LillAI

Motore di ricerca per dispense universitarie

Autori:

Paolo Di Simone - matr. 584638

Filippo M. Gaglioti - matr. 582704

Matteo Wissel - matr. 534693

Disponibile su GitHub:

https://github.com/Data-Integration-Lilla-team/LILL_AI

Data: 12 settembre 2023

Indice

1 Introduzione **3**

2 Use Case **4**

3 Tecnologie e modelli **5**

3.1 Lucene 5

3.2 LDA 6

4 Sistema **8**

4.1 Primitive 8

4.1.1 Memorizzazione 8

4.1.2 Indicizzazione 9

4.2 Crawler 10

4.2.1 Aggiornamento dbA 10

4.2.2 Aggiornamento dbB 10

4.3 Estrazione del testo 11

4.4 Cleaning 11

4.5 Indicizzazione 12

4.5.1 Lucene 12

4.5.2 LDA 13

4.6 Sistema di quering 14

4.7 Funzione di aggregazione 15

5 Risultati **16**

5.1 Data-set 16

5.2 Efficacia 17

5.2.1 Test LDA 17
5.2.2 Metriche 18
5.2.3 LDA accuracy 18
5.2.4 Luceene Accuracy 19
5.2.5 Lill-A.I: general accuracy 19
5.3 User satisfaction 20

6 Conclusioni e sviluppi futuri **21**

Bibliografia **22**

1 Introduzione

In ambito universitario è molto comune l'accumulo di una vasta quantità di materiali didattici, dal momento che quasi tutti i docenti forniscono dispense sotto forma di pdf o pptx. Tuttavia, con la crescita di questi documenti e la sovrapposizione che si crea fra il contenuto di ciascun corso, sorge un problema cruciale: come possiamo effettivamente accedere in modo efficiente alle informazioni contenute in questi documenti?

Il problema in questione si presenta come una sfida significativa per studenti e docenti. Spesso, si verifica un'ampia sovrapposizione di argomenti tra corsi diversi o persino all'interno dello stesso corso. Di conseguenza, la ricerca di informazioni specifiche relative a un particolare argomento può rivelarsi una vera e propria impresa. Attualmente, la soluzione predominante a questa problematica è un processo manuale e mnemonico.

Per affrontare questa sfida, emerge una necessità critica: sviluppare un sistema o una soluzione che consenta di recuperare in modo rapido ed efficiente informazioni specifiche riguardanti determinati argomenti all'interno dei documenti dei vari corsi universitari. Questo problema di gestione delle informazioni educative rappresenta un terreno fertile per l'applicazione della tecnologia, dell'elaborazione del linguaggio naturale e dell'informatica, con l'obiettivo di semplificare l'accesso all'informazione e migliorare il processo di apprendimento e insegnamento.

2 Use Case

Il problema consiste in:

- Un insieme di documenti (pdf) dei vari corsi universitari.
- Diversi corsi possono trattare argomenti uguali o simili.

La soluzione attuale consiste in una ricerca manuale, quindi:

1. Ricordare e trovare in quale corso è stato trattato un determinato argomento;
2. Ricordare e trovare in quale pdf del corso è contenuto l'argomento;
3. Trovare la slide in cui è trattato l'argomento.

La necessità è quindi quella di recuperare informazioni riguardanti un determinato topic dai pdf dei vari corsi.

3 Tecnologie e modelli

3.1 Lucene

Lucene [1] è una libreria Java che fornisce potenti funzionalità di indicizzazione e ricerca, nonché controllo ortografico, evidenziazione dei risultati e funzionalità avanzate di analisi e tokenizzazione. Lucene fornisce quindi un motore di ricerca completo che può essere integrato in varie applicazioni per consentire la ricerca testuale avanzata.

Ecco alcune delle principali funzionalità di Lucene:

- Indicizzazione: Lucene consente di creare un indice dei documenti testuali in modo efficiente. Gli indici sono strutture dati ottimizzate per la ricerca testuale rapida.

- Ricerca testuale: Lucene supporta una vasta gamma di query testuali, inclusi operatori booleani, wildcard, fuzzy search, e molto altro. Questo consente di effettuare ricerche complesse e precise all'interno dei documenti indicizzati.

- Ranking dei risultati: Lucene utilizza algoritmi di ranking per determinare la rilevanza dei documenti rispetto a una query specifica. Questo aiuta a restituire i risultati più pertinenti in cima alla lista.

- Gestione dell'analisi del testo: Lucene fornisce strumenti per l'analisi del testo, come il tokenization, lo stemming e la rimozione delle stop word. Questo aiuta a migliorare la precisione delle ricerche e a gestire le varianti delle parole.

- Supporto multilingua: Lucene è in grado di gestire testi in diverse lingue e offre analizzatori predefiniti per molte lingue comuni.

- Scalabilità: Lucene è progettato per essere altamente scalabile e può essere utilizzato in applicazioni che gestiscono grandi quantità di dati.

- Integrazione: Lucene può essere incorporato in diverse applicazioni, tra cui motori di ricerca web, sistemi di gestione dei contenuti, software di analisi dei dati e molto altro.

3.2 LDA

Il topic modeling è una tecnica di analisi del testo utilizzata nell'ambito dell'elaborazione del linguaggio naturale e del data mining. Consiste nell'identificazione e nell'estratto di temi o argomenti chiave presenti in un grande corpus di testi senza la necessità di etichettarli in anticipo. L'obiettivo principale è quello di scoprire automaticamente le strutture latenti all'interno dei testi, consentendo di raggruppare documenti simili in cluster e di identificare le parole più rilevanti associate a ciascun tema.

Il Latent Dirichlet Allocation (LDA) [2] è un modello statistico generativo utilizzato per il topic modeling di testi. Consente di scoprire argomenti nascosti all'interno di un corpus di documenti e assegnare a ciascun documento una distribuzione di probabilità sugli argomenti.

I principali componenti del modello LDA includono:

- Collezione di K argomenti (topics): LDA assume la presenza di K argomenti all'interno del corpus di documenti. Gli argomenti sono distribuzioni di probabilità sui vocaboli del corpus e rappresentano i temi nascosti che compaiono nei documenti.

- Distribuzione di Dirichlet sui temi: Per ogni argomento, è presente una distribuzione di Dirichlet, che modella la probabilità con cui i vocaboli del corpus sono associati all'argomento.

- Distribuzione di Dirichlet sulle proporzioni di argomenti per ciascun documento: Per ciascun documento, è presente una distribuzione di Dirichlet, che rappresenta la probabilità con cui gli argomenti compaiono nel documento.

L'obiettivo del modello è stimare i parametri di queste distribuzioni, ovvero le distribuzioni di argomenti sui vocaboli e le distribuzioni di proporzioni di argomenti nei documenti. Questi parametri vengono stimati utilizzando l'algoritmo di inferenza basato su tecniche MCMC (Markov Chain Monte Carlo) o variazionale.

4 Sistema

Il sistema lavora su un File-System centralizzato (ad esempio HDFS [3]). L'utente ha la piena libertà di gestire il File-System a sue piacere creando cartelle, rinominando file o spostando file da una cartella ad un'altra.

In questo capitolo verrà analizzata nel dettaglio l'implementazione dell'architettura in Figura 1 che permette il crawling e l'indicizzazione dei documenti.

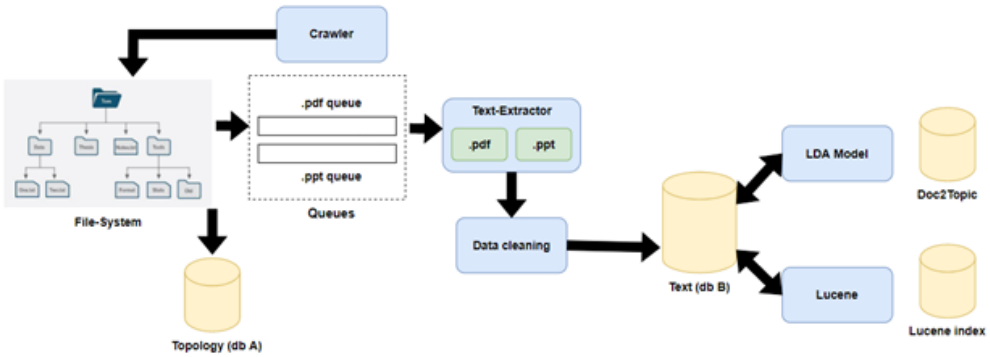

Figura 1: Architettura di crawling e indicizzazione

4.1 Primitive

Il sistema offre due primitive di sistema all'utente utilizzatore; queste sono:

- Memorizzazione
- Indicizzazione

4.1.1 Memorizzazione

L'utente seleziona i file da inserire nel File-System dedicato in una cartella da lui creata. Il sistema memorizza nel indice (dbA) le meta-informazioni

relative ai file caricati quali:

- Nome file: il nome del documento salvato;
- Path file: il nome del path selezionato per la memorizzazione del documento;
- Data caricamento: data di caricamento/modifica del documento;
- Formato file: estensione del file (.pdf, .pptx, .ppt) utile per la fase di memorizzazione;
- Indexed: valore booleano (false se il documento non ha subito indicizzazione, true altrimenti);

Parallelamente il sistema elabora il raw file ed estraendo dal documento informazioni relative al testo memorizzandole nel (dbB). Nello specifico il sistema estrae per ogni pagina/slide del file caricato:

- Testo
- Numero pagina

Tale operazione viene effettuata sfruttando diversi moduli python specifici nell'apertura ed elaborazione di file pdf e pptx. In questo modulo potrebbero essere inserite ulteriori sotto-operazioni come la traduzione dei file.

4.1.2 Indicizzazione

Questa primitiva offre all'utente la possibilità di indicizzare i documenti caricati o modificati. Il sistema scansiona il (dbA) ed individua i documenti aventi valore Indexed a false restituendo i loro path. I path sono utilizzati per accedere al (dbB) ed estrarre il testo delle pagine dei documenti da indicizzare. Il testo viene utilizzato per aggiornare i serach engines utilizzati dal

sistema ovvero Lucene e Topic-Modeling, come approfondito nei paragrafi successivi.

4.2 Crawler

Il desktop-crawler implementato ha il compito di navigare il file-system dedicato ed aggiornare i dbA (topologia) e gestire il corretto caricamento dei dati nel dbB (indice), mediante l'ausilio di moduli dedicati all'estrazione del testo dai diversi tipi di file ed alla pulizia del testo estratto.

Il crawler implementato nel sistema Lill-Ai è trasparente alla topologia dei file.

4.2.1 Aggiornamento dbA

Per l'aggiornamento del dbA (topologia) il sistema naviga il file-system e verifica quali documenti necessitano di essere indicizzati. Questi ultimi vengono suddivisi in diverse code (una per ogni tipologia di file supportato) a seconda dell'estensione del file. Le code verranno utilizzate per l'operazione di aggiornamento del dbB (indice).

4.2.2 Aggiornamento dbB

Sfruttando le code generate nello step precedente il crawler acquisisce i path dei documenti da processare. A seconda della coda (e quindi della tipologia di file) il crawler richiama un modulo specializzato nell'analisi e segmentazione (suddivisione) del file in pagine ed estrazione del testo. Nello specifico, il sistema supporta lettura ed indicizzazione di file di tipo .pptx (.ppt) e .pdf.

Terminata la segmentazione dei documenti in pagine, il crawler invoca il modulo di cleaning il quale ha il compito di pulire e preparare il testo per la memorizzazione.

4.3 Estrazione del testo

Una volta recuperati i documenti è necessario estrarre il testo da essi. A tal scopo sono state utilizzate due librerie:

- *PyPDF2* [4]: è una libreria python utilizzata per lavorare con file pdf. Essa fornisce una serie di funzionalità per la lettura, la manipolazione e la creazione di file pdf;
- *Python-pptx* [5]: è una libreria python open-source che consente di creare, modificare e manipolare presentazioni PowerPoint (.pptx). Con questa libreria, è possibile automatizzare la creazione di presentazioni PowerPoint, aggiungere diapositive, testo, immagini, forme, grafici e altri elementi, nonché formattare e personalizzare le diapositive secondo le proprie esigenze.

La pipeline di estrazione del testo è molto semplice: per ogni file recuperato in fase di crawling, si controlla il formato del file (pdf o pptx) e, a seconda di questo formato, si utilizza un modulo specifico (*extractor_pdf.py* o *extractor_pptx.py*) per l'estrazione del testo.

4.4 Cleaning

Il modulo di cleaning implementato all'interno di Lill-AI è stato ideato con l'intento di minimizzare il numero di token memorizzati, massimizzando il contenuto informativo presente all'interno di ogni pagina indicizzata. Inoltre, per via di performance non ottimale delle librerie utilizzate in fase di text-extraction, la pipeline di pulizia è stata resa "self-improving". Questo poiché ad ogni nuovo documento il sistema impara nuovi simboli da eliminare, ovvero

caratteri speciali mai visti in altri documenti e per il quale non si ha interesse nel memorizzararli. La pipeline di pulizia si presenta come segue:

1. Suddivisione parole: il sistema elabora ogni token ed individua eventuali parole erroneamente congiunte e le suddivide (es. HelloWorld → Hello World, MatteoWissel → Matteo Wissel). Tale operazione si è resa necessaria per via di inefficienze dei moduli di text extraction;

2. Rimozione stop-words e special characters: il sistema analizza ogni pagina ed elimina le stop-words presenti. Il sistema capitalizza le stop-words offerte dalla libreria NLTK [6]. In questa fase il sistema elimina anche eventuali caratteri speciali memorizzati all'interno del suo self-improving dictionary di special characters;

3. Eliminazione spazi doppi e trasformazione lower case;

4. Lemming.

4.5 Indicizzazione

4.5.1 Lucene

Per il parsing del documento utilizziamo due tipi di Analyzer, gli Analyzer sono delle funzioni che ci permettono di parsare il testo secondo delle regole da noi individuate. I tipi di Analyzer utilizzati sono:

- WhitespaceAnalyzer: è un analyzer che fa il parsing di una frase in base agli spazi;

- createNGramAnalyzer: è un analyzer customizzato che fa il parsing di una frase sulla base di n grammi;

Il documento viene indicizzato con due attributi titolo e contenuto. Per indicizzare il titolo abbiamo utilizzato il WhitespaceAnalyzer mentre per indicizzare il contenuto abbiamo utilizzato createNGramAnalyzer con n-grammi di dimensione minima 2 e massima 3.

4.5.2 LDA

Il modello LDA acquisisce in input il testo pulito dei documenti da indicizzare ed opera due operazioni necessarie ai fini dell'addestramento e predizione dei topic, ovvero: Tokenizzazione e Vettorizzazione. Nello specifico, la prima operazione operata sui documenti è la tokenizzazione, ovvero la trasformazione da testo in token (elementi da indicizzare). Per tale operazione si è deciso di utilizzare una tokenizzazione di singola parola (WhitespaceAnalyzer) in sinergia con il sistema parallelo Lucene. Questa scelta si è dimostrata essere la più efficacie come dimostrato nei test condotti utilizzando anche 2-grammi e 3-grammi.

Una vota trasformato il testo in token, il sistema opera una vettorizzazione, ovvero trasforma il testo in un formato vettoriale comprensibile al modello LDA. La rappresentazione scelta per tale operazione è Bag-of-Words (BOW) (nel Capitolo 6 si presenta possibili alternative più efficienti di BOW che potrebbero incrementare le performance del sistema). BOW trasforma ogni documento in una lista di coppie (*id_parola*, *numero_occorrenze*). Questa rappresentazione viene quindi utilizzata per l'addestramento del modello (la logica di funzionamento del modello è descritta nel Paragrafo 3.2).

Un punto fondamentale per questo sistema è stata la scelta degli iperparametri del modello discusso. Questi ultimi sono stati selezionati seguendo la strategia elbow-method e mediante l'analisi dei risultati ottenuti si è deciso di settare il modello di produzione con i seguenti iperparametri:

- *number_clusers*: 30
- *max_topic_for_cluser*: 5
- *passes*: 10 (numero di iterazioni su ogni documento in ogni epoca)
- *decay*: 0.5 (questo prametro gestisce la velocità di convergenza dell'algoritmo)
- *iterations*: 1000 (numero di epoche)
- *min_prob*: 0.2 (probabilità minima di un topic in un doc)

Le performance del sistema sono discusse nel Capitolo 5.2.

4.6 Sistema di quering

In Figura 2 è riportato il sistema che permette all'utente di sottomettere una query e recuperare i documenti.

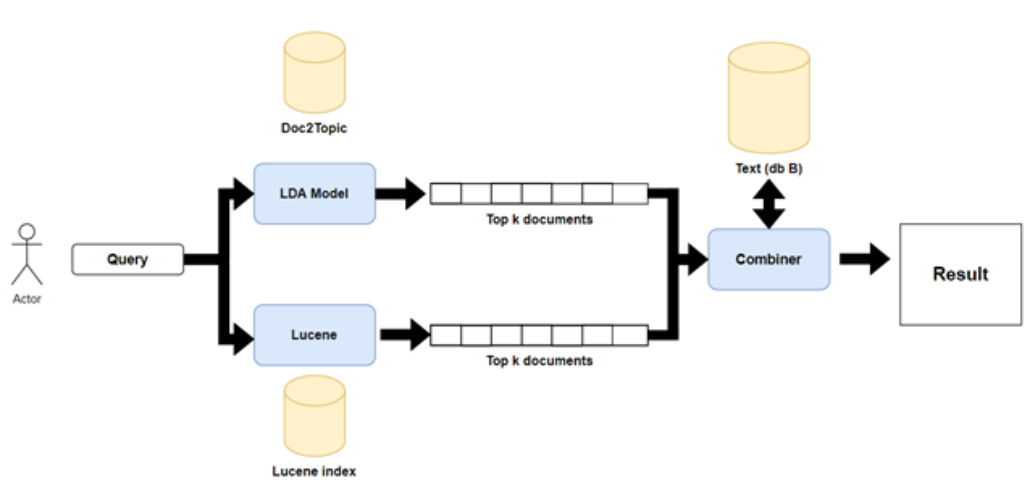

Figura 2: Sistema di quering

L'utente sottomette una query in linguaggio naturale. Il testo della query viene elaborato con la stessa logica utilizzata per l'indicizzazione dei documenti. Nello specifico, il testo viene pulito, tokenizzato ed utilizzato come input per i due sistemi (Lucene e LDA). I risultati (ranked) dei due sistemi vengono combinati con un opportuna funzione di aggregazione illustrata al Paragrafo 4.7.

4.7 Funzione di aggregazione

La funzione di aggregazione dei risultati dei due search engine può essere sintetizzata nella seguente logica.

1. La funzione prende in input una query Q;
2. Con LDA vengono calcolate le percentuali di affinità tra la query ed i topic, ottenendo una lista del tipo $[(topic_1, perc_1), ..., (topic_N, perce_N)]$;
3. Si selezionano i k topic con percentuale di affinità più alta;
4. Si recuperano i documenti che appartengono ai k topic selezionati dal sistema *doc2topic*;
5. Si sottomette la query a Lucene, che restituisce una lista di documenti rankata;
6. Si prende l'intersezione dei documenti restituiti da Lucene e LDA;
7. Si calcola lo score di ogni documento con la funzione in Eq. 1;
8. Si restituiscono i top n documenti con $score_{final}$ più alto.

$$score_{final} = \alpha \cdot score_{Lucene} + (1 - \alpha) \cdot score_{LDA} \tag{1}$$

5 Risultati

Il sistema Lill-A.I Ã¨ stato valutato eseguendo diversi test mirati a misurare l'efficacia e la user satisfaction. Per la valutazione sono stati preparati 4 test-case distinti, ognuno caratterizzato da un corpus di documenti specifico per di ramo universitario ovvero:

1. Ing. Informatica;
2. Economia lista del tipo $[(topic_1, perc_1), ..., (topic_N, perce_N)]$;
3. Architettura
4. All-in-one: composto da un sample di documenti selezionati dai diversi rami

Di seguito sono riassunte le caratteristiche relative ai dataset utilizzati nei 4 test-case discussi, nonchÃ© i risultati ottenuti dal sistema in termini di efficacia e user satisfaction.

5.1 Data-set

Per l'esecuzione dei test-sperimentali ci siamo avvalsi di diversi file in formato .pdf e .pptx (.ppt) specifici in una specifica area di interesse (come riportato nel paragrafo precedente). La Tabella ?? riporta la popolositÃ del dei documenti utilizzati per ogni test-case.

Inoltre, per misurare l'efficacia del sistema sono state preparate 5 query specifiche per ogni test-case e le documentazioni sono state opportunamente etichettate.

5.2 Efficacia

Per valutare l'efficacia complessiva del sistema ci siamo concentrati sulla valutazione delle performace dei due sistemi (LDA e Lucene) misurate sulla base dei risultati ottenuti dalle query note (discusse nel pragrafo precedente). Inoltre, in quanto il modello LDA Ã¨ un modello non supervisionato, ci siamo avvalsi di metriche note in tale contesto, quali Perplexity score e Topic Coherence. Si presentano di seguito i risultati ottenuti.

5.2.1 Test LDA

Le metriche utilizzate per la valutazione del modulo LDA sono:

- Perplessità (Perplexity): La perplessità è una misura di quanto bene il modello è in grado di predire il corpus di testo. Minore è il valore della perplessità, migliore è la capacità predittiva del modello.
- Coerenza degli argomenti (Topic Coherence): La coerenza degli argomenti valuta quanto coerenti e interpretabili sono gli argomenti estratti dal modello. Questa metrica misura quanto le parole associate a un argomento sono semanticamente simili tra loro.

In quanto gli iperparametri utilizzati non vengono modificati per ogni run, per garantire la completa automazione del processo, abbiamo definito una soglia minima di accuratezza, per entrambe le metriche.

Anche in questo caso sono stati preparati diversi test-case, ognuno avente un numero di record diversi e soprattutto riguardanti argomenti differenti.

I risultati sono riportati in Tabella 1;

Test case	Perplexity target max	Perplexity real	Topic Coherence target	Topic Coherence real
1	35	29,12	0,70	0,84
2	35	15,1189	0,70	0,82
3	35	30,124	0,70	0,87
4	35	12,33	0,70	0,72

Tabella 1: Risultati test LDA

5.2.2 Metriche

Le metriche utilizzate per misure l'accuracy del sistema sono:

- Precision
- Recall
- F1-score

5.2.3 LDA accuracy

Per i test il modello e' stato addestrato utilizzando i seguenti iperparametri:

- N Clusters: 30
- Max topic4Cluster: 4
- Passes: 10
- Decay: 0.5
- N iterzioni: 1000
- Threshold rilevanza: 0.2

Test case	Precision	Recall	F1-score
1	0.81	0.78	0,79
2	0.83	0.80	0,81
3	0,74	0,77	0,76
4	0,63	0,54	0,58

Tabella 2: Risultati test LDA 2

5.2.4 Luceene Accuracy

Test case	Precision	Recall	F1-score
1	0.88	0.90	0,88
2	0.86	0.83	0,84
3	0,91	0,82	0,86
4	0,73	0,64	0,68

Tabella 3: Risultati test Lucene

5.2.5 Lill-A.I: general accuracy

Per lo scopo l'iperparametro alpha Ã¨ stato settato a 0.7 dando maggiore importanza ai riusltati di Lucene rispetto a quelli LDA. La Tabella 4 mostra i risultati ottenuti per il sistema nel complesso.

Test case	Precision	Recall	F1-score
1	0.84	0.89	0,87
2	0.82	0.81	0,81
3	0,78	0,85	0,82
4	0,71	0,63	0,64

Tabella 4: Risultati Lill-A.I

5.3 User satisfaction

La user satisfaction del sistema é stata valutata facendo sottomettere a 3 studenti distiniti, ognuno esperto in un ramo dei test-set discussi, 5 query. I risultati delle query sono stati poi valutati su una scala 1 a 10, dove 10 Ã¨ risposta perfetta e 1 risposta insoddisfacente. Nell'ultimo test-case si sono rieseguite le medesime query sul corpus All-in-one. I voti assegnati ad ogni query sono calcolati come la media dei singoli voti inseriti dagli utenti. Nella Tabella 5 sono riportate le valutazioni effettuate dai partecipanti al questionario.

	Query1	Query2	Query3	Query4	Query5
Partecipante 1	4	7	8	2	6
Partecipante 2	5	9	8	9	3
Partecipante 3	8	7	4	6	9
Partecipante 4	5	7	6	6	5

Tabella 5: Valutazione utente

6 Conclusioni e sviluppi futuri

Abbiamo presentato in questa relazione Lill-AI un sistema user-friendly specializzato nel retrive di documenti di interesse generico per studenti universitari. Lill-AI capitalizza due motori di ricerca distinti i cui risultati vengono combinati per presentare all'utente la lista di pagine semanticamente rilevante alla query sottoposta. Sicuramente il sistema può essere migliorato incrementando il numero di features considerate in fase di retrieve, ad esempio capitalizzando sulle funzioni NLP di POS o NER. Presentiamo adesso in breve una lista di possibili migliorie da inserire all'interno del sistema per incrementare la sua efficienza.

- Tag: L'introduzione dei tag utente sui file memorizzati rappresenterebbe un significativo miglioramento per il nostro motore di ricerca. Questa innovazione consentirebbe agli utenti di arricchire i metadati associati ai propri file, aumentando la precisione delle ricerche e migliorando la pertinenza dei risultati. Inoltre, l'utilizzo dei tag utente faciliterebbe la categorizzazione e l'organizzazione dei dati personali, promuovendo un'esperienza utente più efficiente e personalizzata.

- NER e POS: L'integrazione di Named Entity Recognition (NER) e Part-of-Speech (POS) tagging nel nostro motore di ricerca apporterebbe notevoli vantaggi. L'impiego del NER consentirebbe l'identificazione e la classificazione di entità come nomi di persone, luoghi e organizzazioni nei documenti archiviati, migliorando così la precisione delle ricerche e facilitando la ricerca di informazioni specifiche in modo più rapido ed efficace. Allo stesso tempo, l'utilizzo del POS tagging consentirebbe un'analisi più dettagliata della struttura grammaticale dei testi, identificando le parti del discorso come verbi, sostantivi e aggettivi. Questo

rafforzerebbe la comprensione del contesto e la capacità del motore di ricerca di restituire risultati più coerenti e pertinenti.

In conclusione, l'adozione di NER e POS tagging costituirebbe un importante passo avanti nell'ottimizzazione del nostro motore di ricerca, migliorando la qualità delle ricerche, l'accuratezza dei risultati e l'esperienza complessiva dell'utente. Ciò contribuirebbe a consolidare la nostra posizione sul mercato e a soddisfare meglio le esigenze degli utenti.

Riferimenti bibliografici

[1] A. S. Foundation. "Apache Lucene - Scoring." (2011), indirizzo: `http://lucene.apache.org/java/3_4_0/scoring.html`.

[2] D. Blei, A. Ng e M. Jordan, "Latent Dirichlet Allocation," vol. 3, gen. 2001, pp. 601–608.

[3] Apache Software Foundation, *Hadoop*, ver. 0.20.2, 19 feb. 2010. indirizzo: `https://hadoop.apache.org`.

[4] M. Fenniak, *PyPDF2: A PDF toolkit for Python*, `https://github.com/mstamy2/PyPDF2`, Version 1.26.0, 2018.

[5] S. Canny, *python-pptx*, MIT License (MIT). indirizzo: `https://github.com/scanny/python-pptx`.

[6] S. Bird, E. Klein e E. Loper, *Natural language processing with Python: analyzing text with the natural language toolkit.* " O'Reilly Media, Inc.", 2009.