sm"), the model for the
    italian language
        tokens = [ nlp(word)[0].lemma_ for word in tokens ]
    # remove stop words
    tokens = [ token for token in tokens
        if token not in self.__stopWords ]
    # keep only alphanumeric tokens
    tokens = [ token for token in tokens
        if token not in customPunctuation ]
    return tokens
```

Listing 4.3: Generazione dei token e relativa lemmatizzazione

La parte di generazione dei bigram è stata realizzata attraverso la funzione “*Phrase*” realizzata da Gensim [45]. Una volta terminata la fase di tokenizzazione e lemmatizzazione si ottiene per ciascun documento la relativa lista di unigram, la quale viene utilizzata per analizzare le occorrenze delle parole e per individuare i possibili bigram. Nella Figura 4.4 viene riportata la relativa realizzazione, i valori per le variabili “*min_count*”, “*threshold*” e “*scoring*” sono rispettivamente 100, 0.1 e “*npmi*”. I valori di “*min_count*” e “*threshold*” sono stati determinati attraverso un approccio di tipo empirico, il modello veniva allenato sulla lista di unigram e, successivamente, si utilizzava una nuova lista di unigram per analizzare il comportamento. Ovvero, se due parole scollegate venivano riconosciute come bigram, si modificava la configurazione fino alla generazione dei bigram in maniera corretta. Infine, per l’implementazione dell’algoritmo BM25 è stata utilizzata la libreria rank-bm25, alla quale viene data come input la lista contenente, per ogni frammento di documento, l’unione tra unigram e bigram. Per ottenere la lista dei frammenti rilevanti, lo stesso processo descritto è applicato alla query e, una volta ottenuta la lista di token, questi vengono cercati nella matrice per ottenere il numero di occorrenze e il punteggio viene calcolato con la formula discussa nella Sezione 1.3.1.

```
def __trainPhrases(self):
    if not self.__sentencesList:
        self.__loadUnigram()

```

```
    bigramModel = Phrases([
        sentence[self.DocudipityFileProperties.tokens.value]
        for sentence in self.__sentencesList ],
        min_count=self.__MIN_COUNT,
        threshold=self.__BIGRAMS_FREQUQNCY_THRESHOLD,
        scoring=self.__SCORING_ALGORITHM
    )
    frozenModel = bigramModel.freeze()
    frozenModel.save(self.__PHRASE_MODEL_PATH)
```

Listing 4.4: Phrases addestramento sulla lista di unigram

4.2.2 SBERT

Con l'applicazione dell'encoding di SBERT ad un documento si ottiene come output la sua rappresentazione vettoriale. Questo impedisce di riutilizzare un algoritmo che assegna il punteggio di rilevanza in base alle occorrenze dei token della query nel documento, pertanto, è necessario utilizzare un nuovo meccanismo per misurare la similarità tra due vettori (documenti). Le principali tecniche per misurare tale similarità sono due: "*Cosine similarity*" e "*dot-product*".
La "*Cosine similarity*" misura il coseno dell'angolo tra due vettori, restituendo un valore nel range $[-1.0, 1.0]$, definito come il rapporto tra il "*dot-product*" dei due vettori e il prodotto della loro lunghezza mostrato nell'Equazione 4.1.

$$cos\theta = \frac{A \cdot B}{\|\boldsymbol{A}\|\|\boldsymbol{B}\|} \tag{4.1}$$

Un altro modo per definire la similarità è il "*dot-product*" che è possibile derivare dalla definizione del coseno. Infatti, il "*dot-product*" è il prodotto tra la lunghezza dei vettori e il coseno, come mostrato nell'equazione 4.2, e di conseguenza non restituisce più un valore nel range $[-1.0, 1.0]$ ma può essere un qualunque numero reale in quanto influenzato dalla lunghezza dei vettori.

$$AB = \|\boldsymbol{A}\|\|\boldsymbol{B}\|cos\theta \tag{4.2}$$

Da notare che nel caso di vettori normalizzati, ovvero i vettori la cui lunghezza è uno, il "*dot-product*" restituisce come misura quella del coseno, fatto dimostrato nell'Equazione 4.3 .

$$AB = \|\boldsymbol{A}\|\|\boldsymbol{B}\|cos\theta = 1 * 1 * cos\theta = cos\theta \tag{4.3}$$

Per ottenere il punteggio di similarità tra la query e i frammenti della collezione di documenti, bisogna applicare l'encoding sia alla query dell'utente sia ai singoli frammenti. Tuttavia, l'operazione di encoding su CPU risulta molto più lenta rispetto all'esecuzione su una GPU, inoltre il tempo richiesto è direttamente proporzionale al numero di

frammenti. Per tale ragione, l'encoding per i frammenti dei documenti è stato eseguito su colab (nell'environment che mette a disposizione, gratuitamente, una T4 GPU), i cui risultati sono stati memorizzati in un file pickle che veniva dato in input all'applicazione per avere l'encoding da confrontare con quello della query. Nella Figura 4.3 viene mostrata la panoramica generale per la generazione dell'encoding dei frammenti e della query per la ricerca semantica.
L'encoding è stato configurato per restituire come output un vettore normalizzato (la cui lunghezza è uno) e convertito come Tensore. Successivamente, siccome i vettori sono normalizzati, è possibile applicare il "*dot-product*" per ottenere la misura del coseno tra la query e il frammento.

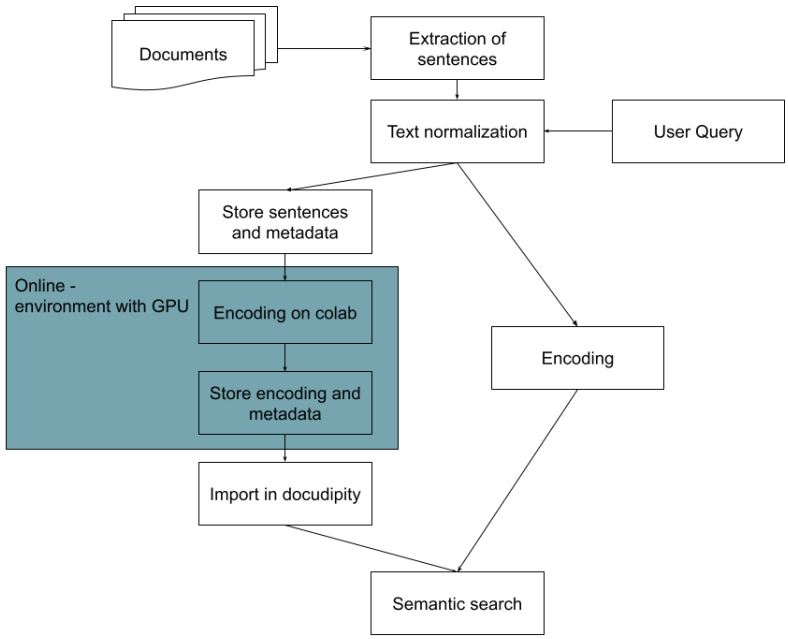


Figura 4.3: Panoramica dell'implementazione di SBERT e delle fasi per la generazione dell'encoding per i frammenti e query

4.2.3 Score

Per implementare il motore di ricerca ibrido è necessario trovare un modo per combinare il punteggio ottenuto da BM25 (ricerca lessicale) e da SBERT (ricerca semantica). In letteratura l'approccio seguito è l'utilizzo della funzione "*Weighted Sum Model*" [33, 6], che prevede un "*hyperparameter*" (iperparametro) α per determinare il peso dei due punteggi. In generale, i valori che BM25 può assegnare ad un documento possono variare nel range $[0, \infty]$, con la conseguenza dell'impossibilità di combinare tale valore

con quello prodotto da SBERT (che utilizzando la "*Cosine similarity*" rientra nel range $[-1.0, 1.0]$). Per superare questa limitazione è possibile normalizzare i valori calcolati dal BM25, attraverso la divisione di ciascun termine, per il massimo valore dell'attuale ricerca (Equazione 4.4).

$$Score_i = \frac{Score_i}{MAX(Scores)} \tag{4.4}$$

Infine, per determinare il punteggio finale per l'i-esimo documento si utilizza la Formula 4.5.

$$ScoreFinal_i = \alpha ScoreSbert_i + (1 - \alpha) ScoreBM25_i \tag{4.5}$$

4.3 Generazione viste

La pipeline per la generazione delle viste, Figura 4.4, inizia dalla rappresentazione RASH del documento ma, prima di procedere, è necessario generare per ogni tag del documento un identificativo univoco per risalire ad esso anche nelle altre rappresentazioni. Questo si effettua attraverso una visita dell'albero del documento e aggiungendo ad ogni elemento la proprietà "*data-docudipity-id*" con un intero che viene incrementato alla visita di ogni nodo. In questa fase, oltre all'identificativo, per ogni elemento viene aggiunto il pattern di appartenenza e vengono aggiornati i "*source*" delle immagini con i link per consentire il funzionamento all'interno dell'applicazione docudipity. Per l'aggiunta del pattern, si immette all'elemento l'attributo "*data-docudipity-pattern*", il cui valore dipende unicamente dal tag del nodo. Ciò è reso possibile grazie a RASH, che verrà approfondito nella Sezione 4.3.1 con relativa tabella di tag e pattern di appartenenza. Invece, per quanto riguarda la gestione delle immagini, si presentano fondamentalmente due casi:

1. URL assoluto: l'immagine viene scaricata nella cartella "*img*", relativa al corrente documento, dove sono presenti le immagini del documento/articolo e l'URL viene aggiornato con la nuova destinazione dell'immagine

2. URL relativo: l'immagine è già stata scaricata ed è già presente nella cartella "*img*", quindi non è necessario fare altro

In seguito, si crea la visualizzazione SunBurst approfondita nella Sezione 4.3.3. Come discusso nella Sezione 3.3.2, utilizzando la vista SunBurst generata dal documento RASH per evidenziare i frammenti rilevanti di un documento, risulta poco efficace a causa dell'eccessivo livello di annidamento. Per tale ragione è stata creata una nuova versione di RASH, il compact RASH, con un livello di annidamento ridotto che verrà discusso nella Sezione 4.3.2.

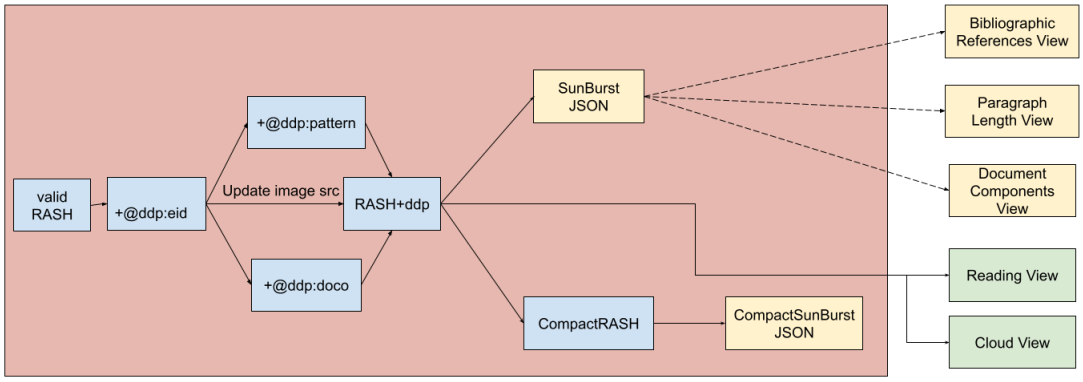


Figura 4.4: Panoramica della pipeline della generazione delle viste alternative di DocuDipity

4.3.1 RASH

RASH si basa sulla teoria dei pattern, il cui principio cardine prevede che ogni tag HTML ha uno ed un solo pattern strutturale, definito dalle seguenti proprietà:

- la possibilità di contenere testo (+t o -t)
- la possibilità di contenere altri blocchi (+s o -s)
- l'obbligo di essere contenuto da un altro elemento che può contenere o meno altro testo (+T o -T)

Combinando le proprietà sopra elencate si ottengono otto pattern:

1. Inline [+t+s+T]: un qualunque contenitore di testo e/o di altri blocchi, inclusi (anche ricorsivamente) altri elementi inline
2. Block [+t+s-T]: un qualunque contenitore di testo e/o di altri blocchi, escludendo (anche ricorsivamente) altri elementi block
3. Popup [-t+s+T]: un qualunque elemento senza testo ma che può essere utilizzato da altri elementi che permettono testo al loro interno
4. Container [-t+s-T]: un qualunque contenitore di sequenze di blocchi che non contengono direttamente testo
5. Atom [+t-s+T]: un qualunque contenitore di testo, senza la possibilità di contenere ulteriori blocchi al suo interno. Il cui utilizzo è permesso ad elementi che possono contenere altro testo ma non da container

6. Field [+t-s-T]: un qualunque contenitore di testo, senza la possibilità di contenere ulteriori blocchi al suo interno. Il cui utilizzo è permesso a container ma non da elementi che possono contenere altro testo

7. Milestone [-t-s+T]: un qualunque elemento senza contenuto (sia di testo che di altri blocchi) e che può essere utilizzato da blocchi che permettono altro testo ma non da un container

8. Meta [-t-s-T]: un qualunque elemento senza contenuto (sia di testo che di altri blocchi) e che può essere utilizzato da un container ma non da blocchi che permettono altro testo

Nella Tabella 4.1 sono mostrati, per ogni famiglia di pattern, i tag individuati da RASH per quella specifica famiglia. Il principale punto di forza di RASH consiste nella gram-

Pattern	RASH element
inline	a,code, em, math, q, span, strong, sub, sup, svg
block	figcaption, h1, p, pre, th
popup	none
container	blockquote, body, figure, head, html, li, ol, section, table, td, tr, ul
atom	none
field	script, title
milestone	img
meta	link, meta

Tabella 4.1: Associazione pattern con tag HTML

matica che definisce, in maniera chiara e restrittiva, l'utilizzo dei tag e permette di identificare per ogni tag il relativo ruolo in maniera non ambigua. Il vantaggio è evidente in quanto si evita il fenomeno in cui il ruolo dei tag può essere modificato attraverso l'uso di regole di stile scritte dagli sviluppatori. Invece, un possibile svantaggio di questo approccio è l'obbligo di dover scrivere, eventualmente, uno script per la conversione dal formato originario della collezione di documenti a RASH. Considerando i vantaggi e gli svantaggi, si è deciso di utilizzare RASH come formato supportato di default da Docudipity poiché i benefici, come ad esempio la rappresentazione omogenea di tutti i documenti delle collezioni e la verifica della validità di documenti, sono maggiori degli svantaggi introdotti.

4.3.2 CompactRASH

I documenti che hanno una struttura annidata di tag generano un SunBurst estremamente profondo che tende a perdere i blocchi di testo, per la visualizzazione delle

similarità risulta essere problematico per mostrare all'utente i frammenti di testo. Per tale ragione, è stato deciso di generare una nuova vista attraverso un file xslt applicato a RASH (Figura 4.5), determinando una versione maggiormente compatta.

```
<?xml version="1.0"?>
<xsl:stylesheet
    xmlns="http://www.w3.org/1999/xhtml"
    xmlns:h="http://www.w3.org/1999/xhtml"
    xmlns:xsl="http://www.w3.org/1999/XSL/Transform"
    exclude-result-prefixes="h"
    version="1.0">

    <!-- ================================  -->
    <!--            strip                  -->
    <!-- ================================  -->

    <xsl:template match="
        h:tbody | h:thead |
        h:a | h:code | h:em | h:math | h:q | h:span |
            h:strong | h:sub | h:sup | h:svg |
        h:figure |
        h:tr | h:td | h:th | h:li |
            h:p[ancestor::h:table] | h:p[ancestor::h:ul] |
        h:blockquote |
        h:p[h:img] |
        h:*[ancestor::h:section[@class='tableofcontents']]
    ">
        <xsl:apply-templates select="h:* |
            text() | comment()"
        />
    </xsl:template>

    <!-- ================================  -->
    <!--            pseudo block           -->
    <!-- ================================  -->
    <xsl:template match="h:table | h:ul">
    <xsl:copy>
        <xsl:attribute
            name="data-docudipity-pattern-compact">
            pseudoblock
        </xsl:attribute>
```

```
        <xsl:apply-templates select="@*"/>
        <xsl:apply-templates select="h:* |
            text() | comment()"
        />
    </xsl:copy>
    </xsl:template>

    <!-- ================================  -->
    <!--            rename                 -->
    <!-- ================================  -->
    <xsl:template match="h:pre">
        <p>
            <xsl:apply-templates select="h:* | @* |
                text() | comment()"
            />
        </p>
    </xsl:template>

    <!-- ================================  -->
    <!--            identity               -->
    <!-- ================================  -->

    <xsl:template match="h:* | @* | text() | comment()">
      <xsl:copy>
        <xsl:apply-templates select="h:* | @* |
            text() | comment()"
        />
      </xsl:copy>
    </xsl:template>

</xsl:stylesheet>
```

Listing 4.5: File XSLT per la generazione di compactRASH

Nello specifico, agisce sulle tabelle e sulle liste in cui si genera un nuovo blocco, detto "*pseudoblock*", in cui vengono collassati i contenuti delle stesse. In questo modo si ha una visualizzazione semplificata che permette di evidenziare i blocchi di testo. Partendo da questa nuova versione di RASH viene generato il relativo SunBurst per utilizzarlo nella similarities view.

4.3.3 SunBurst View

La generazione della vista SunBurst parte dalla rappresentazione RASH del documento, navigando l'albero dei tag che lo compongono. Partendo dal tag del body, viene effettuata una DFS (Depth First Search) come mostrato nella Figura 4.6. Il motivo della scelta di DFS è dovuta al fatto che la rappresentazione SunBurst deve rispettare la struttura del documento, poiché con una BFS (Breadth First Search) sarebbe risultato meno naturale. Per rappresentare nella visualizzazione SunBurst l'ampiezza di ogni segmento, si utilizza il numero di caratteri dell'elemento del documento RASH. Come panoramica generale, partendo dal body si visita l'albero con DFS e, una volta raggiunte le foglie, si restituisce il numero di caratteri e si propaga il risultato ai genitori che combinano con la somma la lunghezza dei rispettivi figli, fino a ritornare al body (elemento radice) con il numero totale di caratteri del documento. La sola informazione del numero di caratteri non sarebbe sufficiente in quanto non si avrebbero altre informazioni, per ogni elemento del SunBurst sull'elemento RASH corrispettivo. Quindi è necessario utilizzare una struttura dati più complessa, in cui memorizzare altre informazioni utili per preservare la relazione iniettiva tra gli elementi del SunBurst e RASH.

```
def dfsVisit(root, p_avg):
    # check title and img -> len(p)
    tag_name = ET.QName(root).localname
    if(tag_name == 'img' or tag_name == 'h1'):
        return {
            "htmltag": ET.QName(root).localname,
            "xmlId": root.get('data-docudipity-id'),
            "pattern": root.get('data-docudipity-pattern'),
            "text": p_avg,
            "children": []
        }
    if(len(root.getchildren()) == 0 ):
        return {
            "htmltag": ET.QName(root).localname,
            "xmlId": root.get('data-docudipity-id'),
            "pattern": root.get('data-docudipity-pattern'),
            "text": count(root.xpath('./text()')) if root.text != None
                else 0,
            "children": []
        }
    parent_json = {
            "htmltag": ET.QName(root).localname,
            "xmlId": root.get('data-docudipity-id'),
            "pattern": root.get('data-docudipity-pattern'),
            "text": count(root.xpath('./text()')) if root.text != None
                else 0,
            "children": []
        }
    for child in root.getchildren():
```

```
30          res = dfsVisit(child, p_avg)
31          parent_json["children"].append(res)
32          parent_json["text"] = parent_json["text"] + res["text"]
33      return parent_json
```

Listing 4.6: Generazione SunBurst

4.4 Query Highlight

Per suggerire all'utente le sezioni del documento risultate rilevanti per la sua ricerca, i segmenti del SunBurst vengono evidenziati con il medesimo colore dell'argomento di appartenenza del documento. Inoltre, siccome esistono paragrafi più o meno rilevanti in base al valore del punteggio, si voleva comunicare questa informazione in maniera grafica attraverso l'intensità del colore. Per l'implementazione è stata utilizzata la libreria D3 [7] attraverso la funzione "*quantize*", che permette di specificare un dominio e di segmentarlo in maniera uniforme in "k" segmenti, dove "k" è il numero di elementi nel range. Ovvero, dato un dominio (da zero al massimo punteggio ottenuto da un blocco), per ogni valore ad esso appartenente, assegna in maniera univoca un colore specificato nel range. Nella Figura 4.7 viene riportata l'implementazione di tale funzione.

```
const colors = [
    {
        colorArray: ["#6daed5","#4b97c9","#2f7ebc",
            "#1864aa", "#0a4a90","#08306b"
        ],
        root: "#08306b"
    },
    ...
]
...
this.color = d3.scale.quantize()
    .domain([0, maxBlockScore])
    .range(colors[index]);
...
```

Listing 4.7: Implementazione della funzione di colorazione del SunBurst

Capitolo 5

Valutazione

Il capitolo si compone di due parti: una relativa al test incentrato sul SunBurst come metodo di visualizzazione dei risultati di ricerca e per la visualizzazione/navigazione dei frammenti di un singolo documento e l'altra relativa al test dell'algoritmo di ricerca ibrido che combina la Sparse Vector Representation (BM25) con la Dense Vector Representation (SBERT).
Il test relativo all'interfaccia è stato svolto in un laboratorio dell'Università di Bologna da utenti che non avevano mai interagito prima con l'applicazione proposta. Gli utenti coinvolti sono studenti e/o dottorandi con background sia tecnico sia umanistico. Si precisa che per lo svolgimento di questo test è stato utilizzato il solo algoritmo BM25 e i motivi di questa scelta sono molteplici:

- il focus del test era sul SunBurst e non sull'algoritmo di ricerca, pertanto, è stato utilizzato l'approccio discusso nella Sezione 1.3.1 del Capitolo 1
- l'utilizzo di un algoritmo di ricerca ibrido avrebbe reso difficile, l'eventuale, confronto con altre applicazioni che non implementano un motore di ricerca ibrido
- lo stato dell'arte sull'implementazione di un algoritmo di ricerca ibrido è recente e quindi non esiste un approccio standardizzato
- infine, in caso di problematiche avrebbe reso difficile distinguere tra problemi legati all'interfaccia piuttosto che al calcolo della rilevanza

Nella seconda sezione viene presentato il metodo con il quale l'algoritmo di ricerca è stato valutato rispetto a una versione classica basata su BM25.

5.1 Interfaccia: Test con utenti

Per valutare il SunBurst come tecnica alternativa per la visualizzazione dei risultati di ricerca e per la visualizzazione/navigazione dei frammenti di un singolo documento, l'applicazione è stata sottoposta ad una sessione di test con utenti. Durante la definizione del test è stata valutata la possibilità di utilizzare un'applicazione per la ricerca classica già implementata oppure implementare una versione semplificata che conservi le principali funzionalità di DocuDipity ma che presenti i risultati di ricerca come semplice lista. Dato che confrontare DocuDipity con un'altra applicazione che non implementa le stesse feature sembrava meno efficace ai fini del test è stato scelto di implementare l'applicazione "*Simple DocuDipity*". Il test è suddiviso in due fasi, in cui si utilizzano le seguenti applicazioni:

1. simple-docudipity: versione semplificata di docudipity, che mette a disposizione le medesime funzionalità tranne il sunburst per la navigazione del documento e per la visualizzazione dei risultati di ricerca che vengono presentati come semplice lista (conservando però la colorazione della rilevanza), Figura 5.1

2. DocuDipity: applicazione precedentemente descritta, che propone la tecnica del SunBurst per la visualizzazione dei risultati di ricerca e per la visualizzazione/navigazione dei frammenti di un singolo documento (affiancata alla vista ipertestuale), come descritto nella Sezione 3.3.

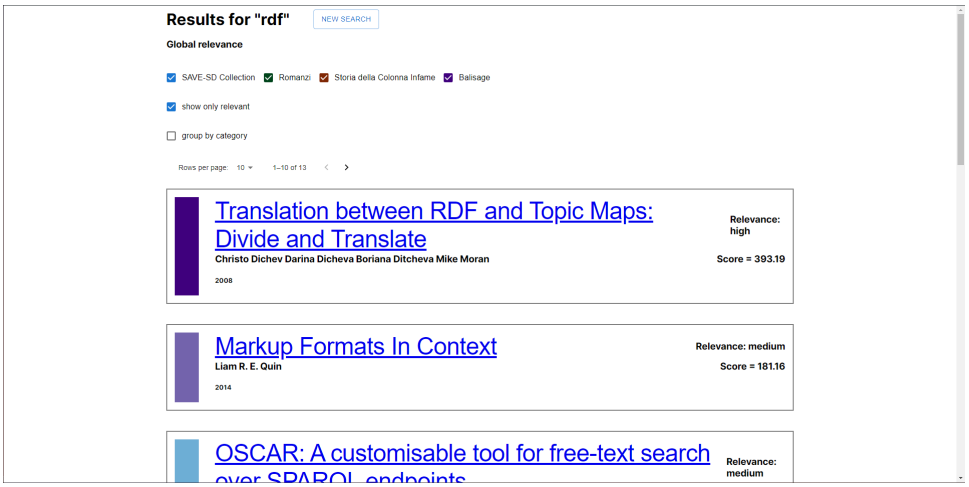


Figura 5.1: Overview dell'applicazione Simple Docudipity, quando si seleziona un elemento della lista viene visualizzato il documento con la vista ipertestuale

La predetta divisione è stata ritenuta necessaria affinché gli utenti eseguissero uno stesso task sulle due interfacce e avere, quindi, un confronto sia quantitativo (il tempo

impiegato per svolgere lo stesso task sulle due applicazioni) sia qualitativo incentrato sul feedback dell'utente e del suo grado di soddisfazione per lo svolgimento del task sulle due interfacce (con particolare attenzione al confronto dei due approcci).
Il test è stato svolto in collaborazione con la casa editrice "*Il Mulino*" che ha messo a disposizione 506 testi riguardanti le categorie "*Economia*", "*Scienza Politica*" e "*Sociologia*".
I task sono incentrati sulla ricerca di una parola chiave e successivamente di effettuare un'analisi o sulla distribuzione dei risultati di ricerca o di effettuare una specifica ricerca all'interno dei documenti, nella Tabella 5.1 vengono riportati alcuni task usati nel test. La prima parte del test è stata eseguita su "*Simple-Docudipity*" e la seconda parte su

Task	Testo
Analisi sulla distribuzione dei risultati di ricerca	Quale tra i due argomenti "*Capitalismo*" e "*Comunismo*" è trattato da più testi?
	Considerando il solo argomento "*Comunismo*", in quale disciplina tale argomento è trattato da più testi?
Ricerca all'interno di un documento	Indica il titolo di tre libri della categoria "*Economia*" che si occupano (anche in maniera marginale) di "*febbre*" secondo l'accezione medica (malattia). Ti chiediamo quindi di escludere i testi in cui la parola è usata solo con altre accezioni.
	Indica il titolo della sezione più rilevante sul tema "*Apple*" nei libri della categoria "*Economia*".

Tabella 5.1: Testo di alcuni task presenti nel test

"*Docudipity*", entrambe composte da tre task. Il primo task si compone di quattro domande sull'analisi della distribuzione dei risultati di ricerca, mentre il secondo e terzo task sono composti da una sola domanda incentrata su una ricerca all'interno di un documento. Nello specifico, si richiede di riportare o la sezione maggiormente rilevante per la ricerca o i libri che trattano un determinato tema escludendo i possibili falsi positivi (ad esempio per la ricerca della parola chiave "*Ferrari*", intesa come casa automobilistica, escludere i possibili riferimenti ad un autore di cognome Ferrari). Inoltre, gli utenti sono stati divisi in due gruppi ("*Gruppo A*" e "*Gruppo B*") in cui le domande per le due fasi del test erano scambiate, ovvero, le domande della prima parte per il "*Gruppo A*" corrispondono a quelle della seconda parte del "*Gruppo B*" e lo stesso per la seconda parte del test. Questo ha permesso di verificare l'indipendenza dei risultati ottenuti dai due gruppi dalle domande poste, evitando che i risultati potessero essere influenzati da una

domanda piuttosto che dall'applicazione utilizzata. Nella Tabella 5.2 vengono riportate le domande per ciascun gruppo.

Gruppo A	*Gruppo B*
Quale tra i due argomenti "*Capitalismo*" e "*Comunismo*" è trattato da più testi?	Quale tra i due argomenti "*Nazismo*" e "*Fascismo*" è trattato da più testi?
Considerando il solo argomento "*Comunismo*", in quale disciplina tale argomento è trattato da più testi?	Considerando il solo argomento "*nazismo*", in quale disciplina tale argomento è trattato da più testi?
Considerando ancora il solo argomento "*Comunismo*", in quale disciplina tale argomento è trattato da meno testi?	Considerando ancora il solo argomento "*nazismo*", in quale disciplina tale argomento è trattato da meno testi?
Fai un confronto quantitativo sul numero e la rilevanza dei libri che trattano i temi "*Comunismo*" e "*Capitalismo*" nelle diverse discipline. Evidenzieresti qualche trend significativo e/o aggiungeresti qualche osservazione su questi dati?	Fai un confronto quantitativo sul numero e la rilevanza dei libri che trattano i temi "*nazismo*" e "*Fascismo*" nelle diverse discipline. Evidenzieresti qualche trend significativo?
Indica il titolo di tre libri della categoria "*Economia*" che si occupano (anche in maniera marginale) della casa automobilistica Ferrari. Ti chiediamo quindi di escludere i testi in cui la parola è usata solo con altre accezioni.	Indica il titolo di tre libri della categoria "*Economia*" che si occupano (anche in maniera marginale) di "*febbre*" secondo l'accezione medica (malattia). Ti chiediamo quindi di escludere i testi in cui la parola è usata solo con altre accezioni (es. come passione viva, desiderio impaziente, ecc.).
Indica il titolo della sezione più rilevante sul tema "*Apple*" nei libri della categoria "*Economia*". Nota: per gli scopi di questo test la sezione più rilevante è quella che contiene il maggior numero di occorrenze della parola chiave indicata.	Indica il titolo della sezione più rilevante sul tema "*Renault*" nei libri della categoria "*Economia*". Nota: per gli scopi di questo test la sezione più rilevante è quella che contiene il maggior numero di occorrenze della parola chiave indicata.

Tabella 5.2: Elenco delle domande per ciascun gruppo

Gli utenti che hanno svolto il test sono venti, raggruppati secondo i seguenti background:

- Laurea Triennale (x10)
- Laurea Magistrale (x7)

- Dottorando
- Laurea ciclo unico (x2)

Il background degli utenti comprende sia percorsi di laurea umanistici sia corsi di laurea STEM, questo permette di analizzare il comportamento dell'applicazione su numerosi tipi di utenti che possono contribuire a far emergere diversi problemi od osservazioni in base al loro background. Nello specifico, sei utenti hanno conseguito una laurea STEM, mentre quattordici utenti hanno come background una laurea umanistica.
L'applicazione viene valutata secondo tre metriche:

1. "*effectiveness*" (correttezza): data dalle corrette risposte alle domande dei task
2. "*efficiency*" (efficienza): data dal tempo che l'utente impiega per svolgere il task
3. "*satisfaction*" (soddisfazione): dato dai commenti degli utenti e dal punteggio del SUS (System Usability Scale)

Il test è stato realizzato utilizzando LimeSurvey [29], uno strumento che permette di creare sondaggi con diverse tipologie di domande e mette a disposizione un'interfaccia grafica per la realizzazione/gestione dei sondaggi creati. In generale, ogni sondaggio è composto da uno o più gruppi di domande in cui è possibile raccogliere numerose informazioni per ciascuna fase del sondaggio, come ad esempio il tempo che un utente impiega per rispondere a una domanda o il "*timestamp*" di inizio e completamento di una domanda o dell'intero sondaggio. La possibilità di avere il tempo impiegato dall'utente per le varie fasi del sondaggio ha avuto un forte peso sulla scelta di questo strumento, in quanto permette in maniera agevole di misurare la metrica dell'efficienza senza cronometrare a mano il tempo che un utente impiega a rispondere ad un quesito.

Correttezza (effectiveness)

Come mostrato nella Tabella 5.3, la percentuale di risposte corrette per ogni domanda risulta particolarmente elevata ($\geq 85.00\%$) per entrambe le applicazioni, tuttavia, il task 3 sembra andare in controtendenza. Esaminando le risposte il motivo consiste nella terminologia utilizzata dalla domanda, in quanto non tutti gli utenti hanno compreso il significato della frase "*titolo della sezione*" ma una volta chiarito, la percentuale di risposte corrette nell'applicazione successiva si riallinea a quello degli altri task. Prendendo in considerazione solamente la metrica della correttezza si potrebbe concludere che entrambe le applicazioni siano ugualmente adatte a svolgere questi tipi di task. Tuttavia, considerando il feedback degli utenti emerge che, per aiutarsi nella navigazione del documento, hanno utilizzato la funzione di ricerca messa a disposizione dal browser con relativa heatmap ma questo non è stato sufficiente.

	Task 1			Task 2	Task 3
	Domanda 1	Domanda 2	Domanda 3	Domanda	Domanda
Simple Docudipity	100%	100%	100%	85.00%	70.00%
Docudipity	100%	100%	100%	90.00%	85.00%

Tabella 5.3: Percentuale di risposte corrette per ogni domanda dei task

Efficienza (efficiency)

Considerando la metrica dell'efficienza (Tabella 5.4), risulta evidente che mettendo a disposizione degli utenti il SunBurst (Docudipity) aiuta gli stessi ad essere maggiormente efficienti nello svolgimento del task.

	Task 1	Task 2	Task 3
Simple Docudipity	158.76s	293.60s	173.56s
Docudipity	83.88s	208.60s	117.48s

Tabella 5.4: Tempo medio di esecuzione di ogni task

Analizzando in maggior dettaglio i tempi di esecuzione e prendendo in considerazione il background dell'utente emerge che gli utenti con background STEM sono stati più rapidi rispetto agli utenti con background umanistico, come mostrato nella Tabella 5.5. Infatti, si ottiene che per ogni task un utente con background umanistico ha impiegato in media 70.44s in più rispetto ad un utente con background STEM. Prendendo in considerazione i dati sulla varianza per i due background, risulta evidente che per gli umanisti i tempi medi sono stati molto diversi tra di loro in quanto la varianza risulta, per la maggior parte dei task, un ordine di grandezza maggiore rispetto agli utenti con background STEM. Questo significa che alcuni utenti umanistici durante l'esecuzione del test hanno riscontrati grandi difficoltà che gli hanno impedito di essere maggiormente efficienti, tuttavia si sottolinea che questo vale solamente per qualche caso specifico in quanto se fosse un problema comune a tutti la varianza dovrebbe essere nettamente minore o vicina a zero. Per analizzare e capire gli ostacoli o problemi che hanno riscontrato questi utenti è necessario prendere in considerazione il feedback degli stessi, che verranno discussi in seguito. Comunque, rimane il fatto che anche per gli utenti con background umanistico l'introduzione del SunBusrt, per la visualizzazione dei risultati di ricerca e navigazione del documento, ha portato a risultati nettamente migliori in quanto sia il tempo medio che la varianza per la seconda parte risultano nettamente inferiori rispetto alla prima.

	Simple Docudipity			Docudipity		
Background	Task 1/SD	Task 2/SD	Task 3/SD	Task 1/D	Task 2/D	Task 3/D
STEM	81.20s	234.01s	159.05s	43.64s	147.64s	72.54s
Umanistico	192.01s	319.14s	179.78s	101.13s	234.69s	134.04s
STEM Var	5625.17	3933.61	4600.68	1530.21	4082.50	287.39
Uman. Var	70 851.06	13 918.44	10 892.74	17 532.20	10 458.90	4380.94

Tabella 5.5: Tempo medio di esecuzione di ogni task per background d'utente e varianza, dove la lettera a destra del numero rappresenta l'applicazione su cui il task è stato svolto

Soddisfazione (satisfaction)

Non solo Docudipity ha mostrato una maggiore efficienza nello svolgimento dei task ma anche una maggiore semplicità d'uso che emerge chiaramente dal confronto della System Usability Scale (SUS). Il System Usability Scale è stato definito da John Brooke nel 1996 [9] come un semplice strumento per catturare l'usabilità percepita di un sistema. Tale strumento è composto da dieci quesiti le cui risposte sono rappresentate da una scala Likert [28] di 5 elementi. Lo score è un punteggio nel range 0-100 che, oltre a rappresentare l'usabilità del sistema, permette il confronto con altri sistemi/applicazioni. Secondo lo studio condotto da Jeff Sauro e James R. Lewis [50], utilizzando oltre 446 studi presenti in letteratura e 5000 risposte ai questionari SUS, è emerso che il punteggio medio ottenuto è 68. Tale valore può essere considerato come benchmark in quanto la metà delle applicazioni ottengono un punteggio inferiore a questo. Nella Tabella 5.6 vengono riportati i punteggi ottenuti dalle due applicazioni utilizzate per il test.

	SUS (System Usability Scale)
Simple Docudipity	71.12%
Docudipity	76.13%

Tabella 5.6: Media del punteggio del System Usability Scale (SUS) ottenuto dalle due applicazioni

Per analizzare i feedback degli utenti, a ciascun commento è stata assegnata una lista di keyword che dipendono dalle feature discusse dall'utente. Successivamente, per ogni keyword viene calcolata la frequenza nei commenti e analizzato l'utilizzo, con particolare attenzione se viene sottolineata una criticità nella feature o se, invece, viene espresso un parere positivo sulla stessa. Infine, le keyword che non compaiono almeno due volte sono state filtrate e nella Figura 5.2 è riportato il risultato di questa analisi. Ad esempio, considerando la keyword "*SunBurst*", tredici utenti hanno evidenziato in maniera positiva l'utilizzo del SunBusrt come tecnica per visualizzare i risultati di ricerca e per navigare i documenti. Esaminando i risultati emerge che i principali problemi riscontrati dagli utenti riguardano: il primo utilizzo dell'applicazione, la mancanza dei filtri dinamici e

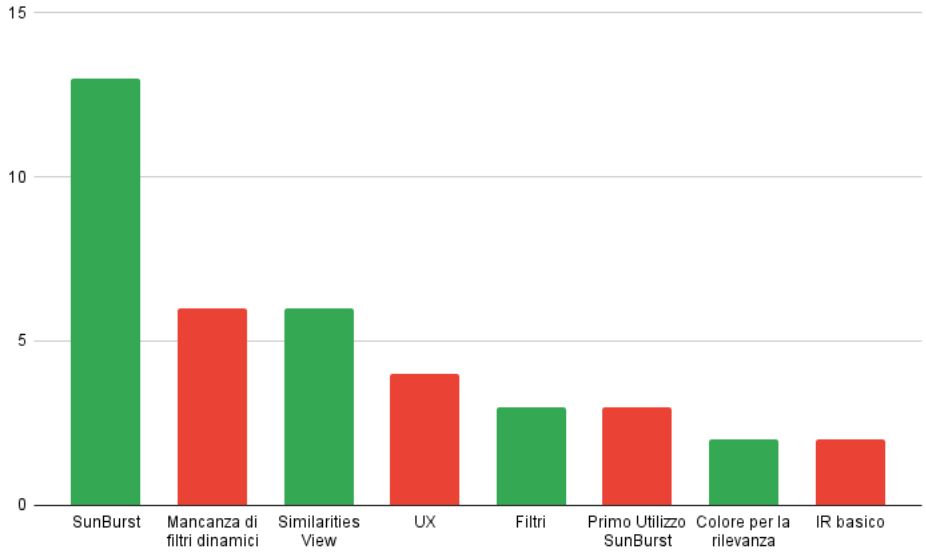


Figura 5.2: Overview feedback utenti, i commenti positivi sono riportati con il colore verde e rosso per i negativi

di alcune scelte sull'interfaccia grafica che ha degradato la "*User Experience*" di alcuni utenti. Approfondendo, il background degli utenti che hanno segnalato questi problemi è emerso che sono principalmente utenti con background umanistico e questo spiegherebbe la differenza di esecuzione dei task tra utenti STEM e umanistici. La spiegazione potrebbe consistere nel fatto che gli utenti con background umanistico risentono maggiormente della mancanza di un tutorial iniziale o la posizione non ottimale di alcune feature dell'applicazione in quanto si aspettano di essere maggiormente guidati, mentre per gli utenti con background tecnico sono portati a essere maggiormente indipendenti ed a sperimentare per trovare la soluzione senza un aiuto esterno. Invece, il sunburst e la similarity view sono state apprezzate da entrambi i tipi di utenti e non sono presenti commenti che fanno emergere particolari criticità nel loro utilizzo ed implementazione.
Dopo aver discusso ad alto livello dei feedback ricevuti, in questa ultima parte verranno presentati ed approfonditi i commenti ritenuti maggiormente interessanti.
I feedback sulla visualizzazione SunBurst proposta, sono stati soprattutto positivi:

> La visualizzazione rende immediata la risposta a domande quantitative e il ritrovamento di sezioni utili, inoltre offre uno sguardo d'insieme senza la necessità di scorrere molte pagine.

> Sicuramente la navigazione tramite il grafico e la possibilità di navigare agevolmente tra le sezioni, per esempio passando dalla sezione Conclusioni all'introduzione cliccando sul grafico specifico del libro.

Si precisa che non sono presenti commenti che sconsigliano e/o si lamentano della visualizzazione proposta ma piuttosto sono incentrati su un problema del primo utilizzo, come mostrato dal seguente commento

> Ho notato come il SunBurst diversamente dall'indice dei risultati accelera notevolmente le task date, sia con task generiche che più specifiche (ricorrenza locale di un termine per paragrafo). Penso sia riutilizzabile in diversi contesti e anche se l'ho trovata inizialmente poco intuitiva, una volta ricevuto istruzioni su come leggerla, è stato semplicissimo e quasi divertente utilizzarla.

che, tuttavia, può essere facilmente risolto con un tutorial al primo utilizzo dell'applicazione. Oppure legate a piccoli accorgimenti per migliorare l'usabilità dell'applicazione

> Mancanza di possibilità di aggiornare la stessa visualizzazione in maniera dinamica una volta effettuata la ricerca e l'assenza di percentuali visualizzate per la distribuzione dei documenti quando si effettua una ricerca su tutte e tre le categorie.

Infine, anche gli utenti che preferiscono o che si sentono soddisfatti di uno strumento di navigazione maggiormente tradizionale hanno espresso un parere positivo sul possibile utilizzo del SunBurst come complemento delle tecniche tradizionali. Come viene mostrato nel seguente commento:

> Sicuramente le funzionalità offerte dalla vista sunburst e dal menu radiale, nonostante una vista del genere possa risultare di non immediata comprensione. Ho trovato comoda la navigazione tra le occorrenze tramite doppio clic, notando come velocizza il lavoro di ricerca all'interno di un documento. Al tempo stesso mi sarei sentito a mio agio anche utilizzando uno strumento di navigazione più tradizionale, e ritengo che potrebbe essere utile implementarlo come complemento.

5.2 Motore di ricerca ibrido: valutazione ranking

Prima di procedere con la definizione del test è necessario stabilire quali sono le metriche da utilizzare per confrontare i risultati e secondo gli autori ChengXiang Zhai e Sean Massung [60], per i sistemi di "*Information Retrieval*", ne esistono principalmente tre:

- efficacia o accuratezza: Quanto sono accurati i risultati della ricerca? Questa metrica misura la capacità del sistema di inserire prima le risorse maggiormente rilevanti, rispetto a quelle meno o non rilevanti
- efficienza: misura il tempo richiesto per il sistema di restituire i risultati all'utente e le risorse richieste per processare la query
- usabilità: permette di determinare l'esperienza che l'utente ha avuto utilizzando l'interfaccia e le features del sistema. Tuttavia, tale misura rientra maggiormente nel campo della "*User Experience*" piuttosto che nel campo della "*Information Retrieval*"

Da notare che queste metriche assomigliano a quelle utilizzate nella Sezione 5.1, tuttavia in questo caso vengono utilizzate con un'accezione diversa rispetto al contesto precedente di usabilità. Dato che lo scopo del test è quello di confrontare il ranking dei documenti utilizzando due sistemi di "*Information Retrieval*", uno classico e uno ibrido, la misura fondamentale da utilizzare è l'efficacia o accuratezza. Per l'usabilità, il test dell'interfaccia è stato esposto nella Sezione 5.1 e la metrica dell'efficienza non risulta interessante ai fini di questo test.
La misura dell'efficacia o accuratezza si basa sul concetto dei documenti rilevanti o non rilevanti, dove il concetto di rilevanza non è legato alla query ma alle esigenze informative dell'utente. Pertanto, un risultato è rilevante se e solo se soddisfa l'esigenza informativa formulata nella query dall'utente. Questa separazione viene spesso indicata come "*Gold Standard*" o "*Ground Truth*" del giudizio di rilevanza.
In generale, ogni test di un sistema di "*Information Retrieval*" è composto da tre elementi principali [35]:

1. una collezione di documenti, ovvero l'insieme dei documenti che il sistema dovrà in qualche modo ordinare per soddisfare il bisogno informativo di un utente
2. un insieme di query, ovvero il bisogno informativo dell'utente espresso come una frase o una parola di ricerca
3. una mappa di giudizi, ovvero data la coppia query-documento viene espresso un giudizio sul suo grado di rilevanza

Lo scopo del test è quello di avere una prima valutazione del motore di ricerca implementato e non ha la pretesa di ottenere un risultato definitivo, ma piuttosto una valutazione preliminare in attesa di procedere con un test maggiormente esaustivo. In particolare, un test di valutazione di un motore di ricerca ibrido esaustivo è stato discusso da Priyanka Mandikal e Raymond Mooney [33], in cui viene utilizzato il "*Cystic Fibrosis Database*" (CF) [52] composto da 1239 documenti pubblicati dal 1974 al 1979 ed un insieme di 100 query con i rispettivi documenti rilevanti. Sfruttando il "*Golden Standard*" del dataset, sono stati confrontati i risultati delle varie configurazioni per determinare quella maggiormente efficace/accurata.
Al fine di questo test, come collezione di documenti sono stati scelti un insieme di articoli scientifici, esposti alle conferenze Balisage[1] e SAVE-SD[2], in quanto devono contenere un "*Abstract*" in cui viene presentato ad alto livello il contenuto dell'articolo, che può essere utilizzato per presentare all'utente una breve descrizione permettendogli di esprimere un giudizio di rilevanza senza la lettura dell'intero articolo. L'insieme delle query è stato definito durante la preparazione del test, in quanto se l'utente fosse stato libero di scegliere una query non sarebbe stato possibile confrontare in maniera significativa il ranking dei due motori di ricerca. Il predetto dataset non contiene un "*Golden Standard*" per le query, pertanto, la valutazione si basa principalmente su una metrica incentrata sulla valutazione dell'ordine di presentazione dei risultati di ricerca (DCG), metrica che verrà ampiamente approfondita in seguito. Infine, la mappa dei giudizi è costruita dagli utenti compilando il questionario realizzato con Google Form [21], strutturato come segue:

- introduzione, dove si spiega lo scopo del sondaggio, la sua struttura e la stima del tempo per completarlo
- background, dove viene presentata la query con una breve descrizione sull'argomento in caso l'utente non ne fosse a conoscenza
- articoli, dato il background l'utente deve decidere in maniera binaria se l'articolo dato è rilevante per la ricerca oppure no
  - non rilevante: è presentato l'articolo successivo
  - rilevante: viene chiesto all'utente di assegnare uno score di rilevanza tra 1 e 5

Per questo test sono state selezionate due query, "*XQuery*" e "*RDF*", ed un totale di sette articoli che rappresentano l'unione dei primi tre risultati di ricerca ottenuti utilizzando il motore di ricerca classico (BM25) ed ibrido (BM25 + SBERT). La configurazione del motore di ricerca ibrido è la seguente:

[1] `https://www.balisage.net/index.html`
[2] `https://save-sd.github.io/`

- modello (SBERT): all-mpnet-base-v2, basato sul modello pre-addestrato microsoft/mpnet-base. Questo modello permette di ottenere l'encoding di frasi e paragrafi, catturando le informazioni semantiche contenute nel testo originale. L'output è un vettore che può essere utilizzato per task di "*Sentence Similarity*" o "*Information Retrieval*"

- l'*alpha* è stato assegnato il valore 0.80 [33]

In Figura 5.3 sono riportate le principali metriche utilizzate per la valutazione di un sistema di "*Information Retrieval*" discusse da Christopher D. Manning [35].

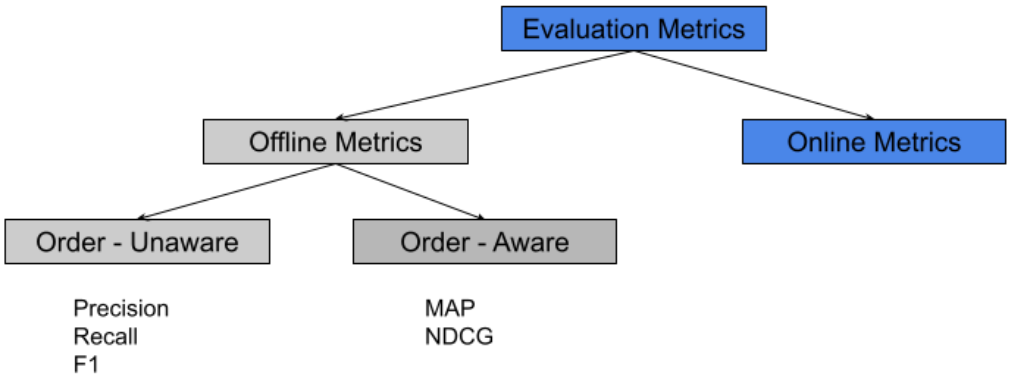


Figura 5.3: Overview delle metriche di valutazione di un sistema di "*Information Retrieval*"

Una prima divisione consiste tra metriche "*Online*" e "*Offline*", in cui le prime sono create da sessioni reali di utenti con il sistema e che si basano sui log ottenuti con l'iterazione dello stesso. Alcuni esempi possono essere il "*Session Success Rate*", il quale misura il rapporto delle sessioni che terminano con un successo (bisogno informativo soddisfatto), oppure il "*Zero Result Rate*" che misura il rapporto delle SERP (Search Engine Results Pages) che ritornano con zero risultati. Invece, le metriche "*Offline*" vengono create da sessioni in cui si richiede ad un gruppo di utenti di assegnare un punteggio di rilevanza ai risultati di una ricerca. Il punteggio di rilevanza può essere sia binario (risultato rilevante o non rilevante) sia multi-livello (per esempio assegnare uno score di rilevanza da 0 a 5), per ciascun documento di una risposta ad una query. Considerando le metriche di valutazione "*Offline*" è possibile dividerle ulteriormente in:

- order unaware: ovvero, i documenti restituiti non sono ordinati per un punteggio di rilevanza. In questo caso. le misure considerano solamente che nella risposta

ci siano documenti rilevanti ma non tengono conto del possibile ordinamento degli elementi. Le misure maggiormente utilizzate sono: "*precision*", "*recall*" e "*F1*"

- order aware: dato l'espandersi del volume dei dati e delle informazioni, data una query d'utente, difficilmente è possibile restituire tutti i documenti/elementi risultati rilevanti. Per tale motivo, i motori di ricerca applicano funzioni di "*ranking*", le quali ordinano i documenti/elementi risultati rilevanti e li restituiscono all'utente. In questo caso non è più sufficiente utilizzare le misure per valutare la presenza di documenti/elementi rilevanti ma devono anche tenere conto dell'ordinamento degli stessi. Un ottimo sistema di IR posiziona gli elementi/documenti maggiormente rilevanti in cima e in basso quelli marginalmente rilevanti. Rientrano in questa categoria le misure "*Mean Average Precision (MAP)*" e "*Normalized Discounted Cumulative Gain (NDCG)*"

Iniziamo con l'approfondire le principali misure utilizzate per la valutazione di un sistema di "*Information Retrieval*" che *non* tengono conto dell'ordinamento dei risultati di ricerca, ossia la "*precision*", la "*recall*" e "*F1*".
La "*precision*" viene definita come il rapporto tra gli elementi restituiti, dal sistema, risultati rilevanti ed il numero totale di elementi restituiti.

$$Precision = \frac{\#relevant\_items\_retrieved}{\#retrieved\_items} \tag{5.1}$$

Invece, la "*recall*" viene definita come il rapporto tra gli elementi restituiti che sono risultati rilevanti ed il numero di elementi che erano effettivamente rilevanti per la query (sull'intera collezione di documenti/elementi).

$$Recall = \frac{\#relevant\_items\_retrieved}{\#relevant\_items} \tag{5.2}$$

Infine, la "*F1*" combina insieme la "*precision*" e la "*recall*" attraverso la loro media armonica.

$$F1 = \frac{2 * Precision * Recall}{Precision + Recall} \tag{5.3}$$

L'utilizzo della media armonica, invece che l'utilizzo della semplice media aritmetica, è giustificata dal fatto che per ottenere il 100% di "*recall*" è sufficiente ritornare l'intera collezione di dati, che con l'utilizzo della media aritmetica si ottiene sempre almeno il 50%. Il vantaggio dell'utilizzo della media armonica consiste nel fatto che è sempre minore o uguale della media aritmetica. Inoltre, quando due numeri sono molto distanti la media armonica è maggiormente vicina al loro minimo rispetto alla semplice media aritmetica.
Le misure presentate fino ad ora considerano solamente la distinzione tra documenti rilevanti e non, ma non considerano l'ordinamento dei risultati di ricerca, in quanto un

buon sistema di "*Information Retrieval*" è in grado di ordinare i risultati per mettere i più rilevanti in cima e i meno rilevanti in fondo. Per tenere conto di questo fatto è stata definita la seguente formula:

$$AP = \frac{\sum_{k=1}^{n} P(k) \times rel(k)}{\#relevant\_items} \tag{5.4}$$

dove $rel(k)$ è la funzione indicatore che ritorna 1 se l'elemento alla posizione K è rilevante, 0 altrimenti. $P(k)$ rappresenta la "*precision at k*", che ritorna il numero di elementi rilevanti nelle prime K posizioni. Tuttavia, tale misura è applicata ad una singola query che non risulta esaustiva per la valutazione di un sistema di "*Information Retrieval*" oppure un suo confronto con un altro sistema. Per tale motivo si preferisce utilizzare una misura che tenga conto di tutte le query ovvero la "*Mean Average Precision*" definita come segue

$$MAP = \frac{\sum_{q=1}^{\#Q} AP(q) \times rel(k)}{\#Q} \tag{5.5}$$

dove Q rappresenta l'insieme delle query e la funzione AP è l'*Average Precision* definita nell'Equazione 5.4.
La principale limitazione della misura MAP è la definizione della funzione $rel(k)$, la quale considera i risultati binari distinguendo tra documenti rilevanti e non rilevanti, escludendo un giudizio di rilevanza più raffinato tra documenti maggiormente/minormente rilevanti con un giudizio non binario.
Per superare questa limitazione è stata definita la metrica "*cumulative gain*", definita come segue:

$$CG = \sum_{i=1}^{k} rel_i \tag{5.6}$$

L'Equazione 5.6 modifica la definizione della funzione binaria rel permettendole di restituire un grado di rilevanza di un i-esimo documento nel range $[0, 5]$, quindi la formula esprime il giudizio di rilevanza come sommatoria di rilevanza di ciascun elemento dei risultati di ricerca. Tuttavia, questa misura presenta una limitazione poiché non considera l'esatta posizione del risultato. Si consideri il seguente esempio, assumiamo di avere due sistemi di "*Information Retrieval*" che per una certa query restituiscono gli stessi documenti ma ordinati in maniera differente. Il primo sistema restituisce $CG = (5 + 4 + 5 + 2 + 1) = 17$ mentre il secondo $CG = (1 + 2 + 4 + 5 + 5) = 17$, il punteggio della formula CG è lo stesso ma prendendo in considerazione l'ordinamento i due sistemi non sono equivalenti. Infatti, in un sistema di "*Information Retrieval*" l'ordine dei risultati è fondamentale in quanto ci si aspetta che i documenti maggiormente rilevanti siano presentati *prima* di quelli meno rilevanti.
Per superare tale limitazione è stata definita una nuova misura denominata "*Discounted*

Cumulative Gain" definita nell'Equazione 5.7.

$$DCG = \sum_{i=1}^{k} \frac{rel_i}{\log_2(i+1)} \tag{5.7}$$

Questa nuova formula introduce una funzione logaritmica per ridurre il punteggio di rilevanza con il progredire delle posizioni considerate. Riprendendo il precedente esempio, con questa nuova metrica otteniamo: $DCG = (\frac{5}{\log_2 2} + \frac{4}{\log_2 3} + \frac{5}{\log_2 4} + \frac{2}{\log_2 5} + \frac{1}{\log_2 6}) = 11.27$ e $DCG = (\frac{1}{\log_2 2} + \frac{2}{\log_2 3} + \frac{4}{\log_2 4} + \frac{5}{\log_2 5} + \frac{5}{\log_2 6}) = 8.35$.
Sfortunatamente, una limitazione della metrica *DCG* consiste nella difficoltà di interpretazione dello score finale, in quanto viene fortemente influenzato dal range utilizzato dalla funzione *rel*. Una soluzione consiste nell'utilizzare una versione normalizzata del punteggio *DCG*, ovvero la "*Normalized Discounted Cumulative Gain*" (NDCG). Tale normalizzazione prevede l'utilizzo della *Ideal DCG*, che assume l'ordinamento ideale ovvero i risultati di ricerca ordinati per la loro rilevanza per la query. L'Equazione 5.8 riporta la formula *NDCG*.

$$NDCG = \frac{DCG}{IDCG} \tag{5.8}$$

Le formule utilizzate per la valutazione e confronto dei due motori di ricerca sono le seguenti:

- "*Precision*", utilizzata per analizzare il rapporto tra gli elementi che sono stati identificati come rilevanti dai tester e gli elementi restituiti dai sistemi di "*Information Retrieval*". Tale misura viene calcolata per ogni query presente nel test
- "*Discounted Cumulative Gain*", metrica utilizzata per confrontare i due ordinamenti e in particolar modo analizzare l'ordinamento dei documenti/articoli. I documenti che i tester valutano come rilevanti e assegnano un punteggio di rilevanza elevato devono essere presentati prima di quelli segnati come non rilevanti o marginalmente rilevanti

Nella Tabella 5.7 sono confrontati i risultati del motore di ricerca classico con quello ibrido, utilizzando le metriche esposte. Il questionario è stato compilato da dieci persone tra studenti di Informatica e dottorandi. Analizzando i risultati ottenuti, in particolare prendendo in considerazione la metrica "*Precision*", risulta evidente che per la prima query i due motori di ricerca hanno ottenuto lo stesso punteggio (in quanto i documenti che hanno restituito sono stati considerati ugualmente rilevanti dai tester) ma per la seconda query il motore di ricerca classico ha ottenuto un punteggio nettamente superiore. La causa di questo crollo consiste nel fatto che il motore di ricerca ibrido ha selezionato per la query "RDF" un articolo in cui si faceva riferimento a "*knowledge graphs*", che non è direttamente collegata alla parola chiave della query ma la cui rappresentazione viene fatta in "RDF". Infatti, analizzando le risposte dei tester, gli unici che hanno segnato

	BM25		BM25 + SBERT	
	Query 1	Query 2	Query 1	Query 2
Precision	93%	83%	93%	63%
DCG	7.51	6.86	6.85	4.51

Tabella 5.7: Risultati del test di valutazione del motore di ricerca ibrido, dove "*Query 1*" corrisponde a *XQuery* e "*Query 2*" a *RDF*

tale documento come rilevante sono stati quelli che avevano un'elevata conoscenza del tema, mentre i tester meno pratici lo hanno segnato, semplicemente, come non rilevante. Questo è un dato molto interessante in quanto, come detto precedentemente, l'utilizzo di "SBERT" permette di superare il meccanismo basato su "*exact match*" ma risulta utile solamente per persone con elevata padronanza del tema, altrimenti, i risultati proposti possono sembrare inutili o irrilevanti.
Proseguendo l'analisi con la metrica "DSG", emerge che l'algoritmo di ricerca classico è più efficace nel restituire i documenti maggiormente rilevanti in cima, anche se, per la prima query la differenza risulta poco marcata, mentre per la seconda (anche a causa delle motivazioni già discusse) risulta elevata.
I primi risultati ottenuti sembrano indicare che l'approccio di un motore di ricerca basato sulla tecnica lessicale (Sparse Vector Representation) risulta leggermente migliore rispetto ad un approccio ibrido. Tuttavia, tale risultato potrebbe essere influenzato dall'utilizzo di un dataset limitato ad un dominio specifico e alla mancanza di un numero sufficiente di query per analizzare nel dettaglio il comportamento dei due motori di ricerca. Inoltre, sembra suggerire che l'approccio ibrido riesca a superare la limitazione dell'exact match, ma applicato ad un dominio specifico può essere utile solamente a utenti esperti del medesimo dominio, altrimenti la sensazione della qualità dei risultati diminuisce significativamente. Considerando questi risultati, sarebbe interessante procedere con un test di valutazione maggiormente strutturato per avere un confronto tra i due approcci in maniera più esaustiva.

Capitolo 6

Conclusioni

Il problema trattato in questo elaborato può essere analizzato da due prospettive:

1. interfaccia: utilizzare la visualizzazione SunBurst per presentare i risultati di ricerca all'utente, superando la classica vista come lista. Inoltre, viene proposto l'utilizzo del SunBurst sul singolo documento come strumento per la visualizzazione e navigazione dello stesso

2. motore di ricerca ibrido: approfondire il recente sviluppo di Large Language Models per applicarli al campo della Ricerca Semantica e sviluppare un sistema di Information Retrieval che combini quest'ultima tecnica con l'algoritmo BM25

Per quanto riguarda il primo obiettivo, approfondendo lo Stato dell'Arte, non sono emersi studi/pubblicazioni che proponessero la vista SunBurst per la visualizzazione dei risultati di ricerca e per la navigazione dei documenti. Successivamente, sfruttando l'applicazione DocuDipity sono state integrate le funzionalità relative alla ricerca di documenti, alla visualizzazione dei risultati di ricerca con la vista SunBurst e all'utilizzo del SunBurst per la navigazione dei risultati di ricerca nel documento (sfruttando lo score per evidenziare all'utente le parti del documento risultate rilevanti). L'implementazione del motore di ricerca, necessario per assegnare uno score ai frammenti di un documento data una query d'utente, è stato realizzato in modo da non limitarsi all'approccio classico, basato sull'exact match dei termini delle query, ma in modo da essere facilmente estendibile con altri approcci basati sui Large Language Model. L'integrazione è stata resa possibile grazie allo sviluppo di una classe base che definisce i metodi da implementare e all'utilizzo della funzione "*Weighted Sum Model*" per combinare lo score delle due tecniche.
Dal momento che sono stati definiti due obiettivi distinti, questa divisione si riflette anche nella modalità di valutazione: una sull'interfaccia e una sul motore di ricerca ibrido. Lo scopo della prima valutazione era quello di esaminare il comportamento del SunBurst con utenti che non avevano interagito con l'applicazione, per valutarne l'efficacia e l'usabilità, per cui è stato scelto di utilizzare un algoritmo di ricerca classico basato su BM25

(Sparse Vector Representation). Tale scelta viene rafforzata dal fatto che l'utilizzo del motore di ricerca ibrido avrebbe reso difficile distinguere i problemi legati all'interfaccia piuttosto che al metodo utilizzato per determinare i frammenti di documento rilevanti. I risultati ottenuti hanno evidenziato i numerosi vantaggi dell'utilizzo della tecnica del SunBurst, sia per utenti con background tecnico sia umanistico, legati all'efficienza (tempo impiegato per svolgere un task) e alla soddisfazione (feedback degli utenti e System Usability Scale). Sempre grazie ai risultati del test, è stato possibile evidenziare le principali criticità dell'applicazione, soprattutto legate alla User Experience e al primo utilizzo. Infatti, alcuni utenti hanno risentito della mancanza di un tutorial iniziale e al posizionamento di alcune feature che hanno degradato la loro esperienza, tuttavia, le criticità segnalate sono di facile soluzione attraverso l'implementazione di un tutorial iniziale e a qualche accorgimento sull'interfaccia per rendere più fluido l'utilizzo della stessa.
Il test riguardante il motore di ricerca ibrido, invece, è stato realizzato chiedendo ad un gruppo di utenti di valutare il grado di rilevanza di un insieme di paper per confrontare i ranking del motore di ricerca ibrido con quello classico attraverso le metriche di "*Precision*" e "*Discounted Cumulative Gain*" (DSG). I risultati hanno evidenziato che l'approccio classico risulta leggermente migliore rispetto a quello proposto ma, approfondendo i risultati ottenuti, emerge che per sfruttare al massimo la ricerca semantica è necessario avere un'elevata competenza di dominio, altrimenti, non sempre risulta chiaro il motivo della selezione di certi documenti che non sono immediatamente collegabili alla query. Tuttavia, questo potrebbe essere causato dall'utilizzo di un dataset non ottimale per la valutazione di un sistema di "*Information Retrieval*" o di query troppo specifiche, per tale ragione sarebbe interessante ripetere il test sul dataset "*Cystic Fibrosis Database*" (CF) e svolgere la valutazione realizzata da P. Mandikal e R. Mooney [52].
In conclusione, possiamo affermare che l'introduzione del SunBurst è stata estremamente positiva sia come strumento per la navigazione del documento sia come tecnica per la visualizzazione dei risultati di ricerca. In particolare, per quest'ultimo caso può essere utilizzata o per sostituire la visualizzazione dei risultati di ricerca come lista oppure come suo complemento.
La valutazione del motore di ricerca ibrido, invece, non permette di dare giudizi definitivi ma suggerisce che l'impiego di un test maggiormente approfondito non solo è necessario ma anche utile per analizzare nel dettaglio il comportamento dei due motori di ricerca.
Come sviluppi futuri, sicuramente, l'interfaccia dell'applicazione DocuDipity necessita di un aggiornamento, tenendo presente le problematiche emerse dal test, e valutare l'introduzione di un tutorial iniziale per presentare l'applicazione con le features proposte.
Infine, per quanto riguarda il motore di ricerca ibrido, come detto precedentemente, effettuare un test di valutazione approfondito sul dataset "*Cystic Fibrosis Database*" per confrontare i risultati dei due motori di ricerca in maniera precisa.

Bibliografia

[1] Rony Attar and Aviezri S Fraenkel. Local feedback in full-text retrieval systems. *Journal of the ACM (JACM)*, 24(3):397–417, 1977.

[2] Vimala Balakrishnan and Ethel Lloyd-Yemoh. Stemming and lemmatization: A comparison of retrieval performances. *Lecture Notes on Software Engineering*, 2:262–267, 01 2014.

[3] Benjamin B Bederson and Ben Shneiderman. *The craft of information visualization: readings and reflections*. Morgan Kaufmann, 2003.

[4] Stefan Behnel, Martijn Faassen, and Ian Bicking. lxml: Xml and html with python, 2005.

[5] Steven Bird, Ewan Klein, and Edward Loper. *Natural language processing with Python: analyzing text with the natural language toolkit*. "O'Reilly Media, Inc.", 2009.

[6] Lorenzo Bonetti and Paolo Torroni. Design and implementation of a realworld search engine based on okapi bm25 and sentencebert. 2021.

[7] Michael Bostock, Vadim Ogievetsky, and Jeffrey Heer. D3: Data-driven documents. *IEEE Trans. Visualization & Comp. Graphics (Proc. InfoVis)*, 2011.

[8] Gerlof Bouma. Normalized (pointwise) mutual information in collocation extraction. *Proceedings of GSCL*, 30:31–40, 2009.

[9] John Brooke et al. Sus-a quick and dirty usability scale. *Usability evaluation in industry*, 189(194):4–7, 1996.

[10] Tom Brown, Benjamin Mann, Nick Ryder, Melanie Subbiah, Jared D Kaplan, Prafulla Dhariwal, Arvind Neelakantan, Pranav Shyam, Girish Sastry, Amanda Askell, et al. Language models are few-shot learners. *Advances in neural information processing systems*, 33:1877–1901, 2020.

[11] Vannevar Bush et al. As we may think. *The atlantic monthly*, 176(1):101–108, 1945.

[12] Chen-Yuan Chen, Bih-Yaw Shih, Zih-Siang Chen, and Tsung-Hao Chen. The exploration of internet marketing strategy by search engine optimization: A critical review and comparison. *African Journal of Business Management*, 5(12):4644–4649, 2011.

[13] Edward Clarkson, Krishna Desai, and James Foley. Resultmaps: Visualization for search interfaces. *IEEE Transactions on Visualization and Computer Graphics*, 15(6):1057–1064, 2009.

[14] Royal Society (Great Britain). Scientific Information Conference. *Report and Papers Submitted*. Royal Society, 1948.

[15] Jacob Devlin, Ming-Wei Chang, Kenton Lee, and Kristina Toutanova. Bert: Pre-training of deep bidirectional transformers for language understanding. *arXiv preprint arXiv:1810.04805*, 2018.

[16] Angelo Di Iorio, Francesco Poggi, and Kevin Maiani. Progettazione e sviluppo di un ambiente di lettura e confronto tra articoli scientifici. 2021.

[17] Angelo Di Iorio, Francesco Poggi, and Stefano Notari. Progettazione e sviluppo di un'api rest per un ambiente di lettura e confronto tra articoli scientifici. 2021.

[18] Inc. Docker. Docker. `https://www.docker.com/`, 2021. Ultimo accesso: 28.05.2024.

[19] OpenJS Foundation. Express: Fast, unopinionated, minimalist web framework for Node.js. `http://expressjs.com/`, 2024. Ultimo accesso: 28.05.2024.

[20] OpenJS Foundation. Node.js. `https://nodejs.org/en`, 2024. Ultimo accesso: 28.05.2024.

[21] Google. Google Forms: Online Form Creator — Google Workspace. `https://www.google.com/forms/about/`, 2024. Ultimo accesso: 05-06-2024.

[22] Miguel Grinberg. *Flask web development: developing web applications with python*. "O'Reilly Media, Inc.", 2018.

[23] Matthew Honnibal and Ines Montani. spaCy 2: Natural language understanding with Bloom embeddings, convolutional neural networks and incremental parsing. To appear, 2017.

[24] Gord Hotchkiss, Steve Alston, and Greg Edwards. Eye tracking study. *Research white paper, Enquiro Search Solutions Inc*, 129, 2005.

[25] Anjali Ganesh Jivani et al. A comparative study of stemming algorithms. *Int. J. Comp. Tech. Appl*, 2(6):1930–1938, 2011.

[26] Jashanjot Kaur and P Kaur Buttar. A systematic review on stopword removal algorithms. *International Journal on Future Revolution in Computer Science & Communication Engineering*, 4(4):207–210, 2018.

[27] Divya Khyani, BS Siddhartha, NM Niveditha, and BM Divya. An interpretation of lemmatization and stemming in natural language processing. *Journal of University of Shanghai for Science and Technology*, 22(10):350–357, 2021.

[28] Rensis Likert. A technique for the measurement of attitudes. *Archives of psychology*, 1932.

[29] LimeSurvey Project Team / Carsten Schmitz. *LimeSurvey: An Open Source survey tool*. LimeSurvey Project, Hamburg, Germany, 2012.

[30] Aldo Lipani, Mihai Lupu, Allan Hanbury, and Akiko Aizawa. Verboseness fission for bm25 document length normalization. In *Proceedings of the 2015 International Conference on the Theory of Information Retrieval*, pages 385–388, 2015.

[31] Chanjun Liu and Peng Wang. A sunburst-based hierarchical information visualization method and its application in public opinion analysis. In *2015 8th International Conference on Biomedical Engineering and Informatics (BMEI)*, pages 832–836. IEEE, 2015.

[32] Hans Peter Luhn. A statistical approach to mechanized encoding and searching of literary information. *IBM Journal of research and development*, 1(4):309–317, 1957.

[33] Priyanka Mandikal and Raymond Mooney. Sparse meets dense: A hybrid approach to enhance scientific document retrieval. *arXiv preprint arXiv:2401.04055*, 2024.

[34] Thomas M. Mann. *Visualization of search results from the World Wide Web*. PhD thesis, Universität Konstanz, Konstanz, 2001.

[35] Christopher D Manning, Prabhakar Raghavan, and Hinrich Schütze. *Introduction to information retrieval*. Cambridge university press, 2008.

[36] Thorsten Merten, Daniela Jüppner, and Alexander Delater. Improved representation of traceability links in requirements engineering knowledge using sunburst and netmap visualizations. In *2011 4th International Workshop on Managing Requirements Knowledge*, pages 17–21. IEEE, 2011.

[37] Meta Open Source. React: The library for web and native user interfaces. `https://react.dev/`, 2024. Ultimo accesso: 28.05.2024.

[38] Lukas Michelbacher. *Multi-word tokenization for natural language processing*. PhD thesis, 01 2013.

[39] Tomas Mikolov, Ilya Sutskever, Kai Chen, Greg S Corrado, and Jeff Dean. Distributed representations of words and phrases and their compositionality. *Advances in neural information processing systems*, 26, 2013.

[40] George A Miller. Wordnet: a lexical database for english. *Communications of the ACM*, 38(11):39–41, 1995.

[41] Inc. MongoDB. MongoDB: The database for modern applications. `https://www.mongodb.com/`, 2024. Ultimo accesso: 28.05.2024.

[42] Tien Nguyen and Jin Zhang. A novel visualization model for web search results. *IEEE transactions on visualization and computer graphics*, 12(5):981–988, 2006.

[43] Silvio Peroni, Francesco Osborne, Angelo Di Iorio, Andrea Giovanni Nuzzolese, Francesco Poggi, Fabio Vitali, and Enrico Motta. Research articles in simplified html: a web-first format for html-based scholarly articles. *PeerJ Computer Science*, 3:e132, 2017.

[44] Francesco Poggi, Paolo Ciancarini, Angelo Di Iorio, Silvio Peroni, Fabio Vitali, et al. Exploiting coordinated views for scholarly reading and analysis. In *DMSVIVA*, pages 113–134, 2019.

[45] Radim Rehurek and Petr Sojka. Gensim–python framework for vector space modelling. *NLP Centre, Faculty of Informatics, Masaryk University, Brno, Czech Republic*, 3(2), 2011.

[46] Nils Reimers and Iryna Gurevych. Sentence-bert: Sentence embeddings using siamese bert-networks. *arXiv preprint arXiv:1908.10084*, 2019.

[47] Stephen E Robertson, Steve Walker, Susan Jones, Micheline M Hancock-Beaulieu, Mike Gatford, et al. Okapi at trec-3. *Nist Special Publication Sp*, 109:109, 1995.

[48] Kerry Rodden. Applying a sunburst visualization to summarize user navigation sequences. *IEEE computer graphics and applications*, 34(5):36–40, 2014.

[49] Mark Sanderson and W Bruce Croft. The history of information retrieval research. *Proceedings of the IEEE*, 100(Special Centennial Issue):1444–1451, 2012.

[50] Jeff Sauro and James R Lewis. *Quantifying the user experience: Practical statistics for user research*. Morgan Kaufmann, 2016.

[51] Marc M Sebrechts, John V Cugini, Sharon J Laskowski, Joanna Vasilakis, and Michael S Miller. Visualization of search results: a comparative evaluation of text, 2d, and 3d interfaces. In *Proceedings of the 22nd annual international ACM SIGIR conference on Research and development in information retrieval*, pages 3–10, 1999.

[52] William M Shaw Jr et al. The cystic fibrosis database: Content and research opportunities. *Library and Information Science Research*, 13(4):347–66, 1991.

[53] Michael Taylor, Hugo Zaragoza, Nick Craswell, Stephen Robertson, and Chris Burges. Optimisation methods for ranking functions with multiple parameters. In *Proceedings of the 15th ACM international conference on Information and knowledge management*, pages 585–593, 2006.

[54] Gemini Team, Rohan Anil, Sebastian Borgeaud, Yonghui Wu, Jean-Baptiste Alayrac, Jiahui Yu, Radu Soricut, Johan Schalkwyk, Andrew M Dai, Anja Hauth, et al. Gemini: a family of highly capable multimodal models. *arXiv preprint arXiv:2312.11805*, 2023.

[55] Andrew Trotman, Antti Puurula, and Blake Burgess. Improvements to bm25 and language models examined. In *Proceedings of the 19th Australasian Document Computing Symposium*, pages 58–65, 2014.

[56] Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N Gomez, Łukasz Kaiser, and Illia Polosukhin. Attention is all you need. *Advances in neural information processing systems*, 30, 2017.

[57] S Vijayarani, Ms J Ilamathi, Ms Nithya, et al. Preprocessing techniques for text mining-an overview. *International Journal of Computer Science & Communication Networks*, 5(1):7–16, 2015.

[58] Jonathan J Webster and Chunyu Kit. Tokenization as the initial phase in nlp. In *COLING 1992 volume 4: The 14th international conference on computational linguistics*, 1992.

[59] Kyle Wiggers. OpenAI launches an API to commercialize its research — venturebeat.com. `https://venturebeat.com/ai/openai-launches-an-api-to-commercialize-its-research/`, 2020. [Accessed 11-05-2024].

[60] ChengXiang Zhai and Sean Massung. *Text data management and analysis: a practical introduction to information retrieval and text mining*. Association for Computing Machinery and Morgan & Claypool, 2016.

[61] George Kingsley Zipf. The psychology of language. In *Encyclopedia of psychology*, pages 332–341. Philosophical Library, 1946.

Ringraziamenti

Vorrei ringraziare il Prof. Angelo Di Iorio e il Prof. Francesco Poggi per la disponibilità ed i preziosi consigli ricevuti durante lo svolgimento della tesi e nella sua stesura.

Un sincero ringraziamento ai miei genitori, nonché a tutte le persone che hanno creduto in me supportandomi in questo percorso.