

## File System

Il sistema lavora su un File-System centralizzato (HDFS (?)). 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.

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

## Topologia-db (db A)

Il db A memorizza la topologia del File-System memorizzando per ogni documento individuato una serie di features utili in termini di retrival del documenti. Queste features sono:

* 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)

## Text-db (db B)

Il db B memorizza per ogni pagina il risultato dell’operazione di cleaning, ovvero il testo “pulito”. Il testo memorizzato nel db B è utilizzato per l’indicizzazione dei documenti (parallelamente sia a Lucene sia al modello LDA).

## Crawler

Il desktop-crawler implementato ha il compito di navigare il File-System dedicato ed implementa le seguenti funzionalità:

1. Aggiornamento del db A
2. Gestione del corretto funzionamento delle operazioni di estrazione e pulizia del testo

### Aggiornemtno db A

Per l’aggiornamento del db A (topologia) il crawler 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 db B (raw text)

### Aggiornamento db B

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.

## Text-Extractor

[PAOLO]

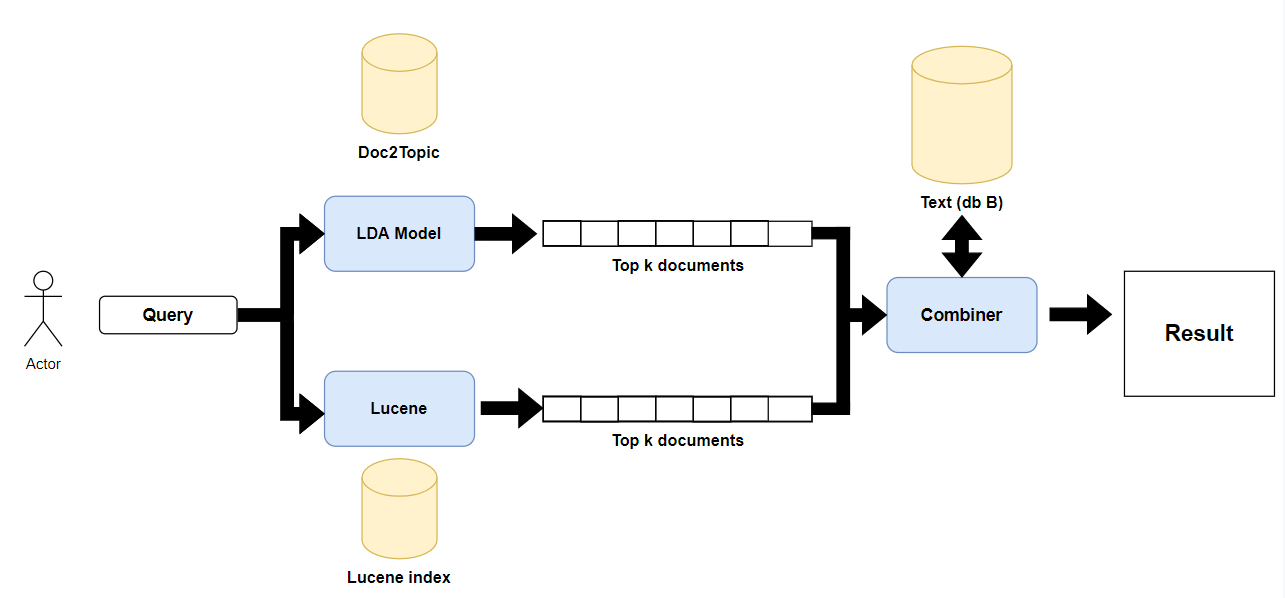
## Pipe-line di cleaning

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 pipe-line 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 memorizzare.

La pipe-line 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 & special characters**: il sistema analizza ogni pagina ed elimina le stop-words presenti. Il sistema capitalizza le stop-words offerte dalla libreria NLTK. 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 lower\_case transformation**
4. **Lemming**

****

# 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.

Traduzione automatica

# LDA MODEL

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
* Vettorizzazione

### Tokenizzazione

Nello specifico, la prima operazione operata sui docuementi 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.

### Vettorizzazione

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 Conclusioni e sviluppi futuri si presenta possibili alternative più efficienti a 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 nei Capitoli precedenti).

### Iperparametri

Un punto fondamentale per questo sistema è stata la scelta degli ipermarametri 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:

1. **number\_clusers**: 30
2. **max\_topic\_for\_cluser:** 5
3. **passes**: 10 (numero di iterazioni su ogni documento in ogni epoca)
4. **decay**: 0.5 (questo prametro gestisce la velocità di convergenza dell’algoritmo)
5. **iterations**: 1000 (numero di epoche)
6. **min\_prob**: 0.2 (probabilità minima di un topic in un doc)

Le performance del sistema sono discusse nei Capitoli successivi.