UNIVERSITÀ
DEGLI STUDI
DI PADOVA

DIPARTIMENTO
DI INGEGNERIA
DELL'INFORMAZIONE

DIPARTIMENTO DI INGEGNERIA DELL'INFORMAZIONE

CORSO DI LAUREA IN INGEGNERIA INFORMATICA

"Analisi e Refactoring di un Sistema di Monitoraggio Media per l'Estrazione di Tematiche da Articoli di Quotidiani"

Relatore: Prof. Emanuele Di Buccio

Laureando: Giovanni De Maria

ANNO ACCADEMICO 2023 – 2024

Data di laurea 27/09/2024

Indice:

Capitolo 1

Introduzione

Il monitoraggio media è un'attività cruciale per comprendere le dinamiche informative e l'opinione pubblica e con l'avvento delle tecnologie digitali e la proliferazione delle testate giornalistiche online, è emersa la necessità di sviluppare sistemi avanzati in grado di analizzare e rappresentare in modo efficace le tematiche trattate nei media. Per questo motivo è importante analizzare non solamente una ristretta cerchia di testate giornalistiche che parlano di un argomento specifico sul quale ci si vuole concentrare, ma piuttosto cercare di sviscerare più fonti possibili perché potrebbero trattare l'argomento in questione non esplicitamente e risultare altrettanto importanti rispetto a quelle che lo affrontano in maniera diretta.

In questo modo però si andrà sicuramente incontro a un problema relativo all'enorme quantità di dati da analizzare, la maggior parte dei quali potrebbe non essere di alcun interesse per l'argomento sul quale si sta cercando di documentarsi. Una soluzione a questo problema è data dall'utilizzo di algoritmi di Topic Modeling il cui obiettivo è identificare ed estrarre gli argomenti principali dato l'insieme dei documenti originali.

Questa tesi si propone di analizzare e fare il refactoring di un sistema di monitoraggio media pre-esistente, TIPS [1], con l'obiettivo di studiarne il funzionamento, migliorarne l'efficienza e affrontare dei problemi nel caso in cui se ne riscontrassero.

Il sistema in esame è strutturato in molteplici servizi, ciascuno dei quali svolge compiti specifici. Tra questi servizi, alcuni sono dedicati all'estrazione di rappresentazioni tematiche da un insieme di articoli pubblicati su testate giornalistiche online, che vengono poi restituite in formato JSON, offrendo una base di dati strutturata che può essere utilizzata per ulteriori analisi. Il focus principale di questa tesi riguarda proprio questi servizi, con particolare

attenzione allo studio delle loro funzionalità, alla loro interazione e al ruolo specifico che ricoprono.

1.1 Struttura della tesi

Il presente documento è articolato come segue:

Monitoraggio dei Media e modellazione delle tematiche. All'inizio di questa tesi, nel capitolo 2, verranno introdotti i concetti di Media Monitor, e di come questo concetto sia stato sviluppato e implementato nel sistema TIPS di riferimento, e di Topic Model e la sua interpretazione come modello generativo probabilistico. Infine viene introdotta la Latent Dirichlet Allocation come modello di riferimento.

Architettura del progetto. Nel terzo capitolo si parla dell'architettura del modulo relativo al topic modeling del progetto TIPS e di una serie di strumenti e framework utilizzati per realizzare i servizi di monitoraggio. Tra questi vi sono MongoDB per la gestione dei dati, Elasticsearch per le funzionalità di ricerca e indicizzazione, Mallet per l'analisi tematica e Spring per l'implementazione del backend. La scelta di questi strumenti è motivata dalle loro capacità di gestire grandi volumi di dati e di offrire prestazioni elevate in contesti di analisi complesse.

Testing. Nel quarto capitolo verranno illustrati i test che sono stati fatti sul sistema per capire dove poter agire per migliorarne l'efficienza e analizzare e risolvere gli eventuali errori.

Conclusione. Infine nell'ultimo capitolo vengono presentate le conclusioni e le possibili modifiche che si possono implementare per il miglioramento del sistema.

Capitolo 2

Monitoraggio dei Media e modellazione delle tematiche

Il Media Monitoring è il processo di raccolta, analisi e valutazione delle informazioni provenienti da diverse fonti mediatiche, come giornali, siti web, social media, blog e televisioni. L'obiettivo è monitorare ciò che viene detto su argomenti specifici, aziende, marchi o individui, per ottenere insight strategici utili alle decisioni aziendali, alla gestione della reputazione e alla comprensione delle tendenze di mercato.

Il problema del Media Monitoring è più attuale che mai, poiché la quantità di informazioni generate quotidianamente attraverso canali digitali, social media e piattaforme di notizie è cresciuta di molto nel tempo. Con miliardi di post, articoli e contenuti pubblicati ogni giorno, per le aziende, i governi e le organizzazioni diventa sempre più difficile gestire e analizzare questi dati in modo efficace.

Tra le tecniche avanzate utilizzate per analizzare efficacemente questi dati ci sono gli algoritmi di Topic Modeling, approcci che consentono di estrarre automaticamente i temi ricorrenti all'interno di grandi volumi di dati. I temi estratti consentono di ottenere una visione strutturata e sintetica delle discussioni presenti nei media, facilitando l'analisi dei trend e l’organizzazione delle informazioni rilevanti.

In questo capitolo verranno trattati inizialmente i Media Monitor e successivamente i Topic Model, con un approfondimento su un particolare algoritmo di machine learning.

2.1 Media Monitor

I media monitor [2] sono strumenti o servizi utilizzati per osservare, analizzare e raccogliere informazioni dai media. Questi possono includere sia i media tradizionali, come giornali, televisioni e radio, sia i nuovi media, come siti web, blog e social media.
Il monitoraggio media è una componente essenziale per comprendere come i temi scientifici e tecnologici vengono presentati e percepiti dal pubblico attraverso l'analisi dei contenuti.

L'analisi di questi ultimi può avvenire con diverse modalità: la maggior parte degli studi passati si concentrava su dei campioni, come può essere un sottoinsieme di articoli relativi al particolare argomento di studio. Questa modalità di ricerca può andare bene per certi tipi di studi, ma altri potrebbero richiedere l'analisi o il confronto con un panorama più vasto di fonti.

Quest'ultimo tipo di analisi può essere richiesto per esempio nello studio del numero degli articoli su un particolare argomento nel tempo: in un caso come questo lavorare solamente con documenti inerenti al tema che si sta studiando porterebbe inevitabilmente alla conclusione che l'attenzione mediatica nei confronti di quest'ultimo aumenti sempre di più nel tempo dato l'aumento del numero di articoli prodotti.
Questa conclusione potrebbe anche essere corretta ma le premesse e le modalità di studio che portano a quest'ultima sicuramente non lo sono, in quanto l'attenzione mediatica nei confronti di un determinato tema si può studiare solamente tenendo conto della totalità degli articoli. [3]

Il considerare altri campioni oltre a quelli che trattano direttamente un certo argomento è utile anche per un altro motivo: se siamo interessati a seguire il discorso su scienza e tecnologia nei giornali, concentrarsi solo sulle sezioni "Scienza" e "Tecnologia" potrebbe non rivelarsi una scelta adeguata perché potremmo perdere parte del discorso.

Nel caso dei giornali, però, gli studi sono raramente condotti su tutti i giornali disponibili online in quanto la mole di documenti sarebbe troppo grande ed anche il processo di monitoraggio di questi ultimi richiederebbe uno sforzo notevole: viene selezionato un sottoinsieme per includere quelli rappresentativi di diverse prospettive. Anche quando viene selezionato un sottoinsieme di giornali (e fonti in generale), la dimensione dei dati può richiedere una piattaforma scalabile per gestirla.

Un altro aspetto da considerare nello studio delle fonti è quello relativo al tipo di ricerca che si vuole fare: se si mira a analizzare l'evoluzione di un tema discusso nei media da molti anni, come per esempio le tecnologie informatiche, allora si potrebbe propendere per gli archivi dei giornali, mentre se si vogliono analizzare questioni più recenti o studiare nuove prospettive si potrebbe optare per un'analisi basata sui social media.

Media Monitor e il Progetto TIPS

Per affrontare i problemi sopracitati è stato creato il progetto interdisciplinare TIPS (Technoscientific Issues in the Public Sphere) che utilizza vari strumenti e framework che verranno esposti nel capitolo seguente.

Il primo problema, relativo al non limitare la propria ricerca a campioni di documenti ma piuttosto effettuare un monitoraggio continuo, è stato affrontato progettando e sviluppando una piattaforma software modulare che consente di analizzare articoli raccolti da quindici giornali in diverse lingue, per alcuni dei quali sono anche disponibili articoli che risalgono agli anni '80. Inoltre tutti i giornali sono ancora monitorati in modo da poter lavorare anche su questioni più recenti.

Per quanto riguarda il secondo problema, ovvero l'accesso e l'elaborazione di fonti eterogenee, la piattaforma è progettata per lavorare con documenti arbitrari e in diverse lingue, che permettono di eseguire studi comparativi tra diversi paesi. Anche se non vengono monitorate le piattaforme di social media, i dataset esistenti possono essere facilmente inclusi ed elaborati tramite la pipeline esistente.

Il sistema infine fornisce funzionalità che supportano il filtraggio e la ricerca, l'estrazione di entità nominate, l'identificazione della prima occorrenza di parole ed espressioni e funzionalità di ricerca avanzate come le espressioni regolari o i vincoli booleani. Una funzionalità dedicata nella piattaforma consente l'estrazione delle prime entità nominate e dei sostantivi per una data query, aiutando così gli utenti a esprimere meglio le loro esigenze. Inoltre, gli utenti possono lavorare su subcorpora specifici, ottenuti tramite query booleane, direttamente all'interno della piattaforma, utilizzando così tutte le funzionalità e gli indicatori disponibili senza dover fare affidamento su altre piattaforme o librerie.

2.2 Topic Model

I topic model sono approcci che permettono di individuare dei temi ricorrenti all'interno di un insieme di produzioni scritte, ossia una collezione di testi.
Tali approcci non necessitano di dati aggiuntivi e possono elaborare grandi volumi di informazioni in maniera autonoma, senza bisogno di esempi di risultati finali, e in tempi molto ridotti se paragonabili a quelli umani.

L'idea che sta alla base dei topic model è la seguente:

- Ogni topic è identificato da un insieme pesato di parole del vocabolario della collezione; nell'approccio che considereremo, ogni topic corrisponde ad una distribuzione di probabilità sulle parole del vocabolario.

- I documenti possono essere "composti" da uno o, più spesso, più topic contemporaneamente.

A livello pratico, ogni topic rappresenta un concetto, ma il modello non assegna automaticamente questo concetto al topic. Tale associazione può essere fatta da un utente umano che, analizzando le parole che hanno il peso maggiore all'interno della distribuzione di ciascun topic, la identifica e la associa a quest'ultimo. Queste parole più rilevanti, dette top words, possono essere considerate quelle più rappresentative del concetto corrispondente in quanto appaiono con maggiore frequenza.

Un esempio di topic potrebbe essere il seguente: le sue 5 top words sono "chatGPT", "IA", "intelligenza", "artificiale" e "quotidianità". Intuitivamente la tematica che potremmo associare al topic è l'utilizzo di ChatGPT.

Latent Dirichlet Allocation

Uno degli approcci di Topic Modeling più rappresentativi è la Latent Dirichlet Allocation (LDA) [4], un modello bayesiano composto da una struttura che, partendo dalla descrizione riportata in [5], potremmo sintetizzare con i seguenti punti:

- ogni topic corrisponde ad una distribuzione di probabilità sulle parole del vocabolario della collezione;

- ogni collezione di documenti ha un numero deciso a priori di topic che possono essere presenti nei documenti di tale collezione;

- ogni topic ha un peso all'interno di ogni documento, peso che indica quanto quel topic può essere rappresentativo del documento stesso;

- ogni parola in un documento ha un topic associato; più parole di un certo topic sono presenti all'interno di un documento, più esso sarà importante e rappresentativo del documento stesso.

Esempio di funzionamento

Immaginiamo di avere un grande quantitativo di articoli di giornale. Ogni articolo parla di vari argomenti, ma non sono mai esplicitamente indicati. Ad esempio, un articolo potrebbe parlare sia di politica che di sport. LDA cerca di identificare questi argomenti nascosti, i topic, e di capire di quali argomenti tratta ciascun articolo.

Idea principale: Ogni documento o articolo è visto come una combinazione di vari argomenti, e ogni argomento è visto come una combinazione di parole.

Documenti come misture di argomenti: Immaginiamo che ogni documento sia generato mescolando un certo numero di argomenti. Per esempio, un articolo potrebbe essere composto per il 70% da argomenti politici e per il 30% da argomenti sportivi.

Argomenti come miscele di parole: Ogni argomento è una distribuzione di parole. Un argomento "Economia" potrebbe contenere parole come "costi", "successo", "fondi", con probabilità diverse, mentre un argomento "sport" potrebbe contenere parole come "partita", "squadra", "sfida".

Obiettivo dell'LDA: Dato un insieme di documenti, LDA cerca di trovare quali sono gli argomenti più probabili e quale combinazione di questi è più adatta per descrivere ogni documento.

Il processo si articola essenzialmente nei seguenti passi:

1. **Inizializzazione casuale**: LDA inizia assegnando casualmente ogni parola in un documento a un argomento.
2. **Aggiornamento iterativo**: LDA poi itera su tutti i documenti e parole per migliorare l'assegnazione degli argomenti in modo che rispecchino meglio i dati osservati.
   Viene calcolata la probabilità che una parola appartenga a un certo argomento, basata su quante volte quella parola appare nell'argomento e su quante volte quell'argomento appare nel documento.
3. **Convergenza**: Dopo molte iterazioni, LDA stabilizza le assegnazioni di parole agli argomenti e dei documenti agli argomenti.

Supponiamo ad esempio di avere il seguente articolo:
“Chi vincerebbe sul ring fra Musk e Zuckerberg? La risposta degli esperti (e di ChatGPT)”

Step 1: Inizializzazione casuale
LDA inizia assegnando parole casuali a tre possibili argomenti, diciamo "Sport", "Economia" e “Informatica”.

Step 2: Iterazione
LDA vede che parole come "incontro", "sfida" appaiono frequentemente con "Sport", "fondi", "successo", "costi" appaiono con "Economia", mentre “chatGPT”, “online” appaiono spesso con “Informatica”.
Aggiorna le assegnazioni per rispecchiare queste osservazioni.

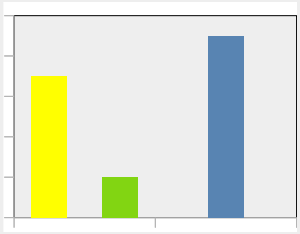
Step 3: Convergenza
Alla fine, LDA stabilisce che l’articolo in questione è una combinazione di “Sport”, “Economia” e “Informatica”.

LDA è utile quando non sappiamo in anticipo quali argomenti esistono nei documenti. Non dice esplicitamente "questo articolo è sulla politica", ma piuttosto assegna probabilità a diversi argomenti, che poi devono essere interpretate.

LDA richiede di definire il numero di argomenti in anticipo, il che potrebbe essere difficile se non si conosce bene il dominio. Inoltre, il processo di convergenza può essere computazionalmente oneroso su grandi dataset.

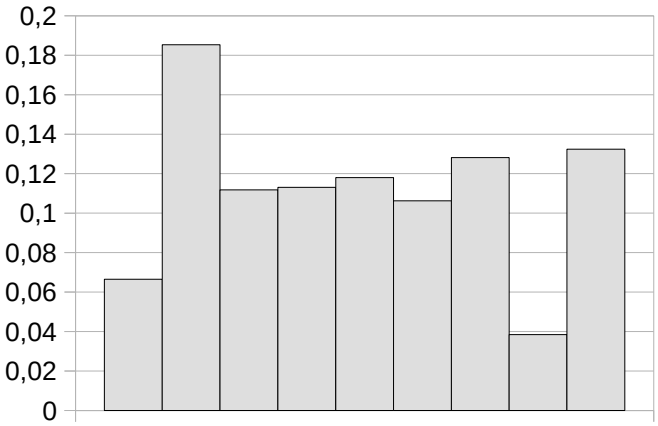
Chi vincerebbe sul ring fra Musk e Zuckerberg?
La risposta degli esperti (e di ChatGPT)

Davvero Elon Musk e Mark Zuckerberg si affronteranno su un ring (anzi, su un ottagono e dentro una gabbia), dopo la sfida lanciata dal primo e raccolta dal secondo? È più probabile di quel che sembra, tant'è che il numero uno dell'UFC, Dana White, ha confermato che i due sono "assolutamente intenzionati" a farlo. Anche perché per il numero uno di Tesla è adesso difficile tirarsi indietro senza fare una figuraccia: insomma, il combattimento potrebbe tenersi davvero, magari in modo simbolico, in diretta streaming e per raccogliere fondi per beneficenza. Chissà. Sperare non costa nulla. Così come fare qualche ipotesi su chi potrebbe davvero spuntarla: chi è fisicamente più preparato fra i due? Chi avrebbe più possibilità di successo? I bookmaker si sono già espressi online.



Il grafico rappresenta una possibile distribuzione dei topic all'interno dell'articolo con le parole che vengono associate a ognuno [6].

"Sport"	**"Economia"**	**"Informatica"**
combattimento	Elon	chatGPT
vincere	Musk	informazione
sport	potere	intelligenza
gabbia	Zuck	artificiale
agilità	Zuckerberg	IA



Il documento utilizzato per questo esempio è uno degli articoli presenti nel dataset utilizzato per i test descritti nel capitolo 4.

Invece per l'ottenimento di quest'ultimo grafico e dei topic con le top words, è stata scritta una classe apposita che applicasse l'algoritmo LDA all'articolo sopracitato. L'ordinata di questo grafico rappresenta il grado di pertinenza che l'algoritmo ha associato ai nove topic [6].

Capitolo 3

Architettura del sistema

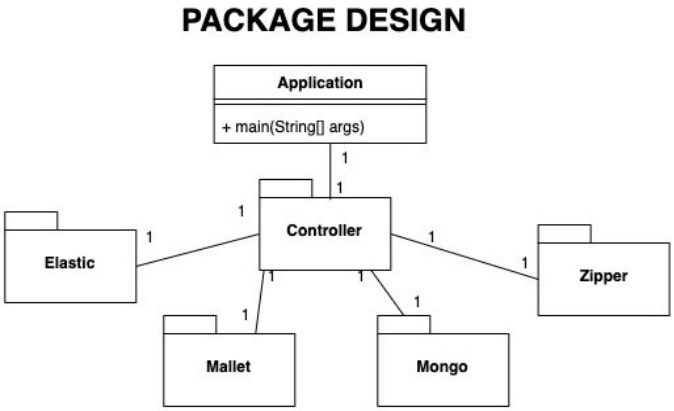


Figura 2.

Il sistema TIPS è organizzato in diversi servizi; questa tesi considera il sottoinsieme di tali servizi coinvolti nell'estrazione di rappresentazioni di tematiche da subcorpora e dell'accesso e l'analisi di tali rappresentazioni. I package sono rappresentati in Figura 2, e ognuno di essi contiene classi collegate tra loro per svolgere delle funzioni specifiche che saranno descritte più nello specifico nelle sezioni successive di questo capitolo.

Il funzionamento generale del sistema prevede il riconoscimento della richiesta effettuata dall'utente da parte del controller, che la memorizza nel database di MongoDB e successivamente si interfaccia a Elasticsearch per cercare e scaricare i documenti dal server.
Fatto ciò i documenti vengono analizzati tramite MALLET che applica l'algoritmo LDA e ne estrae i topic, le top words ecc. Infine lo zipper crea appunto un file .zip contenente i risultati della richiesta.

3.1 Struttura generale

Il controllo delle operazioni è appunto gestito dal package “controller” attraverso le seguenti classi:

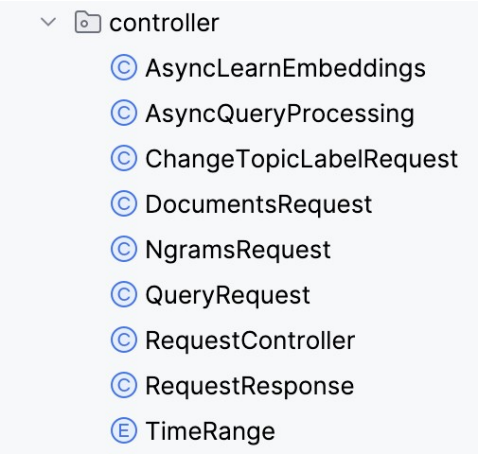


Figura 3.

La classe adibita all’effettivo controllo è *RequestController*, che gestisce tutte le possibili richieste tramite le relative funzioni e annotazioni Spring, framework che descriveremo in questo capitolo.
Tra le richieste più semplici che possono essere soddisfatte ci sono quelle che permettono di ottenere le query di un utente specifico, di ottenere una determinata query specificandone l’id di MongoDB, oppure di eliminare una certa query dal database sempre specificandone l’id.
La richiesta principale però è quella di creazione di una query, che coinvolge anche le altre classi presenti all’interno del package.

AsyncQueryProcessing è la classe che appunto elabora la richiesta e, come si può intuire dal nome, è contrassegnata dall’annotazione *@Async* di Spring che, come vedremo in seguito, permette di eseguire più processi in parallelo.
La classe, che vedremo nel dettaglio nel resto del capitolo, presenta una sola funzione che mette insieme i vari strumenti presenti all’interno del progetto, rendendola la classe di maggior interesse per la sezione sperimentale di questa tesi.

All’interno del package “controller” si trova un’altra classe che si distingue per la parola “Async”, che è *AsyncLearnEmbeddings*. Essa esegue dei processi asincroni per l’apprendimento di modelli di word embedding in modo da rappresentare il significato semantico delle parole.

Le altre classi presenti all’interno del package sono delle utility che hanno lo scopo di semplificare la gestione, ad esempio, degli n-grammi (*NgramsRequest*), dei documenti (*DocumentsRequest*) o delle query (*QueryRequest*).

3.2 MongoDB

MongoDB [7] è un database non relazionale, o NoSQL, progettato per gestire grandi volumi di dati non strutturati o semi-strutturati. A differenza dei database relazionali tradizionali (come MySQL o PostgreSQL), che memorizzano i dati in tabelle con righe e colonne, MongoDB utilizza una struttura dati più flessibile chiamata documento, che è simile a un oggetto JSON. Questa struttura consente di memorizzare dati in una forma più naturale e variabile, il che è particolarmente utile in contesti dove la flessibilità e la scalabilità sono essenziali.
MongoDB è disponibile sia in versione piattaforma su Cloud, chiamato MongoDB Atlas, sia in versione software, nella quale prende il nome di MongoDB server.

In MongoDB, un documento è una singola unità di dati e viene rappresentato come un insieme di coppie chiave-valore. Ogni documento è autonomo e può contenere strutture di dati nidificate come array o altri documenti. Questo consente di rappresentare dati complessi in modo intuitivo e vicino a come sono realmente strutturati.

Inoltre i documenti sono raggruppati in collezioni, le quali sono l'equivalente di una tabella in un database relazionale ma, a differenza delle tabelle, i documenti in una collezione possono avere strutture diverse. Questo significa che non tutti i documenti in una collezione devono avere gli stessi campi o tipi di dati, ritornando appunto a mettere il focus sulla flessibilità che vuole essere un punto a favore fondamentale per il database.

MongoDB ha diverse funzionalità, tra le quali possiamo trovare:

1. **Scalabilità orizzontale**: MongoDB è progettato per poter distribuire i dati su più server o cluster. Questo è possibile grazie a una tecnica chiamata sharding, che permette di suddividere i dati su più nodi, garantendo prestazioni elevate anche con set di dati molto grandi.

2. **Alta disponibilità**: MongoDB supporta la replica dei dati tramite un sistema chiamato replica set. Un replica set è un gruppo di server MongoDB che contengono gli stessi dati, fornendo ridondanza e aumentando la disponibilità del sistema. Nel caso in cui un server fallisse, un altro può prendere il suo posto automaticamente.

3. **Supporto per query avanzate**: MongoDB offre un potente linguaggio di query che consente di eseguire operazioni complesse sui dati, come ricerche di testo, aggregazioni, filtri e ordinamenti. Questo consente di estrarre e manipolare i dati in modi sofisticati senza dover ricorrere a linguaggi di programmazione esterni.

4. **Indici**: Come nei database relazionali, MongoDB supporta la creazione di indici sui campi dei documenti, ma a differenza degli altri, oltre agli indici tradizionali, MongoDB supporta anche indici su campi embedded e array, il che risulta utile per gestire dati complessi.

5. **Backup, ripristino e strumenti di gestione e monitoraggio**: MongoDB fornisce strumenti robusti per eseguire backup e ripristino dei dati, assicurando che i dati possano essere recuperati in caso di perdita o errore, inoltre offre una suite di strumenti per la gestione e il monitoraggio del database, tra cui MongoDB Atlas. Questi strumenti aiutano gli amministratori a monitorare le prestazioni, eseguire backup automatici, e gestire le risorse del database.

MongoDB e TIPS

Visto che cos'è e che funzionalità fornisce MongoDB, possiamo passare a comprendere il suo ruolo all'interno del sistema.

Il suo ruolo principale nel contesto del sistema considerato è quello della gestione e immagazzinamento delle query, che devono essere sempre disponibili per tutte le varie operazioni che l'utente può effettuare, come può essere la richiesta di visualizzare le query associate ad un particolare user o query_id.

Per quanto riguarda invece il suo ruolo nella creazione delle sopracitate query, oltre a memorizzarle una volta che sono state completate, esso consiste nella gestione di tutte le specifiche della stessa in modo da poter inizializzare anche le sezioni successive dell'elaborazione, come il modello LDA o l'estrazione dei Word Embedding.

Il pacchetto di classi e interfacce di MongoDB è organizzato come mostrato nell'immagine seguente:

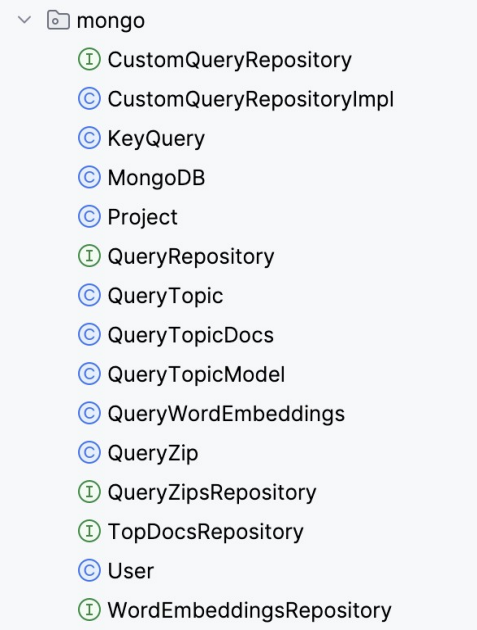


Figura 4.

Le classi principali del package sono *KeyQuery* e *MongoDB* che gestiscono rispettivamente le operazioni che riguardano le query, come l'ottenimento dell'id o del titolo, e le operazioni che riguardano il database di MongoDB.

In particolare la classe MongoDB implementa le varie funzionalità relative alle query di tutto il sistema: all'interno di questa classe si trovano le funzioni di creazione, salvataggio,

reperimento e eliminazione delle query nel database, oltre che quelle di binding (e relativo unbinding) delle query con l'user che le ha effettuate e quelle relative ai topic model e alle word embedding.

3.3 Elasticsearch

Elasticsearch [8] è un motore di ricerca e analisi distribuito, basato su Lucene [9], che offre potenti capacità di ricerca full-text, analisi dei dati e gestione di grandi quantità di dati strutturati e non strutturati. È stato sviluppato da Elastic NV e viene utilizzato in una vasta gamma di applicazioni, come la ricerca di informazioni in grandi database, l'analisi dei log, il monitoraggio delle prestazioni delle applicazioni e molto altro.

L'architettura di Elasticsearch è distribuita e si basa su una struttura detta "cluster" composta da più nodi. Sono riportati qui di seguito alcuni concetti utili alla comprensione dell'architettura:

1. Cluster: Un insieme di uno o più nodi che lavorano insieme per memorizzare dati e fornirli ai client. Un cluster è identificato da un nome univoco.

2. Nodi: Un nodo è una singola istanza di Elasticsearch che fa parte di un cluster. Ogni nodo può ospitare una o più parti di un indice (shard).

3. Shards e Repliche: Gli indici sono "namespace logici che contengono una raccolta di documenti, dove ciascun documento è una collezione di campi — che, a loro volta, sono coppie chiave-valore che contengono i dati" [10]; gli indici sono suddivisi in frammenti (shard) per distribuire il carico di lavoro e migliorare la velocità di ricerca. Ogni shard può essere replicato su altri nodi per garantire la ridondanza e la tolleranza ai guasti.

Le caratteristiche principali di Elasticsearch sono:

- **Ricerca fulltext**: Elasticsearch eccelle nelle funzionalità legate alla ricerca full-text, per analizzare e reperire i testi in modo molto efficiente. Supporta ricerche complesse, come la ricerca di frasi e di parole simili ma non esatte.

- **Scalabilità e distribuzione**: Una delle caratteristiche distintive di Elasticsearch è la sua capacità di scalare orizzontalmente su più nodi, gestendo grandi volumi di dati. L'indice in Elasticsearch può essere suddiviso in frammenti, e ciascun frammento può essere replicato su diversi nodi, garantendo alta disponibilità e tolleranza ai guasti.

- **Indice e analisi**: Durante l'indicizzazione, i dati vengono analizzati e suddivisi in termini di ricerca e ad ogni termine vengono associati i documenti che lo contengono ed eventuali statistiche, il che permette di ottimizzare le operazioni di ricerca e recupero.

- **API RESTful**: Elasticsearch è accessibile tramite un'interfaccia API RESTful, il che lo rende facile da integrare con altri sistemi. Tutte le operazioni, dalla gestione degli indici alla ricerca e aggregazione dei dati, possono essere eseguite tramite semplici chiamate HTTP.

- **Aggregazioni**: Oltre alla ricerca full-text, Elasticsearch supporta potenti funzionalità di aggregazione, che consentono di eseguire analisi avanzate sui dati. Le aggregazioni permettono di calcolare metriche come somme, medie, minimi, massimi, e di suddividere i dati in gruppi (bucket) per analizzare distribuzioni e tendenze.

- **Analisi in tempo reale**: Una delle principali applicazioni di Elasticsearch è l'analisi in tempo reale di dati, come log di server, dati di monitoraggio, e stream di eventi. Elasticsearch è spesso utilizzato insieme a strumenti come Logstash e Kibana (che insieme formano la cosiddetta "ELK Stack") per raccogliere, analizzare e visualizzare dati in tempo reale.

- **Gestione dei documenti JSON**: I dati in Elasticsearch sono memorizzati come documenti JSON, che rappresentano strutture di dati chiave-valore. Questa flessibilità

permette di gestire facilmente dati semi-strutturati e di effettuare query complesse su di essi.

Elasticsearch e TIPS

All'interno del progetto TIPS la funzione che viene ricoperta da Elasticsearch è relativa al reperimento e indicizzazione dei documenti richiesti per l'elaborazione delle query.

Le funzionalità di Elasticsearch all'interno del progetto TIPS sono divise in tre classi come indicato nella figura seguente:

Figura 5.

La classe *Document* gestisce tutto ciò che concerne l'entità documento, quindi le sue caratteristiche, come l'id, il titolo, il testo ecc.
Inoltre la classe è dotata di una sottoclasse che ha lo scopo di creare un'entità documento, *DocumentBuilder*.

Successivamente la classe *ElasticClient* ha il compito di gestire un client di Elastic, che viene configurato e inizializzato dalla classe *Config*.
La gestione del client da parte della classe prevede l'implementazione di diverse funzioni utili per la ricerca dei documenti:

- public List<Document> SearchQueryKeyword(KeyQuery query)
- public Document SearchDocument(String docId)
- public List<Document> SearchDocuments(List<String> documentIds, String elasticIndex)
- private List<Document> GetRequestDocs(SearchRequest request)

3.4 Spring

Spring [11] è un framework per lo sviluppo di applicazioni Java, nato con l'obiettivo di semplificare la creazione di applicazioni enterprise, soprattutto per quanto riguarda la gestione dei componenti e la configurazione. È uno dei framework più popolari e utilizzati nel mondo Java, grazie alla sua flessibilità, modularità e capacità di adattarsi a diversi scenari.

Spring è stato creato da Rod Johnson nel 2003 come risposta alla complessità crescente delle applicazioni Java Enterprise Edition (JEE). L'idea alla base di Spring è quella di ridurre la necessità di codice boilerplate, migliorando la testabilità del software. Questo è reso possibile attraverso l'uso di concetti chiave come l'Inversion of Control (IoC) e la Dependency Injection (DI) che verrà trattata nella sezione seguente.

I concetti principali di Spring sono i seguenti:

- **Inversion of Control (IoC)** (e Dependency Injection): Il principio dell'IoC sposta la responsabilità della gestione dei componenti (o "bean") dall'applicazione al framework. Invece di creare oggetti e gestirne le dipendenze manualmente, Spring si occupa di gestire il ciclo di vita degli oggetti.

- **Aspect-Oriented Programming**: Spring supporta la programmazione orientata agli aspetti, che permette di separare il codice che si occupa di "cross-cutting concerns" (esigenze trasversali) come logging, transaction management, sicurezza, ecc., dal codice di business. Questo migliora la modularità e la manutenibilità del software.

- **Data Access e ORM Integration**: Spring offre strumenti per semplificare l'accesso ai dati, integrandosi con tecnologie di persistenza come JDBC, JPA, Hibernate, e altri ORM (Object-Relational Mapping). Grazie all'astrazione fornita da Spring, è possibile cambiare il layer di persistenza con minimo impatto sul codice.

- **Web Framework**:
  - Spring MVC è un framework per lo sviluppo di applicazioni web basato sul pattern Model-View-Controller (MVC). Permette di costruire applicazioni web

modulari e manutenibili, separando chiaramente le responsabilità di gestione dei dati (Model), visualizzazione (View), e controllo del flusso applicativo (Controller).

- Spring WebFlux: A partire da Spring 5, è stato introdotto WebFlux, un framework reattivo per lo sviluppo di applicazioni non bloccanti. Questo è particolarmente utile per applicazioni che devono gestire un elevato numero di richieste simultanee.

- **Spring Boot**: Spring Boot è un'estensione del framework Spring che facilita la creazione di applicazioni standalone, pronte per la produzione. Automatizza molte delle configurazioni necessarie per avviare un'applicazione Spring, riducendo drasticamente il tempo necessario per avviare nuovi progetti. È particolarmente utile per le architetture a microservizi [12], dove velocità e semplicità sono cruciali.

- **Test**: Spring offre un supporto esteso per il testing delle applicazioni. Grazie alla sua architettura basata su DI, è facile testare unitariamente i componenti dell'applicazione. Spring supporta anche il testing d'integrazione, offrendo strumenti per simulare contesti di applicazione reali durante l'esecuzione dei test.

- **Modularità**: Il framework Spring è modulare, il che significa che è possibile usare solo i moduli necessari per la propria applicazione. Ad esempio, è possibile includere solo la parte relativa all'accesso ai dati senza il web framework, se l'applicazione necessita solo della prima. Questo riduce la complessità e il footprint del progetto.

Dependency Injection

La Dependency Injection [13], uno dei principali concetti sposati da Spring, è un insieme di principi e patterns di software design che permette di sviluppare del codice a basso accoppiamento. Ma che cosa significa codice a basso accoppiamento?
Ciò significa che i componenti o i moduli di un sistema software sono progettati per essere indipendenti l'uno dall'altro, di modo che effettuando modifiche ad un componente l'impatto sugli altri sia minimo o, idealmente, nullo.

La scrittura di codice a basso accoppiamento è un mezzo che serve ad arrivare a un codice manutenibile, che è lo scopo ultimo che si deve avere quando si scrive del codice software.

Adesso passiamo a vedere come tutto ciò può essere applicato al software design sfruttando una similitudine con il cablaggio elettrico come in [13].
Se per far funzionare un asciugacapelli lo si collega direttamente al muro senza passare attraverso una presa elettrica allora la sua manutenzione o sostituzione diventa una questione complicata e che può essere svolta solamente da personale specializzato. Inoltre per fare ciò può essere per esempio necessario staccare la corrente all'abitazione in cui si trova l'apparecchio, andando quindi a influenzare tutto il sistema circostante.

Questa è la definizione di alto accoppiamento, di cui abbiamo anche già dato implicitamente una soluzione per trattarlo, ossia attraverso l'uso di una presa e di una spina che, tornando alla similitudine iniziale con il software design, rappresentano rispettivamente un'interfaccia e una sua implementazione.

Inoltre, finché l'implementazione è relativa all'interfaccia e obbedisce al "contratto" della stessa, è possibile combinare applicazioni in molti modi, sfruttando diversi pattern:

- *Null Object* design pattern: questo design pattern consiste nel creare un'implementazione di un'interfaccia che non ha alcuna funzionalità, ma che può risultare utile, in un sistema a basso accoppiamento, per sostituire una vera implementazione senza causare problemi.
  Ciò può essere applicato per esempio con i client: un client si aspetta sempre di avere un servizio disponibile, quindi se quest'ultimo viene rimosso si otterrà una *NullReferenceException.* Utilizzando il *Null Object* pattern si potrà sostituire il servizio senza causare problemi.

- *Decorator* design pattern: questo design pattern consiste nell'intercettare un'implementazione di un'interfaccia con un'altra, sempre della stessa interfaccia. Questo permette di introdurre gradualmente nuove funzionalità senza il bisogno di riscrivere o cambiare molto del codice esistente.

- *Composite* design pattern: consiste nell'aggiungere nuove funzionalità a un codice esistente, come il pattern precedente, ma ristrutturando un'implementazione esistente di un'interfaccia con un'altra. Il *Composite* design pattern si ottiene aggregando diverse implementazioni in una.

- *Adapter* design pattern: sempre tornando alla similitudine con le prese e le spine, l'*Adapter* design pattern funziona esattamente come un'adattatore per le spine, può essere utilizzato per far corrispondere due interfacce correlate ma separate fra di loro.

Spring e TIPS

All'interno del sistema TIPS, Spring viene utilizzato attraverso molte delle sue annotazioni:

- *@Bean* è utilizzata in Spring Framework per definire un metodo all'interno di una classe di configurazione che restituisce un oggetto che deve essere gestito automaticamente dal container di Spring.
- *@SpringBootApplication* è un'annotazione che semplifica la configurazione di base di un'applicazione Spring Boot combinando diverse annotazioni comunemente utilizzate, in particolare:
  - *@Configuration* che indica che la classe annotata è una configurazione per Spring e può contenere definizioni di bean;
  - *@EnableAutoConfiguration* che abilita la configurazione automatica di Spring Boot, cercando automaticamente le classi di configurazione e aggiungendole al contesto dell'applicazione;
  - *@ComponentScan* che abilita la scansione automatica dei componenti, dei servizi e delle repository all'interno del pacchetto base dell'applicazione e dei suoi sotto package.
- *@EnableAsync* è un'annotazione di Spring Framework che abilita la gestione asincrona dei metodi all'interno di un'applicazione Spring. Quando viene utilizzata, abilita la creazione di thread aggiuntivi per l'esecuzione di metodi annotati con *@Async,* eseguendoli in un thread separato rispetto al thread principale dell'applicazione. Questo è utile per operazioni che possono richiedere del tempo, come chiamate di rete, operazioni I/O intensive o processi che possono essere eseguiti

in parallelo senza influire sul flusso principale dell'applicazione. Per abilitare questa funzionalità è necessario anche avere una classe annotata con *@Configuration*.

- *@PostConstruct* è un'annotazione che viene utilizzata in un metodo di un bean gestito da Spring per eseguire operazioni di inizializzazione dopo che tutte le dipendenze del bean sono state iniettate e prima che esso sia messo a disposizione per l'uso. Quando un bean viene inizializzato, Spring lo esamina alla ricerca di un metodo annotato con *@PostConstruct* e, se lo trova, esso viene eseguito automaticamente dopo che tutte le sue dipendenze sono state risolte. Questa annotazione è particolarmente utile quando si ha bisogno di eseguire operazioni di inizializzazione che dipendono dalle dipendenze del bean stesso. Ad esempio, potrebbe essere necessario inizializzare una connessione al database, eseguire alcune operazioni di setup o configurare lo stato iniziale del bean.
- *@RestController* è un'annotazione in Spring Framework che combina le funzionalità di due altre annotazioni: *@Controller* e *@ResponseBody*.
  @Controller indica che la classe è un controller di Spring MVC. I controller sono responsabili di gestire le richieste HTTP e di restituire una rappresentazione appropriata della risposta, ad esempio una pagina HTML.
  @ResponseBody indica che il valore restituito dai metodi del controller deve essere serializzato direttamente nel corpo della risposta HTTP, piuttosto che essere trattato come nome di una vista da risolvere.
  Quindi, *@RestController* è un'abbreviazione conveniente per creare un controller che restituisce direttamente dati JSON o XML (solitamente utilizzati in API RESTful) anziché HTML.
- *@Autowired* è un'annotazione di Spring Framework che viene utilizzata per l'iniezione delle dipendenze. Quando viene applicata a una proprietà, un costruttore o un metodo setter di una classe, Spring cercherà automaticamente un bean appropriato per iniettarlo.
- *@RequestMapping* è un'annotazione di Spring Framework utilizzata per mappare richieste HTTP a metodi di gestione nelle classi dei controller. Questa annotazione è una delle più flessibili e può essere utilizzata per mappare metodi a qualsiasi tipo di richiesta HTTP (GET, POST, PUT, DELETE, ecc.) e a qualsiasi percorso URL.
- *@GetMapping* (come *@DeleteMapping* e *@PostMapping*, che però riguardano le richieste HTTP DELETE e HTTP POST) è un'annotazione di Spring Framework utilizzata per mappare richieste HTTP GET a metodi di gestione nelle classi dei

controller. Questa annotazione è un'abbreviazione conveniente per la combinazione di *@RequestMapping*. Quando si utilizza *@GetMapping*, si specifica il percorso dell'URL che il metodo del controller deve gestire.

- *@PathVariable* è un'annotazione di Spring Framework utilizzata per estrarre i valori dalle variabili di percorso di un URL e utilizzarli come parametri nei metodi dei controller. Quando viene utilizzata, si specifica il nome della variabile di percorso nell'URL e il nome del parametro nel metodo del controller. Spring assocerà automaticamente il valore della variabile di percorso specificata nell'URL al parametro del metodo del controller annotato con *@PathVariable*.
- *@RequestBody* è un'annotazione di Spring Framework utilizzata per indicare che un parametro di un metodo del controller deve essere deserializzato dal corpo della richiesta HTTP. Quando viene utilizzata si specifica che il corpo della richiesta HTTP verrà mappato direttamente all'oggetto Java specificato come parametro del metodo del controller.
- *@Service* è un'annotazione di Spring Framework che viene utilizzata per indicare che una classe è un bean di servizio all'interno dell'applicazione. Questo significa che la classe è destinata a fornire servizi o altre funzionalità specifiche dell'applicazione. Spring la rileva automaticamente durante la scansione dei componenti e la registra nel contesto dell'applicazione come un bean gestito. Questo rende la classe disponibile per l'iniezione delle dipendenze in altre classi, come i controller, le repository, o altri bean di servizio.

3.5 Mallet

MALLET (MAchine Learning for LanguagE Toolkit) [14] è un framework software open source progettato per l'analisi statistica di testi e la modellazione dei dati testuali. Sviluppato principalmente in Java, è ampiamente utilizzato in ambito accademico e industriale per compiti di elaborazione del linguaggio naturale (NLP), machine learning e text mining.

MALLET opera principalmente su una pipeline che include diverse fasi.

I documenti vengono importati nel sistema e vengono applicate tecniche di pre-processing come tokenizzazione, rimozione di stop-words e stemming/lemmatizzazione.

Dalle parole o frasi estratte vengono generate delle feature che rappresentano il testo in una forma che può essere utilizzata dagli algoritmi di machine learning.

Le feature vengono quindi utilizzate per addestrare modelli di machine learning, come modelli di classificazione o di topic modeling. I modelli possono essere addestrati su dati etichettati (approccio supervisionato) o su dati non etichettati (approccio non supervisionato).
I modelli addestrati possono essere valutati su set di dati di test o applicati a nuovi documenti per prevedere categorie, temi, o altre proprietà.

Le funzionalità e caratteristiche principali di MALLET sono:

- Topic Modeling: Una delle funzionalità più conosciute di MALLET è la modellazione a temi, in particolare l'algoritmo LDA, introdotto nel capitolo 2.

- Classificazione di testi: MALLET include diversi algoritmi di machine learning per la classificazione dei testi, come il Naive Bayes. Questi algoritmi permettono di classificare automaticamente i documenti in categorie predefinite sulla base del loro contenuto.

- Tokenizzazione e pre-processing del testo: Il toolkit offre strumenti per la tokenizzazione (ossia la divisione del testo in parole o frasi), la rimozione di stop-words (parole comuni che spesso non sono informative, come "il", "e", "di" in italiano), la normalizzazione delle parole (per esempio, la riduzione alla forma base o "lemma") e altre tecniche di pre-processing che preparano i dati per l'analisi successiva.

- Gestione di grandi corpus: MALLET è progettato per gestire e analizzare grandi quantità di dati testuali. Può facilmente processare corpora composti da milioni di documenti.

Mallet e TIPS

Il ruolo di MALLET all'interno del sistema TIPS è un misto delle funzionalità viste nel capitolo precedente, come mostra l'elenco delle classi della Figura 6.

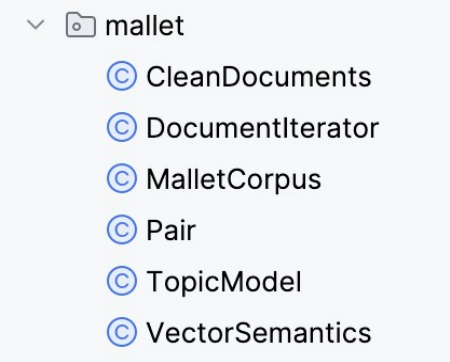


Figura 6.

CleanDocuments è la classe che è stata progettata per preparare una lista di documenti per l'elaborazione linguistica, filtrandoli in base a una lista di "stop words", ai tag grammaticali e ai simboli di punteggiatura. Tutto ciò avviene grazie al costruttore della classe oppure grazie a specifiche funzioni che possono anche elaborare un testo invece di un intero documento.

La classe *DocumentIterator* è un semplice iteratore per navigare in una lista di documenti, come *Pair* è una semplice classe per gestire un oggetto coppia.

MalletCorpus, in particolare il suo costruttore, prende una lista di documenti e li elabora attraverso una serie di trasformazioni, successivamente esegue un'operazione di potatura per ridurre il vocabolario alle caratteristiche più rilevanti creando una nuova lista di istanze potate. Tutto ciò ha lo scopo di preparare il corpus per l'uso in algoritmi di machine learning.

La classe *TopicModel* è quella che effettua fisicamente l'analisi degli argomenti, creando un TopicModel grazie al builder, eseguendo il training sui documenti forniti e estraendo i principali argomenti, le parole associate a ciascun argomento, ed i documenti rilevanti per ciascuno.
Un'altra funzionalità della classe è legata a una classe della libreria MALLET, infatti l'attributo "topicModel" è un'istanza di "ParallelTopicModel", classe appunto della libreria MALLET che permette di eseguire l'analisi degli argomenti in parallelo su più thread, migliorando l'efficienza su grandi dataset.

Infine la classe *VectorSemantics* è responsabile dell'addestramento di modelli di word embedding, ovvero rappresentazioni vettoriali delle parole basate sul contesto in cui appaiono

nei testi. La classe permette di configurare vari parametri per l'addestramento, ottenere i vettori associati a singole parole, e accedere a tutti i vettori di parole nel corpus.

Capitolo 4

Testing

Dopo aver studiato la struttura del modulo per effettuare il topic modeling del progetto TIPS è il momento di cercare di capire se ci sono e quali possono essere i suoi problemi.

Per prima cosa è stato studiato Docker, in particolare è stato utilizzato Docker Desktop [15], che è poi servito per la creazione e gestione dei container di MongoDB e Elasticsearch. Quest'ultimo nello specifico è stato popolato con articoli di giornale estratti da testate online, in particolare gli articoli considerati sono 1676 ed esclusivamente in italiano.

Dopo la configurazione del container di Elasticsearch sono state realizzate due funzioni per monitorare in modo immediato la situazione del container di MongoDB, che sono quelle in Figura 7 e Figura 8.

```
@GetMapping(value = "/queries/export", produces = MediaType.APPLICATION_JSON_VALUE)  no usages
String ExportAllQueries() {
    //System.out.println("Chiamata funzione ExportAllQueries di RequestController.java");
    List<KeyQuery> allQueries = mongoDB.AllQueries();

    System.out.println("Number of total queries: " + allQueries.size());
    String filepath = "/Users/giovannidemaria/Documents/Università/Tesi/Progetto/tm-rest-service 2/src/main/resources/all_queries.txt";

    try (BufferedWriter writer = new BufferedWriter(new FileWriter(filepath))) {
        writer.write( str: "[");
        int i = 0;
        for (KeyQuery keyQuery : allQueries) {
            if(i != 0)
                writer.newLine();
            writer.write( str: "{\"_id\":\"" + keyQuery.get_id() +
                    "\",\"keyword\".\"" + keyQuery.getKeyword() +
                    "\",\"requestDate\".\"" + keyQuery.getRequestDate() +
                    "\",\"user\".\"" + keyQuery.get_user() +
                    "\",\"NumTotalDocs\".\"" + keyQuery.getNumTotalDocs() +
                    "\",\"NumTopics\".\"" + keyQuery.getNumTopics() +
                    "\"}");
            i++;
        }
        writer.write( str: "]");
        return "Export completed!";
    } catch (IOException e) {
        e.printStackTrace();
        return "Export failed";
    }
}
```

Figura 7.

```java
@GetMapping(value = "/queries/total", produces = MediaType.APPLICATION_JSON_VALUE)  no usages
List<KeyQuery> AllQueries() {
    //System.out.println("Chiamata funzione AllQueries di RequestController.java");
    List<KeyQuery> allQueries = mongoDB.AllQueries();

    System.out.println("Number of total queries: " + allQueries.size());
    return allQueries;
}
```

Figura 8.

Queste due funzioni hanno dei compiti molto semplici:

- *AllQueries* restituisce una lista di tutte le queries presenti all'interno del container.

- *ExportAllQueries*, invece, le scrive in un formato leggibile all'interno di un file .txt.

Entrambe queste funzioni possono essere chiamate effettuando una richiesta HTTP, come si può notare grazie all'uso dell'annotazione *@GetMapping* di Spring.

Scritte queste due funzioni, la prima parte del testing è stata focalizzata sui tempi di esecuzione dei vari tipi di query e i risultati sono i seguenti:

1. Le query che riguardano l'accesso alle richieste eseguite da un certo utente vengono risolte in un tempo molto breve quindi non causano problemi al sistema.

2. Le richieste che necessitano della creazione di una nuova query, quindi di tutto il processo che riguarda Elasticsearch e Mallet, invece hanno una durata variabile in base al numero di documenti che riguardano la parola chiave o le parole chiave specificate nella query. Se il numero di documenti è modesto la durata può variare da i 30 secondi a qualche minuto per ogni singola query, mentre per le richieste che richiedono l'analisi di un numero molto elevato di documenti la durata può arrivare anche a 15 minuti. Per esempio se si effettua una query cercando articoli su ChatGPT i documenti che il sistema reperisce è di poco più di un migliaio e il tempo di esecuzione totale è circa 15 minuti.

Il passo successivo è stato quello di scrivere una classe apposita per simulare un numero elevato di richieste in sequenza, dato che farlo tramite riga di comando non è pratico, e la classe scritta è quella delle Figure 9 e 10.

Uno dei problemi per effettuare tale test era legato alla funzione findDuplicates all'interno di *AsyncQueryProcessing*: essa non permette di effettuare il processo completo nella creazione delle query se trova un esatto duplicato della query stessa all'interno del database, quindi le opzioni per la scrittura delle stesse da fare in sequenza erano due:

1. Scrivere un numero elevato di query diverse tra di loro per effettuare ogni volta un processo di generazione dei risultati completo, ma in ogni caso così facendo prima o poi si sarebbe giunti ad avere query uguali tra loro.

2. "Commentare" la chiamata della funzione findDuplicates e scrivere anche solo una singola query, in questo modo ad ogni iterazione il processo di creazione della query sarebbe stato completo.

Considerate entrambe le opzioni si è optato per la seconda come illustrato nel codice in Figura 9.

Nonostante la scelta di cui sopra, tra un'esecuzione e l'altra è possibile cambiare la query, in particolare la keyword che la identifica, e quindi testare diverse richieste in modo da far analizzare al sistema un numero variabile di documenti.

```
public class Simulator {  2 usages
    int n_queries;  2 usages

    public Simulator(int queries){  1 usage
        n_queries = queries;
    }

    public void QuerySimulator(){  1 usage

        //Inizializzo HttpClient
        try (CloseableHttpClient client = HttpClients.createDefault()) {

            //Dati per le richieste
            String postUrl = "http://0.0.0.0:8080/query";
            String getUrl = "http://0.0.0.0:8080/queries/giovanni";
            String exportUrl = "http://0.0.0.0:8080/queries/export";
            String jsonBody = """
                    {
                      "user": {
                        "username": "giovanni"
                      },
                      "keyQuery": {
                        "endDate": "2024-01-01",
                        "elasticIndex": "newssamples",
                        "projectId": "tsm-it",
                        "description": "Documents on Sport",
                        "startDate": "2022-01-01",
                        "keyword": "sport"
                      }
                    }
                    """;
```

Figura 9. Sezione relativa alla scrittura dei diversi tipi di query

```
//Loop per inviare richieste ciclicamente
for (int i = 0; i < n_queries; i++) {
    if(i % 2 == 0) {
        try {
            //Creo la richiesta POST
            HttpPost post = new HttpPost(postUrl);
            post.setHeader( name: "Content-Type",  value: "application/json");
            post.setEntity(new StringEntity(jsonBody));

            //Eseguo la richiesta
            try (CloseableHttpResponse response = client.execute(post)) {
                //Stampo la risposta
                System.out.println("Response status code: " + response.getStatusLine().getStatusCode());
                System.out.println("Response body: " + EntityUtils.toString(response.getEntity()));
            }

            //Attendo prima di inviare la prossima richiesta
            //Thread.sleep(1000);
        } catch (Exception e) {
            e.printStackTrace();
        }
    }
    else{
        //Creo la richiesta GET
        HttpGet getRequest = new HttpGet(getUrl);

        try (CloseableHttpResponse getResponse = client.execute(getRequest)) {
            String getResponseString = EntityUtils.toString(getResponse.getEntity());
            System.out.println("GET Response: " + getResponseString);
        }
```

Figura 10. Sezione relativa alla generazione di query in sequenza

I risultati di questo tipo di test mostrano che il sistema esegue contemporaneamente in parallelo tre richieste di creazione di nuove query grazie al fatto che la classe *AsyncQueryProcessing* è trattata da Spring tramite l'annotazione *@Async*. Se il numero di richieste è superiore a tre le successive andranno in coda, mentre l'esecuzione delle GET request avviene nonostante i tre thread siano occupati.

Il passo successivo a quello appena trattato è stato simulare un servizio MongoDB sovraccaricato di richieste, quindi il numero di query presenti all'interno del database è stato portato circa al migliaio. L'unico intoppo che si è potuto riscontrare in questa fase è legato a un numero massimo di richieste che possono essere contemporaneamente in coda, che per il sistema è 600.

Successivamente è stato testato il tempo di esecuzione di una singola query, che con il processo completo poteva arrivare anche a 15 minuti. Attraverso l'output successivo alla creazione dei risultati, si nota che il processo non si conclude quando sono pronti i topic a causa della sezione di relativa alla creazione dei WordEmbeddings nell'*AsyncQueryProcessing*.
"Commentando" questa sezione all'interno della classe sopracitata la durata diminuisce drasticamente fino ad arrivare a 3-4 minuti per le richieste più corpose.

Capitolo 5

Conclusioni

In questa tesi sono stati esposti diversi strumenti e framework utilizzati all'interno del sistema TIPS, in particolare relativi alla sezione di Topic Modeling dello stesso.
Lo scopo di questo studio era quello di apprendere il funzionamento dei sopracitati strumenti per poter analizzare il sistema alla ricerca di eventuali problematiche attraverso una fase di testing.

Questa fase è stata caratterizzata dalla scrittura di semplice codice al fine di sovraccaricare in vari modi il sistema e il database: quest'ultimo è stato popolato con circa un migliaio di richieste; sono state effettuate diverse query, di natura diversa, e in rapida successione alla ricerca di eventuali breakpoint che però non si sono verificati.

In particolare la rimozione del controllo dei duplicati nella creazione delle query ha avuto due effetti importanti:

1. Non essendo più presente la verifica della presenza della query all'interno del database (esattamente la stessa in termini di keyword e subcorpora), questo comporterà un aumento della quantità di spazio necessaria per memorizzare i risultati delle elaborazioni dal momento che due query identiche verranno considerate come distinte.

2. D'altra parte però, questo aumento di risorse richieste viene compensato dal fatto che non ne sono più necessarie per l'individuazione dei duplicati.
   Inoltre il fatto che ciascuna query sia considerata come distinta, ha in vantaggio che eventuali informazioni aggiunte a quella query e memorizzate in MongoDB, siano proprie solo della stessa. Una delle funzionalità che infatti TIPS mette a disposizione è quella di poter associare delle etichette ai topic estratti da LDA; tali etichette (stringhe)

derivano dall'interpretazione che utenti esperti danno dei topic estratti sulla base delle parole con maggior peso per quel topic e dei documenti in cui quel topic è prominente. Prima ciò non era possibile, mentre ora lo è, consentendo cosi a ciascun utente di dare la propria interpretazione e di non doverla condividere con gli altri che hanno eseguito la stessa query.

In conclusione si può giungere al fatto che il sistema, o almeno la versione locale su un dispositivo singolo, non presenta problematiche particolari da trattare a parte i lunghi tempi di processing delle query più pesanti, ma grazie all'intervento di cui sopra si è risolta una questione di condivisione di informazioni che non dovrebbero essere condivise tra i vari utenti.

L'utilità di tutti gli strumenti esposti che gravitano attorno alla scrittura di questa tesi trascende quella dello studio e analisi del solo sistema TIPS in questione: la conoscenza appresa può essere applicata anche ad altri tipi di progetti futuri.

Sviluppi futuri

Il lavoro svolto in questa tesi porta a pensare di implementare una gestione più efficace delle richieste, e una possibile soluzione potrebbe essere l'utilizzo di una coda, in particolare di un servizio come RabbitMQ [16].

RabbitMQ è un broker di messaggi e streaming che permette di implementare una comunicazione asincrona tra diversi sistemi e componenti software.

Nel caso specifico in questione, ovvero del sistema TIPS, l'ambiente è un'applicazione web che riceve molte richieste di elaborazione da utenti diversi. Invece di gestire tutte le richieste immediatamente, l'applicazione potrebbe inviare le richieste a RabbitMQ, che le deposita in una coda. Le componenti software infine elaborano le richieste in background, distribuendo il carico di lavoro e garantendo che ogni richiesta venga gestita correttamente senza sovraccaricare l'applicazione.

L'integrazione di RabbitMQ nel sistema esistente, oltre all'installazione ed all'implementazione di RabbitMQ, richiederebbe:

- L'identificazione del client da utilizzare per interagire con RabbitMQ; due opzioni potrebbero essere il client ufficiale di RabbitMQ per java, rabbitmq-java-client, oppure Spring AMQP, che supporta il protocollo AMQP e ed è una libreria di Spring, già utilizzata nel sistema.

- La configurazione dei Producer, ossia quelle entità che inviano i messaggi a RabbitMQ, in particolare a una coda specificata dal Producer stesso. Per quanto riguarda il sistema TIPS si potrebbe considerare un Producer per la gestione delle query relative al modulo di Topic Modeling ed eventualmente utilizzare altri Producer per code relative allo scambio di messaggi tra altri servizi.

- La configurazione dei Consumer, il cui ruolo è quello di attendere per ricevere messaggi da una particolare coda ed elaborarli. Nel caso di TIPS, i consumer sarebbero alcuni dei servizi presentati nel Capitolo 3.

L'adozione di software come RabbitMQ richiederà un'ulteriore e dettagliata analisi per capire in quali parti dell'architettura e seguendo quali pattern tale software potrebbe apportare un giovamento. Inoltre sarà necessario progettare ed eseguire una fase di testing con diversi tipi di query e diverso carico; il capitolo 4 di questa tesi potrebbe essere un punto di partenza per la progettazione e l'esecuzione di tali test.

Capitolo 6

Bibliografia

[1] E. Di Buccio, A. Cammozzo, F. Neresini, A. Zanatta (2022). TIPS: Search and Analytics for Social Science Research, *CEUR Workshop Proceedings.*

[2] E. Di Buccio, F. Neresini (2024). Research and Teaching Public Communication of Science and Technology on Digital Data, *CEUR Workshop Proceedings.*

[3] Neresini, F. (2018). Old media and new opportunities for a computational social science on PCST. *Journal of Science Communication*, 16(02), C03. https://doi.org/10.22323/2.16020303

[4] Blei, D. M., Ng, A. Y. & Jordan, M. I. (2003). Latent Dirichlet Allocation. *Journal of Machine Learning Research, 3*, 993–1022.

[5] G. Toto, Un algoritmo di topic modeling per microblog, Tesi di Laurea, *Dipartimento di Scienze Statistiche, Università degli Studi di Padova*, A.A. 2021/2022. https://hdl.handle.net/20.500.12608/11379.

[6] Blei, D. M. (2012). Probabilistic topic models. *Communications of the ACM, 55*(4), 77–84. https://doi.org/10.1145/2133806.2133826

[7] "What is MongoDB?". mongodb.com, 2024, https://www.mongodb.com/docs/manual

[8] "What is Elasticsearch". elastic.co, 2024, https://www.elastic.co/guide/en/elasticsearch/reference/current/elasticsearch-intro.html

[9] "Welcome to Apache Lucene". lucene.apache.org, 2011, https://lucene.apache.org/, *The Apache Software Fundation*.

[10] D. Brimley (2023). What is an Elasticsearch Index. *Elasticsearch B.V.* https://www.elastic.co/blog/what-is-an-elasticsearch-index

[11] "Building REST services with Spring". spring.io, 2005, https://spring.io/guides/tutorials/rest

[12] "Microservices". spring.io, 2005, https://spring.io/microservices

[13] S. van Deursen, M. Seeman (2019). Dependency Injection Principles, Practices, and Patterns, *Manning Publications Co.*

[14] S. Graham, S. Weingart, I. Milligan (2012). Getting Started with Topic Modeling and MALLET, *The Programming Historian.*

[15] "Overview of Docker Desktop". docker.com, 2013, https://docs.docker.com/desktop/, *Docker Inc*

[16] "RabbitMQ". rabbitmq.com, 2005, https://www.rabbitmq.com, *Broadcom Inc.*