UNIVERSITÀ POLITECNICA DELLE MARCHE
FACOLTÀ DI INGEGNERIA
Dipartimento di Ingegneria dell'Informazione
Corso di Laurea Magistrale in Ingegneria Informatica e dell'Automazione

TESI DI LAUREA

Definizione di un framework per l'integrazione di ELK e Python in un contesto di Natural Language Processing

Definition of a framework for integrating ELK and Python in a Natural Language Processing context

Relatore

Prof. Domenico Ursino

Correlatore

Riccardo Zanella

Candidato

Giacomo Licci

ANNO ACCADEMICO 2021-2022

Tutti sono capaci di parlare o di scrivere in modo oscuro e noioso: la chiarezza e la semplicità invece sono scomode. Non solo perché richiedono più sforzo e più talento, ma perché quando si è costretti a essere chiari non si può barare.

Piero Angela

Sommario

Negli ultimi anni il Natural Language Processing è divenuto sempre più uno tra i campi di maggiore successo dell'Intelligenza Artificiale. Sempre più aziende stanno integrando sistemi di ricerca documentale che siano in grado di restituire documenti coerenti il più possibile con il contesto definito nella query di ricerca. Tuttavia, nonostante alcuni di questi motori di ricerca abbiano raggiunto importanti risultati in merito alla ricerca semantica, quest'ultima risulta ancora poco matura, incapace di garantire una scalabilità orizzontale efficiente.

La presente tesi si propone di descrivere il processo di sviluppo di un framework per consentire l'esecuzione di ricerche di tipo semantico, attraverso un interfacciamento tra il linguaggio Python e il motore di ricerca documentale Elasticsearch.

Keyword: Natural Language Processing, Text Mining, Information Extraction, Named Entity Recognition, Elasticsearch, Ricerca semantica, Ricerca full-text

Indice

Elenco delle figure

Elenco delle tabelle

Introduzione

Negli ultimi anni, grazie alla maggiore disponibilità di dati e alle maggiori capacità computazionali fornite dai calcolatori, il campo dell'*Intelligenza Artificiale* è tornato sempre di più in auge, a distanza di tempo dalla sua prima era d'oro.

Tra i diversi campi in cui si dirama l'Intelligenza Artificiale, quello del *Natural Language Processing* è, senza alcun dubbio, uno tra quelli di maggiore successo nell'era moderna. Tra le ragioni di tale successo ne è la domanda sempre maggiore di sistemi *smart*, in grado di soddisfare le richieste dell'essere umano attraverso il solo utilizzo del linguaggio naturale. Basti pensare agli assistenti vocali, come Alexa, Siri, Cortana, etc., i quali vengono utilizzati non solo all'interno dei moderni smartphone e computer, ma anche per la gestione della domotica, all'interno delle case e nei luoghi di lavoro. Un altro esempio riguarda la disponibilità H24 dei servizi offerti ai consumatori, in un mondo dove è sempre più necessaria la possibilità di acquistare un prodotto, oppure di ricevere assistenza, in qualsiasi ora del giorno, per tutti i giorni. Tutti questi bisogni hanno spinto le aziende sempre più nell'integrazione di sistemi automatici, in grado di soddisfare le esigenze dei propri clienti, anche in assenza di personale tecnico.

Un altro settore nel quale è sempre più richiesta l'integrazione del Natural Language Processing, nonché ambito di questo elaborato, riguarda il perfezionamento dei motori di ricerca documentale. Grazie alla maggiore informatizzazione della società, ogni giorno viene prodotta sempre più informazione, sotto forma di documenti. La gestione e l'utilizzo di questa quantità di risorse a disposizione garantisce un forte contributo nell'ottenimento del vantaggio di mercato per le aziende, ma richiede, allo stesso tempo, la gestione di un problema di tipo ingegneristico. Per queste motivazioni, i sistemi di ricerca documentale, in grado di svolgere ricerche testuali su un'insieme di documenti, risultano elementi centrali di importanti studi e progressi tecnologici.

L'analisi dei documenti testuali si sviluppa in due categorie principali: la ricerca sintattica e la ricerca semantica. La prima tipologia di ricerca è quella più matura e largamente utilizzata; algoritmi tipici di ricerca sintattica utilizzano formule di conteggio delle parole come *Term Frequency–Inverse Document Frequency* (TF-IDF) per misurare la rilevanza dei documenti all'interno di un corpus definito precedentemente. Diversi framework esistenti, come Elasticsearch, implementano, già da diverso tempo, questa tipologia di ricerca. Tuttavia, la ricerca sintattica può risultare non abbastanza espressiva per determinati casi d'uso; per questo motivo nasce la necessità di sfruttare la ricerca semantica all'interno dei sistemi di ricerca. L'obbiettivo di questa tipologia di ricerca è il medesimo della sua controparte sintattica, ma essa sfrutta le capacità dei sistemi di Machine Learning (ML) per misurare la rilevanza dei documenti, a partire da una data query. La motivazione che giustifica l'integrazione di

modelli di ML per svolgere la ricerca semantica riguarda la capacità dei suddetti modelli di estrarre il significato delle parole, mantenendo il contesto in cui esse sono definite. Per mezzo della ricerca semantica sarà possibile guardare oltre alle semplici parole e ricercare documenti sulla base dei concetti espressi, al di là della sintassi utilizzata nella frase di ricerca. La ricerca semantica presenta, quindi, un'espressività molto importante e di grande valore per tutti i motori di ricerca di nuova generazione. Tuttavia, il basso grado di maturità di questi ultimi non garantisce tutt'ora una scalabilità orizzontale efficiente, nonostante il raggiungimento di importanti sviluppi in merito a questo campo applicativo.

La presente tesi si colloca proprio in tale contenuto. In particolare, essa descrive lo sviluppo di un possibile framework per la realizzazione di ricerche di tipo semantico attraverso l'esecuzione di un task di Natural Language Processing su specifici documenti. Il framework realizzerà un interfacciamento tra Elasticsearch, lo strumento software impiegato per l'archiviazione e l'elaborazione dei dati, e il linguaggio Python, utilizzato per svolgere tutte le funzioni principali all'interno dello strumento software.

In una prima fase verrà svolta un'analisi dei dati, con, conseguente riadattamento di essi, per renderli conformi con un formato compatibile con Elasticsearch. Una volta terminato il caricamento e l'indicizzazione dei documenti all'interno del software, verranno svolte le prime ricerche eseguibili sui documenti (ricerche sintattiche), dando una dimostrazione delle funzionalità di base fornite da Elasticsearch.

Al termine della realizzazione della ricerca sintattica, si passerà alla progettazione dell'altra tipologia di ricerca, di tipo semantico, la quale corrisponde alla parte centrale di tutto il lavoro svolto. Nel dettaglio, verranno analizzati i possibili task di Natural Language Processing realizzabili all'interno di Elasticsearch. Il task che verrà svolto sui documenti in esame prende il nome di *Named Entity Recognition* (NER); esso permette di svolgere un'estrazione delle parole di maggior significato all'interno del testo; tali parole rimangono, comunque, conformi al contesto definito all'interno del corpus.

Una volta attivate le funzionalità di Machine Learning fornite da Elasticsearch, attraverso un'apposita licenza, verrà selezionato un modello per NER preallenato, disponibile sulla piattaforma *HuggingFace*, compatibile con lo strumento software.

Terminati il deploy e l'attivazione del modello, quest'ultimo verrà utilizzato per svolgere una prima azione di inferenza sui documenti che sono stati caricati nella fase precedente. Per svolgere questa azione, il modello verrà integrato all'interno di un costrutto fornito da Elasticsearch stesso, il quale implementa l'esecuzione di una sequenza di processi elementari (o pipeline) su più documenti, con un'unica azione. Il suddetto costrutto prende il nome di Ingest Pipeline.

Successivamente alla costruzione della pipeline e alla sua esecuzione, nell'elaborazione iniziale dei documenti verranno definiti diversi casi d'uso, nei quali è possibile impiegare la Ingest Pipeline definita, sulla base dei documenti ottenuti dalla sua esecuzione. Attraverso essi, sarà possibile dimostrare l'efficacia della ricerca semantica rispetto alla ricerca sintattica, mediante un confronto dei risultati ottenuti dalle query utilizzate per le due tipologie di ricerca.

Per quanto riguarda, invece, tutte le applicazioni implementabili con la sola ricerca semantica, esse fungeranno da dimostrazione per indicare l'espansione del campo di utilizzo di Elasticsearch in merito al task di NER. Infine, una volta mostrati i campi applicativi, specifici per il caso in esame, verrà svolta una valutazione complessiva dello strumento software, con l'ausilio di determinate metriche, definite a priori. Verranno, quindi, mostrati gli aspetti positivi e negativi del comportamento mostrato da Elasticsearch in merito al lavoro svolto per il caso in esame.

Tutti i dettagli relativi al lavoro appena descritto nonché le nozioni teoriche necessarie per la sua comprensione verranno presentati nel resto della tesi. Quest'ultima è strutturata

come di seguito specificato:

- Nel Capitolo 1 verrà presentata un'introduzione in merito allo strumento software utilizzato per lo sviluppo del framework, ovvero Elasticsearch. Nel dettaglio verrà esposta una panoramica del suo sviluppo storico per poi descrivere la sua architettura, insieme alle funzionalità messe a disposizione per la ricerca full-text.

- Nel Capitolo 2 verrà definita la tematica del Natural Language Processing (NLP), descrivendo le sue fasi principali, le tipiche problematiche affrontate e i rispettivi campi applicativi. Successivamente verrà approfondito più nel dettaglio un ambito applicativo dell'NLP, il quale riguarda l'estrazione delle informazioni dal testo; tale attività è nota come Text Mining. Come per il Natural Language Processing, verranno illustrate le fasi principali che caratterizzano un tipico task di Text Mining, insieme alle problematiche e alle applicazioni di quest'ultimo.

- Nel Capitolo 3 verranno illustrate le prime operatività realizzate su Elasticsearch, a partire da un dataset di riferimento, attraverso il framework costruito in linguaggio Python. Inizialmente, verrà illustrata una descrizione del dataset e del cluster Elasticsearch utilizzati, per poi definire il processo di creazione degli indici utilizzati per il salvataggio dei documenti sul cluster. Successivamente, verranno illustrate tutte le tipologie di ricerca sintattica, fornite dallo strumento software e rese possibili dai dati a disposizione.

- Nel Capitolo 4 verrà fornita una descrizione delle principali funzionalità di Machine Learning, compatibili con Elasticsearch. Al termine della panoramica definita poc'anzi, verrà analizzato il percorso di abilitazione delle suddette funzionalità, a partire da una installazione esistente. Il processo di abilitazione comprenderà, al suo interno, tutti risvolti in merito alla gestione delle risorse impiegate dal cluster, alla compatibilità dei modelli e all'analisi delle licenze.

- Nel Capitolo 5 sarà ripreso il tema implementativo, delineato nel Capitolo 3, descrivendo il processo di integrazione dei modelli di Machine Learning importati all'interno di Elasticsearch. Successivamente, verrà introdotto lo strumento in grado di implementare i processi di inferenza sui dati a disposizione, noto come Ingest Pipeline, mostrando il processo della sua costruzione. Una volta illustrata una possibile esecuzione della pipeline di ingestion, verranno definiti i possibili casi d'uso che coinvolgono il suo utilizzo.

- Nel Capitolo 6 verranno mostrate le metriche utilizzate per la valutazione dello strumento, esponendo, tramite esse, i punti di forza e di debolezza osservati durante lo sviluppo del framework a partire dai risultati ottenuti.

- Nel Capitolo 7 verrà svolta una retrospettiva critica dell'esperienza ottenuta, considerando le lezioni apprese durante lo sviluppo del framework, anche in merito al comportamento dimostrato da Elasticsearch. Verranno, inoltre, illustrati i principali sistemi correlati al motore di ricerca, che possono essere impiegati per la costruzione di un framework alternativo a quello realizzato.

- Nelle conclusioni verrà fornito un riepilogo di tutto il lavoro svolto, descritto nei capitoli precedenti, includendo, con esso, i possibili sviluppi futuri, a partire dal framework realizzato.

CAPITOLO 1

Elasticsearch per la ricerca full-text

Nel primo capitolo di questo elaborato, verrà svolta una panoramica dello strumento principale che verrà utilizzato per la ricerca full-text. Nel dettaglio, verranno descritte la sue principali funzionalità, gli avvenimenti principali del suo sviluppo, la sua architettura, le eventuali problematiche che potrebbero insorgere con il suo utilizzo (split brain, indicizzazione, ricerche inesatte, etc.) e come risolverle. Verrà, inoltre, approfondita la modalità con cui client e server si scambiano le informazioni a vicenda, svolta tramite un insieme di procedure, denominate API REST.

1.1 Che cos'è Elasticsearch

Elasticsearch (Figura 1.1) nasce, ufficialmente, nel 2014 come motore di ricerca open-source che si promette di dare una soluzione al problema di eseguire analisi di dati in maniera performante e orientata a basi di dati molto grandi. La performance viene realizzata non tanto agendo sugli algoritmi di ricerca, già ampiamente sviluppati e discussi a livello mondiale, ma agendo sulla parte infrastrutturale; infatti, uno stesso algoritmo dà una risposta in un certo tempo o nella metà cambiando infrastruttura. Col tempo si è evoluto a database NoSQL con modello dati *Document Store* per implementare funzionalità che permettono di fare istruzioni più vicine ad un base di dati non relazionale (ricerca, aggregazioni etc).

Figura 1.1: Logo Elasticsearch

Il motore di ricerca è scritto in linguaggio Java ed è sviluppato sopra Apache Lucene, una libreria per motori di ricerca full-text sufficientemente complessa. Lucene viene utilizzato internamente per l'indicizzazione e le ricerche, nascondendo la complessità della libreria tramite API RESTful.

Elasticsearch è incluso insieme a tre prodotti che vengono distribuiti come suite dando luogo ad un framework completo:

- *Elasticsearch*: fa da base dati; permette di creare collezioni di documenti nonché di inserire, ricercare e cancellare questi ultimi attraverso chiamate REST.
- *Logstash*: serve per gestire l'auditing, ovvero raccogliere metriche di processo (quanto tempo serve per fare una ricerca da parte di un utente) e fare delle analisi sulle inefficienze dell'applicazione.
- *Kibana*: frontend di Elasticsearch, permette di accedere, in maniera grafica, alle collezioni di dati e ai vari indici dei documenti, eseguire query e creare dashboard personalizzate.

Il software, oltre a svolgere ricerche full-text all'interno di documenti, trova utilizzo in tutti quei campi che riguardano ricerca in siti web, analisi di dati, registrazione di metriche di auditing, analisi di dati aziendali, monitoraggio di performance applicative. Come accennato precedentemente, esso svolge anche la funzione di Document Store distribuito, dove ogni campo viene indicizzato per poter essere ricercato; tutte queste funzionalità descritte vengono raggruppate in un server standalone interrogabile da qualsiasi applicazione per mezzo di API RESTful.

1.2 Un po' di storia

Prima della creazione di Elasticsearch, il suo ideatore, Shay Baron si dedicò allo sviluppo del suo precursore, Compass, costruito sopra Apache Lucene. Lo scopo di Compass era quello di semplificare la ricerca in qualsiasi applicazione Java. Mentre era dedito allo sviluppo della Versione 3.0 del software arrivò alla conclusione che, per fornire una ricerca scalabile, era necessario riscrivere gran parte di Compass. Per questa ragione ricominciò da zero a costruire una soluzione che fosse distribuibile, utilizzando un'interfaccia JSON su HTTP per tutti gli altri linguaggi di programmazione al di fuori di Java; da questa idea nasce la prima versione di Elasticsearch nel 2010 per poi avere la sua prima versione principale nel 2014.

Fino alla Versione 7.10, Elasticsearch utilizzava la licenza open-source Apache 2.0 che regola la possibilità di contribuire alla modifica del software per aggiungere nuove funzioni da parte della community che si è creata col tempo. Nel mentre il motore di ricerca iniziò ad essere utilizzato su più piattaforme di aziende del calibro di Amazon che iniziarono ad offrire ai propri clienti Elasticsearch come servizio a pagamento attraverso Amazon Web Services (AWS). Questo creò un contenzioso dal momento che l'azienda Elastic, proprietaria di Elasticsearch, già offriva un servizio gratuito e non aveva alcuna collaborazione con l'altra azienda; il risultato fu che, nel 2021, a partire dalla versione 7.11, Elastic fece un "relicensing" della propria licenza Apache 2.0 sotto licenza Elastic e Server Side Public Licence, entrambe licenze non open-source. In questo modo Amazon era libero comunque di usare Elasticsearch per sviluppare i propri prodotti, ma non poteva offrirlo come servizio ai clienti. In risposta a questa decisione, Amazon e altre aziende, come CrateDB e Aiven, fecero una "fork" del progetto e continuarono ad usarlo sotto licenza Apache 2.0.

A partire dalla Versione 8 di Elasticsearch è possibile importare modelli PyTorch preallenati; ciò consente di implementare task di Natural Language Processing moderni che sfruttino tutte le potenzialità di un motore di ricerca [Elastic, 2022h]. I casi d'uso più rilevanti sono i seguenti:

- *Language Identification*: identifica in quale lingua è stato scritto un documento.
- *Named Entity Recognition (NER)*: identifica e categorizza parole, nella maggior parte nomi propri, che sono riferite tipicamente a persone, luoghi, oggetti etc. Utilizzata

principalmente per processare ed esplorare grandi collezioni di testi, come articoli di giornale, siti web o enciclopedie, per comprendere il soggetto e raggruppare porzioni di contenuto.

- *Text classification*: associa un'etichetta al testo inserito; trova spazio nella Sentiment Analysis per riconoscere se il sentimento trasmesso in un testo è positivo, negativo o neutro, oppure per riconoscere la tipologia di argomento di discussione (Sport, Economia, Attualità).
- *Zero-shot classification*: rappresenta un task di classificazione dove non è necessario un allenamento del modello.
- *Text embedding*: crea una rappresentazione matematica del testo; può essere utilizzato, ad esempio, per determinare porzioni di testo diversi che hanno significato simile od opposto.
- *Fill-mask*: consente di riempire parole mancanti all'interno di un testo.

1.3 Architettura

L'architettura di Elasticsearch è quella di un tradizionale sistema distribuito composto da un insieme di nodi collegati all'interno di una Computer Network con la possibilità di frammentare i dati per distribuirli sui diversi nodi al fine di garantire maggior parallelismo nell'esecuzione e avere delle repliche che possano garantire una maggiore affidabilità. Unito a ciò sono incluse caratteristiche che riguardano non solo la memorizzazione e la modifica dei dati (frammentazione, replicazione e permessi di lettura/scrittura dei nodi), ma anche come viene garantita la ricerca basata sul contenuto, come avviene l'invio della richiesta al server e il tipo di risposta corrispondente.

1.3.1 Indice invertito

Una caratteristica rilevante dell'architettura di Elasticsearch sta nel come vengono salvati i dati quando viene inserito un documento per consentire una loro ricerca veloce. L'indice invertito (Figura 1.2) consiste in una lista di tutte le parole distinte che compaiono nei documenti dove, a ciascuna parola viene associata una lista dei documenti in cui compare. Attraverso l'utilizzo dell'indice invertito viene creata una mappatura tra le parole contenute in un documento e la loro posizione nel documento e nell'indice. Il risultato di questa azione è quello di un dizionario creato scomponendo le frasi una ad una, parola per parola. Quando viene svolta una ricerca viene stilata una classifica sulla base del posizionamento delle singole parole all'interno del documento, consentendo di svolgere ricerche "content-based" dove vengono riportati, per primi, i documenti più rilevanti sulla base della query scritta dall'utente.

Una problematica preponderante dell'indicizzazione dei documenti è il rischio di indicizzare parole che non forniscono nessuna informazione sul contenuto ma che presentano una frequenza molto alta all'interno di più documenti. Infatti, se l'indicizzazione non viene fatta in maniera controllata filtrando le parole che vengono inserite all'interno della tabella, ma al contrario, tutte le parole distinte presenti nei vari documenti vengono indicizzate, oltre a causare problemi a livello di dimensione della tabella di indicizzazione, si ha anche la presenza di documenti che non sono pertinenti alla specifica ricerca, nella risposta alla richiesta da parte del client. La soluzione a questo problema è quella di generare una lista di parole, che prendono il nome di Stopword (Figura 1.3), che verranno ignorate nella fase di indicizzazione dal momento che sono poco o per niente utili per la ricerca e presentano un'alta frequenza

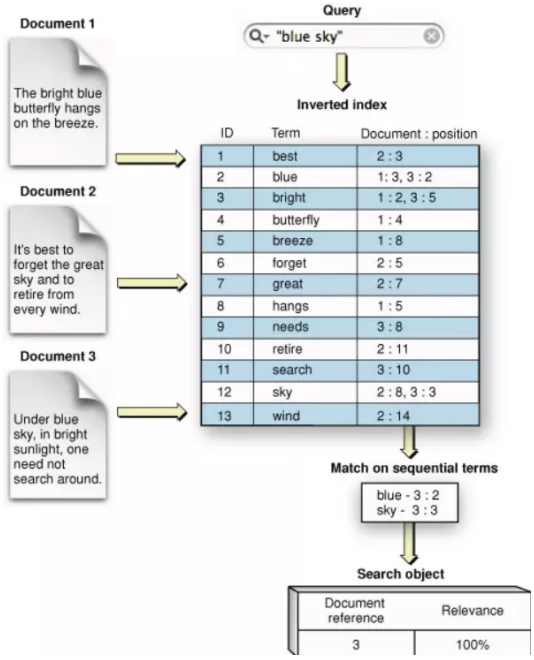

Figura 1.2: Indicizzazione dei documenti e query di ricerca

nei documenti. A questa categoria di parole appartengono articoli determinativi e/o indeterminativi, congiunzioni, preposizioni semplici ed articolate e parole di negazione, in base alla lingua utilizzata per scrivere il contenuto. Anche la lista delle Stopword, denominata Stoplist, deve essere scelta con criterio altrimenti si rischia di eliminare termini utili per la ricerca. Ad esempio, se in "Vitamina A", viene cancellata la "A", perché inclusa tra le stopword, quello che rimane non ha alcun significato.

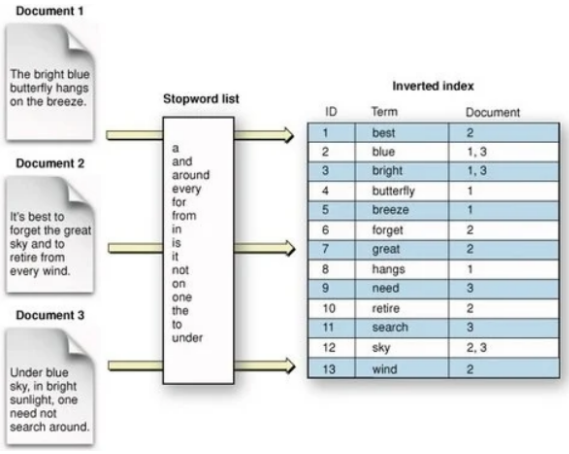

Figura 1.3: Indicizzazione dei documenti senza Stopword

1.3.2 Cluster

Andando in dettaglio alla struttura interna di un tipico cluster Elasticsearch (Figura 1.4), ciascun nodo può essere impostato come nodo che può contenere dati, può svolgere task di Machine Learning, permette di applicare una pipeline di ingestion ai documenti e/o può diventare master. Nello specifico, il nodo si può candidare ad essere master quando il master corrente non è più disponibile; ad esempio, nel caso di un errore hardware del nodo della

rete di comunicazione. Un nodo deve avere almeno un ruolo ma può anche avere più o tutti i ruoli contemporaneamente. Tutti i nodi che possono contenere dati possono contenere al proprio interno più partizioni di dati (shard). L'unica differenza tra shard e repliche è che i primi sono copie primarie mentre le altre no; per il resto sono equamente distribuiti sui vari nodi. In questo modo viene garantito il funzionamento costante del cluster garantendo una disponibilità costante per l'utente nel caso un nodo dovesse guastarsi. Esiste comunque un nodo più importante degli altri (master) che raccoglie tutte le informazioni e i risultati dei sotto-processi eseguiti nei vari nodi e li mette a fattor comune per ottenere il risultato finale.

Esistono due regole fondamentali nel caso di guasto di un nodo:

- C'è un sistema, in Elasticsearch, dove ogni nodo in qualsiasi momento può diventare master, anche per sua presa di posizione e anche se il suddetto nodo è una replica.
- Quando un nodo master precedentemente caduto torna operativo, dopo che è stato eletto uno nuovo, diventa nodo secondario.

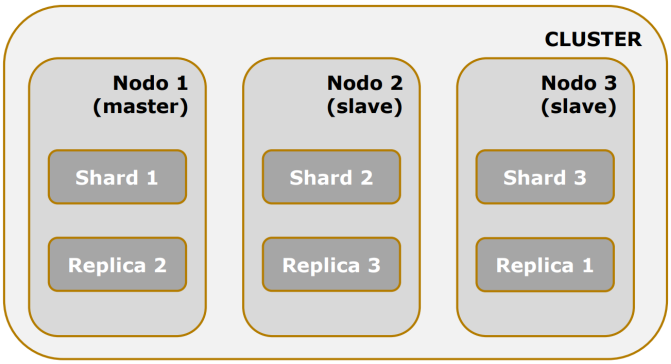

Figura 1.4: Esempio di architettura cluster

Un problema rilevante che è conseguenza di questa configurazione è il problema dello *split brain* (1.5). Prendendo come esempio un cluster con due nodi, rispettivamente, master e slave, che non comunicano tra di loro ma che funzionano entrambi, il nodo master rimane master e considera il cluster composto da un solo nodo, l'altro nodo, che era slave, decide anche lui di essere master, sa di avere una replica, ma sa che la struttura di una replica è una ridondanza di dati; perciò fa diventare shard la replica. La situazione è, quindi, quella di un cluster di due nodi master che agiscono indipendentemente tra loro e che salvano documenti diversi; nel momento in cui si va a ripristinare la configurazione si crea una situazione dove la ricerca di un documento restituisce risultati diversi a seconda di dove viene eseguita. Per risolvere questo problema sono implementate delle regole all'interno di Elasticsearch; più specificatamente:

- solo un nodo alla volta può essere master;
- è possibile configurare il numero di nodi slave che si possono auto-promuovere master (uno di default); è consigliato assegnare un numero dispari.

1.3.3 Accesso ai dati

L'interrogazione dei dati, e in generale delle informazioni del cluster, avviene tramite API che rispettano i vincoli architetturali imposti da REST. Un'*Application Programming Interface* (API) è un insieme di procedure che permettono a due applicazioni distinte di scambiarsi

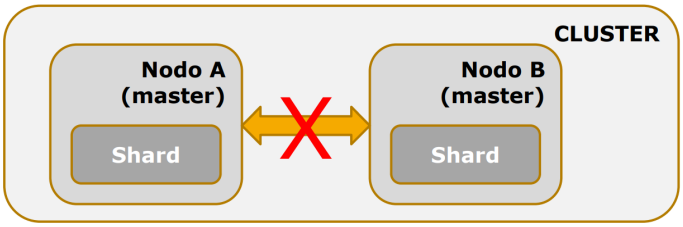

Figura 1.5: Split brain

reciprocamente informazioni. Esse prendono, quindi, le varie richieste da un programma, traducono i messaggi ed eseguono i protocolli necessari per consegnare la richiesta o il messaggio all'applicazione destinataria [TIBCO, 2022]. Un'*API RESTful* è una sottocategoria di procedure che devono essere conformi ai sei vincoli architetturali REST [IBM, 2022].

- *Uniformità dell'interfaccia*: tutte le richieste verso una specifica risorsa devono essere uguali, indipendentemente da dove esse provengono. Lo stesso blocco di dati deve appartenere ad un solo Uniform Resource Identifier (URI) e le risorse devono includere tutte le informazioni di cui il client può aver bisogno, senza eccedere troppo a livello di dimensioni.
- *Disaccoppiamento client-server*: le applicazioni client e server devono essere completamente indipendenti l'una dall'altra; tutto ciò che si deve conoscere lato client è l'URI della risorsa richiesta. Le applicazioni client non possono interagire in alcun modo con le applicazioni server e viceversa. Le applicazioni server non devono modificare quelle lato client, ma devono soltanto trasferire i dati sotto richiesta HTTP.
- *Le API REST sono stateless*: ogni richiesta deve contenere tutte le informazioni necessarie per elaborarla, senza chiedere alcuna sessione lato server e senza l'archiviazione dei dati delle richieste da parte di queste ultime.
- *Memorizzazione nella cache*: le risorse dovono essere memorizzate possibilmente nella cache lato client o server; questa soluzione si pone l'obiettivo di aumentare la scalabilità lato server e di migliorare le prestazioni lato client. Le informazioni relative alla possibilità di memorizzazione devono essere incluse nelle risposte del server.
- *Architettura a livelli*: il client e il server non si connettono direttamente a vicenda, bensì le chiamate e le risposte passano attraverso diversi livelli, come, ad esempio soggetti intermediari. Lo scopo di questa scelta è garantire che né il client né il server riescano a capire se comunicano con l'applicazione finale o con un intermediario.
- *Codice on-demand*: vincolo facoltativo; di solito le informazioni inviate dalle API REST sono statiche ma, in alcuni casi, le risposte possono contenere anche del codice eseguibile la cui esecuzione deve essere svolta on-demand.

Per quanto riguarda Elasticsearch, tramite le API RESTful è possibile ottenere informazioni sugli indici; questi ultimi sono paragonabili alle collezioni di un sistema Document Store qualsiasi. Essi contengono al proprio interno un insieme di documenti correlati e vengono identificati per mezzo di un *Uniform Resource Identifier* (URI). Tra le informazioni in dettaglio degli indici è presente un attributo che può contenere un valore compreso tra *yellow* (giallo) o *green* (verde), che segnala lo stato dell'indice, cioè se esso è ben strutturato secondo determinate regole (le informazioni sono frammentate ed esistono repliche).

Le API REST comunicano attraverso le richieste fornite dal protocollo *HTTP* (Hypertext Transfer Protocol), "mappando" le operazioni *CRUD* (Create, Read, Update, Delete) nelle quattro richieste [Rajesh Kumar, 2018]:

- *GET*: richiesta di lettura da parte del client ad un determinato indice; essa ritorna come risultato tutti i documenti che sono rilevanti per la specifica ricerca.
- *POST*: il client invia un insieme di dati chiedendo al server di elaborarli; essa viene utilizzata principalmente per creare e modificare documenti.
- *PUT*: esattamente come la POST, invia dati per inserire e/o aggiornare un indice o un documento; la differenza tra questa richiesta e quella precedente si trova nella proprietà di idempotenza della PUT. La suddetta proprietà garantisce che ripetere la stessa identica richiesta più di una volta non porterà a nessun cambiamento.
- *DELETE*: permette di cancellare un documento o una collezione dell'URI associato.

L'accesso al cluster può essere svolto anche tramite linguaggio Java attraverso la libreria Elasticsearch JAVA contenente quattro componenti base necessarie per gestire le richieste.

CAPITOLO 2

Text Mining

In questo secondo capitolo verrà approfondita, nel dettaglio, la tipologia di task NLP che verrà maggiormente svolto, ovvero il Text Mining. Per comprendere meglio in cosa consiste il Text Mining è doveroso descrivere anche la branca dell'Intelligenza Artificiale di cui fa parte, ovvero il Natural Language Processing. Per questo motivo, verrà presentata un'introduzione al tema dell'NLP, descrivendo il suo scopo, le fasi principali della sua esecuzione, le eventuali problematiche che possono insorgere e i suoi campi applicativi. A questo punto, verranno utilizzate queste nozioni primarie per comprendere meglio l'ambito del Text Mining. Sarà, quindi, possibile procedere con la definizione degli obbiettivi, delle fasi del processo, delle possibili problematiche e delle sue applicazioni nel mondo reale.

2.1 Natural Language Processing

Gli esseri umani utilizzano il linguaggio come metodo di comunicazione per parlare, leggere e scrivere, in maniera del tutto naturale; tramite il linguaggio si prendono decisioni, si sviluppano pensieri, si trasmette la conoscenza, etc. Con l'avvento dell'Intelligenza Artificiale, gli esperti si sono chiesti se fosse possibile utilizzare il linguaggio naturale per comunicare con la macchine; più precisamente, se fosse possibile addestrare i computer per comprendere una richiesta, in linguaggio umano, ed eseguirla anche rispondendo all'operatore nella sua lingua. Il *Natural Language Processing* (NLP) è il sottocampo dell'Intelligenza Artificiale che si pone l'obbiettivo definito precedentemente, ovvero addestrare i computer a capire ed elaborare il linguaggio umano. A livello pratico, il compito principale consiste nel programmare i computer, che normalmente processano dati strutturati, per analizzare ed elaborare enormi quantità di dati non strutturati.

A prima vista, il Natural Language Processing può risultare una tematica nata recentemente per via dell'obbiettivo imposto. In realtà, le sue origini risalgono alla fine degli anni '40, dove le ricerche si sono concentrate, principalmente, sul *Machine Translation*; l'obbiettivo era l'utilizzo delle macchine per la traduzione automatica da una lingua sorgente ad un'altra lingua. Il successo iniziale, ottenuto in questo campo applicativo, spinse le ricerche successive verso l'integrazione dell'NLP con altri campi.

Il primo campo in cui l'NLP venne integrato fu quello dell'Intelligenza Artificiale (fine anni '60 - fine anni '70). I ricercatori si concentrarono sul mondo della conoscenza e sul ruolo che esso aveva, nella costruzione e manipolazione delle rappresentazioni dei concetti. Successivamente a questa fase, durante la quale non vennero implementati sistemi pratici, le ricerche si incentrarono sull'utilizzo della logica per la rappresentazione della conoscenza e il ragionamento nell'IA. Questa fase prese il nome di fase grammatica-logica e si colloca

tra la fine degli anni '70 e la fine degli anni '80. Successivamente, a partite dagli anni 90, le ricerche si mossero verso l'utilizzo del lessico e del corpus; il primo consiste in un insieme di parole e locuzioni di una lingua o di una sua parte; il secondo, invece, è un insieme ampio e strutturato di testi leggibili dalla macchina. Infine, nel decennio corrente, sono stati introdotti algoritmi di apprendimento automatico per elaborare il linguaggio naturale.

2.1.1 Fasi dell'NLP

Qualsiasi task di Natural Language Processing consiste in una sequenza di quattro fasi principali. A partire da un testo non strutturato in input, viene restituita una singola rappresentazione del significato contenuto all'interno di una frase, di un paragrafo o di un documento intero.

Come si può osservare nella Figura 2.1, i passi logici del Natural Language Processing sono i seguenti:

- *Elaborazione morfologica*: questa fase ha il compito di spezzare il testo in parti più piccole chiamate *token*, corrispondenti a paragrafi, frasi e parole.
- *Analisi sintattica*: questa fase si pone due obbiettivi. Il primo è quello di verificare se una frase è *ben formata* oppure no. Una frase si dice ben formata se essa è conforme alle regole della grammatica formale; se la frase non rispetta tale proprietà, essa viene rifiutata dall'analizzatore sintattico. In caso di esito positivo, si procede con il secondo obbiettivo, ovvero scomporre la frase in una struttura che mostri le relazioni sintattiche tra le diverse parole.
- *Analisi semantica*: lo scopo di questa fase è di trarre il significato esatto dal testo, noto anche come "significato del dizionario". Il testo viene controllato dall'analizzatore semantico per vedere se ha significato e viene estratto il suo significato corretto. L'analisi lessicale risulta molto più complessa, rispetto alla fase precedente; ciò è dovuto alle potenziali ambiguità nel testo. L'ambiguità semantica è un problema più complesso rispetto all'ambiguità sintattica, il quale viene risolto facilmente attraverso il tagging *Part-Of-Speech* (POS) con accuratezza elevata.
- *Analisi pragmatica*: A partire dalla rappresentazione realizzata dall'ultima analisi, l'analisi pragmatica adatta gli oggetti e/o gli eventi reali, che esistono in un dato contesto, di riferimenti di oggetti ottenuti durante l'analisi semantica.

2.1.2 Problema dell'ambiguità

Nella parte precedente, è stata menzionata una problematica piuttosto rilevante che è presente non solo nell'analisi sintattica e semantica, ma anche in tutte le altre fasi. La problematica a cui si fa riferimento è l'ambiguità; essa può essere descritta come la capacità di comprendere le frasi in più di un modo.

L'obbiettivo della disambiguazione del senso delle parole consiste, quindi, nel determinare quale significato della parola è attivato dall'uso di quest'ultima in un particolare contesto. Prima di definire quali sono le possibili soluzioni al problema dell'ambiguità, risulta necessario categorizzare i diversi tipi di ambiguità.

Queste categorie a cui si fa riferimento sono le seguenti:

- *Ambiguità lessicale*: quando una parola può essere trattata come verbo, aggettivo, sostantivo, o qualsiasi altra categoria grammaticale, si ha un caso di ambiguità lessicale.

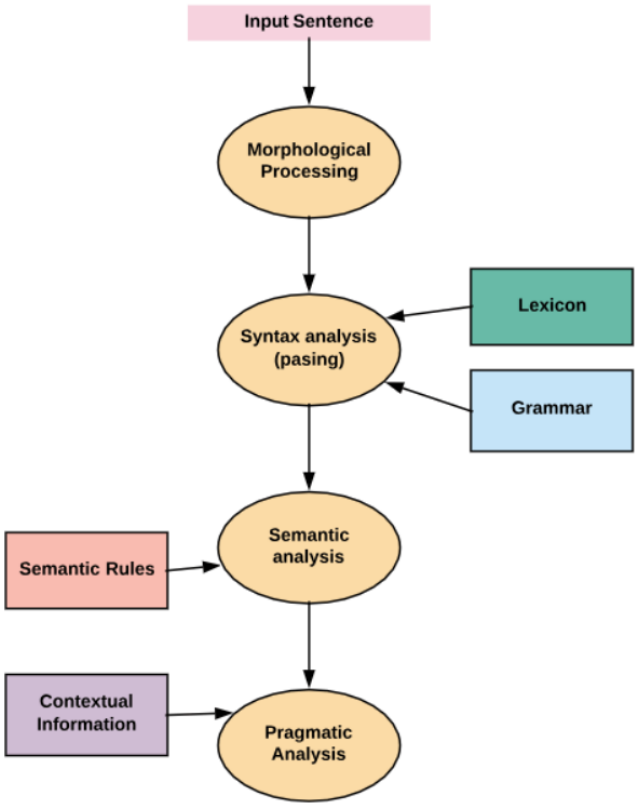

Figura 2.1: Fasi del Natural Language Processing

- *Ambiguità sintattica*: l'ambiguità sintattica si verifica quando una frase viene analizzata in modi diversi.
- *Ambiguità semantica*: questo tipo di ambiguità si verifica quando il significato delle parole stesse può essere male interpretato.
- *Ambiguità anaforica*: se il discorso include l'utilizzo di entità anaforiche (lui, lei, egli, loro, per la lingua italiana), potrebbero insorgere casi di ambiguità anaforica.
- *Ambiguità pragmatica*: l'ambiguità pragmatica fa riferimento al contesto in cui la frase non è specifica, ovvero quando il contesto di una frase dà molteplici interpretazioni.

Come accennato precedentemente, l'ambiguità sintattica viene risolta utilizzando tagger Part-Of-Speech. Questo tipo di tagging svolge una classificazione delle parole in base alla funzione grammaticale associata, in maniera analoga all'analisi grammaticale.

Per quanto riguarda il problema dell'ambiguità semantica, che prende il nome di *Word Sense Disambiguation*, essa risulta più complicata del problema precedente. Questo task richiede l'utilizzo di un dizionario (WordNet, Thesaurus, etc.) per specificare le parole affette da ambiguità (omonimia, sinonimia, polisemia, etc.) ed un corpus di test annotato che registra tutti i sensi corretti.

Il problema del WSD è presente in quasi tutte le applicazioni del Natural Language Processing; tra queste possiamo elencare:

- *Text Mining e Information Extraction*: il WSD aiuta il sistema di raccolta intelligente a effettuare il flagging delle parole corrette. Esso permette, con ciò, di effettuare un'analisi accurata del testo.

- *Traduzione automatica*: è l'applicazione più ovvia che coinvolge il WSD. Infatti, la scelta lessicale delle parole, che hanno traduzioni distinte per i diversi sensi, è fatta dal WSD. I sensi nella traduzione automatica sono rappresentati come parole nella lingua di destinazione.

- *Information Retrieval*: l'Information Retrieval (IR) è una branca dell'NLP che si occupa dell'organizzazione, memorizzazione, recupero e valutazione di informazioni da archivi di documenti testuali. Il fine di queste procedure è quello di assistere gli utenti nel trovare le informazioni richieste; il software restituisce solo i documenti che potrebbero contenere la risposta alla domanda. In questo caso, il WSD ha il compito di risolvere le ambiguità delle query fornite al sistema.

- *Lessicografia*: nell'ambito della lessicografia, il WSD fornisce raggruppamenti empirici di senso e indicatori contestuali statisticamente significativi. Il WSD e la lessicografia possono operare insieme, in maniera iterativa, perché la lessicografia moderna è basata sul corpus.

2.1.3 Campi applicativi dell'NLP

Nell'ultimo decennio, il Natural Language Processing si sta sviluppando sempre maggiormente e molte aziende stanno investendo in Ricerca e Sviluppo su questo ramo dell'IA. Grazie a ciò, si sono sviluppati, e si stanno generando, sempre più campi applicativi che coinvolgono questa disciplina. Tra i più importanti possiamo elencare:

- *Traduttori Automatici*: è il campo applicativo più importante dell'NLP; la Machine Translation è il processo di traduzione di una lingua sorgente o di un testo in un'altra lingua. Le traduzioni possono essere da due lingue particolari (Sistemi bilingue) o tra qualsiasi coppia di lingue (Sistemi multilingue).

- *Lotta allo spam*: utilizzare l'NLP all'interno di filtri antispam consente di rilevare e filtrare le e-mail indesiderate, un problema, ormai, comune nell'attuale scenario. I sistemi di filtraggio dello spam, che rappresentano la soluzione principale in questo campo, possono essere sviluppati, tramite l'NLP, sulla base dei problemi di falsi positivi e falsi negativi.

- *Riassunto automatico*: a partire da documenti di testo lunghi, i sistemi di riassunto automatico sono in grado di creare un riassunto breve e accurato del testo. Ciò permette di individuare le sole informazioni importanti in meno tempo rispetto al documento originario.

- *Question Answering*: il Question Answering è un'altra applicazione importante dell'NLP che coinvolge, principalmente, i motori di ricerca. Questi ultimi, infatti, svolgono un ottimo lavoro nel ricercare le informazioni, ma sono tuttora molto carenti nel rispondere a domande in linguaggio naturale. Questo task si concentra sulla costruzione di sistemi che siano in grado di rispondere automaticamente alle domande, poste in linguaggio naturale. Sistemi di questo calibro hanno la capacità di tradurre le frasi, scritte dagli esseri umani, in una rappresentazione interna, per la generazione di risposte valide.

- *Chatbot*: con il passare del tempo, i sistemi di Question Answering si sono sempre più spostati verso i Chatbot, una delle attuali frontiere dell'NLP, dell'Intelligenza Artificiale e della Data Science. Un Chatbot è un software che simula ed elabora le conversazioni umane per consentire agli utenti di interagire con i dispositivi, come se stessero parlando con un'altra persona. I Chatbot più semplici sono programmi che rispondono a una

semplice query con una singola riga, mentre quelli più sofisticati, come gli assistenti digitali (Cortana, Alexa, Siri, etc.), raccolgono ed elaborano informazioni per fornire livelli crescenti di personalizzazione.

- *Sentiment Analysis*: come si può intuire dal nome, la Sentiment Analysis è usata per identificare i sentimenti contenuti nelle parole, nelle frasi e/o nei documenti, sopratutto dove le emozioni non sono espresse esplicitamente. La Sentiment Analysis viene utilizzata dalle aziende principalmente per identificare l'opinione dei clienti online su prodotti e servizi offerti. Le aziende possono giudicare la loro reputazione generale dai post dei clienti, al fine di limitare gli aspetti negativi e di potenziare quelli positivi.

2.2 Definizione del Text Mining

Data una panoramica sul vasto campo che comprende il Natural Language Processing, e tenendo a mente i concetti principali definiti, si passa alla descrizione dell'ambito più strettamente legato alla presente tesi, ovvero il Text Mining. Il Text Mining, per sua definizione, ha il compito di esaminare grandi collezioni di documenti per scoprire nuove informazioni o per migliorare la risposta alle domande di ricerca. Facendo riferimento alla Text Analytics, il Text Mining è un'Intelligenza Artificiale che sfrutta l'NLP per normalizzare e strutturare qualsiasi testo non strutturato. Il risultato consiste in un insieme di dati utilizzabili per algoritmi di Machine Learning o per un task di analisi [TIBCO, 2020].

Andando più nello specifico, il Text Mining ha il compito di identificare fatti, relazioni, e asserzioni nascoste nell'enorme agglomerato di dati testuali. Tali informazioni, una volta estrapolate, sono convertite in dati strutturati; questi ultimi vengono utilizzati per l'analisi o per la rappresentazione delle informazioni con l'utilizzo di visualizzazioni grafiche (diagrammi, tabelle, grafi, etc.)

2.2.1 Pipeline di processi

Come accennato precedentemente, il processo di Text Mining riceve in input un testo non strutturato, con il fine di ricavare informazioni di valore che, altrimenti, rimarrebbero nascoste e impossibili da identificare. Il task di Text Mining consiste in un insieme di processi sequenziali, i quali vengono riportati nella Figura 2.2. Seguendo la pipeline della loro esecuzione, i processi comprendono [Neelam TyagiMayi, 2021]:

- *Raccolta dei dati*: per svolgere un task di Text Mining, serve sapere quali sono i dati di interesse e dove è possibile estrarli. Dopo aver individuato e scelto le sorgenti di dati, si procede con la raccolta dei dati. In questa fase è altamente probabile che i dati siano molto rumorosi e contengano informazioni non conformi allo scopo finale del task. Per questo motivo, al termine del caricamento, viene svolta una fase di Data Preprocessing. Molto spesso capita che, al termine del caricamento dei dati e prima delle fase successiva, viene svolta una prima pulizia superficiale degli stessi, sopratutto quando essi presentano un evidente livello di rumore.

- *Text Preprocessing*: I dati caricati nella fase precedente, oltre ad avere rumore, si presentano in un formato semi-strutturato o del tutto non strutturato. Le macchine sono in grado di processare solo dati di natura strutturata. Il Text Preprocessing è il processo che permette di risolvere questa problematica, per mezzo di una serie di sottoprocessi di cui esso si compone. Tali sottoprocessi sono i seguenti:

  - *Tokenizzazione*: il testo viene spezzato in elementi più piccoli chiamati "token" (frasi in parole, paragrafi in frasi, etc.).

  - *Filtraggio*: vengono rimosse tutte le parole ridondanti nel testo nonché quelle che non contribuiscono all'analisi perché prive di informazione (Stopword).
  - *Stemming*: è una forma semplificata dell'analisi morfologica; esse estrae la radice da ogni parola ("stem"), tagliando le estremità di quest'ultima.
  - *Lemmatizzazione*: è simile alla tecnica precedente, ma cerca di comprendere il significato delle parole in base al loro contesto.
  - *Linguistic processing*: include processi di Part-Of-Speech Tagging, Word Sense Disambiguation e Struttura Semantica.

- *Indicizzazione*: terminato il processo di Text Preprocessing, il testo strutturato viene caricato all'interno di database per garantire un accesso veloce e l'archiviazione delle informazioni. I software utilizzati comprendono l'utilizzo dell'indice invertito il quale, come accennato nel capitolo precedente, contribuisce allo svolgimento delle ricerche rilevanti per le parole inserite.

- *Data Mining*: a partire dai risultati ottenuti dalle fasi precedenti, si passa all'applicazione della tecnica di Text Mining, il processo centrale di tutta la pipeline. Le tecniche comunemente utilizzate, per via della loro efficacia, sono le seguenti:

  - *Information Extraction*: è il processo di estrazione di informazioni, che risultano significative, dai dati testuali. Nel dettaglio, identifica ed estrae entità, attributi e relazioni tra entità; queste vengono salvati in un database per recuperi futuri e accessi efficienti.
  - *Information Retrieval*: il task di Information Retrieval si occupa, a partire da un insieme di frasi o parole, dell'estrazione di modelli rilevanti alla ricerca. Gli algoritmi utilizzati servono per tenere traccia e seguire il comportamento degli utenti, oltre che per la raccolta di informazioni rilevanti.
  - *Categorization*: consiste nell'esecuzione di un processo di apprendimento supervisionato, dove i testi, in linguaggio naturale, sono ordinati per categorie di argomenti basati sul contenuto informativo. I documenti di testo vengono, quindi, analizzati per trovare gli argomenti o la corretta indicizzazione per ciascuno di essi.
  - *Clustering*: si occupa di identificare e localizzare strutture interne di un insieme di documenti, organizzandole in "cluster" (sottogruppi). La creazione di cluster significativi, a partire da dati testuali non etichettati, è il processo più complicato all'interno del task. Inoltre, il Clustering viene utilizzato per la pre-elaborazione di dati che sono destinati ad altre tecniche di Text Mining.
  - *Summarization*: genera automaticamente una versione compressa di un testo più lungo, contenente le informazioni principali utili all'utente. Il processo esamina fonti diverse di dati testuali e mette insieme riassunti di testi ricchi di informazioni. Il significato e l'intento dei documenti iniziali rimangono invariati.

- *Data Analysis*: è l'ultimo processo della pipeline che riguarda l'estrazione di informazioni e conoscenza dai dati elaborati. È il passo più importante perché consente all'utente di ricercare i soli dati concernenti il al tipo di ricerca che intende svolgere. Se l'utente ha come obbiettivo l'identificazione di pattern nascosti nei dati, questi ultimi vengono rilevati durante questa fase, attraverso rappresentazioni grafiche, all'interno di apposite dashboard.

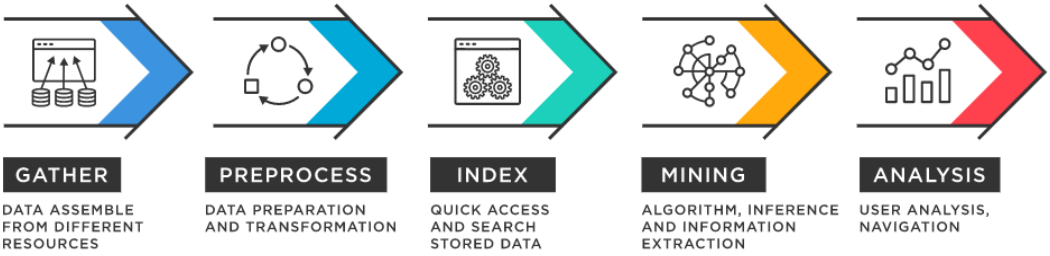

Figura 2.2: Pipeline dei processi di Text Mining

2.2.2 Problematiche

Oltre alla problematica del Word Sense Disambiguation, precedentemente descritta, i problemi che riguardano il Text Mining non sono relativi alla tecnologia, bensì, all'ambito di applicazione. Infatti, l'utilizzo di dati di natura privata, come le cartelle cliniche personali, può causare problematiche di natura etica. Viviamo in un'epoca in cui la tutela della privacy è una tema molto scottante, e c'è un dibattito aperto su quali dati sono, o meno, di natura privata. L'utilizzo del Text Mining senza cognizione di causa è collegato a questo tipo di dibattiti etici, giustificato maggiormente dal comportamento scorretto di determinate aziende. Può capitare, difatti, che le aziende dichiarino di effettuare task di Text Mining per uno scopo ma, in realtà, potrebbero usare i dati per altri fini non dichiarati. Questo uso improprio dei dati rappresenta una minaccia molto rilevante alla privacy degli individui; ciò comporta una maggiore attenzione nel rispetto di tutte le leggi sulla privacy, come il GDPR, da parte degli organizzatori di campagne di Data Analysis.

2.2.3 Applicazioni del Text Mining

Al giorno d'oggi molti più settori e aree stanno impiegando, nelle loro attività, strumenti e tecniche di Text Mining; oltre alle aziende troviamo università, sanità, organizzazioni, piattaforme di social media etc. Nello specifico, le attività in cui il Text Mining si è inserito sono le seguenti:

- *Assistenza clienti*: utilizzare i feedback dei clienti, all'interno della Text Analytics, permette alle aziende di migliorare, ulteriormente, la soddisfazione e l'esperienza dei clienti. I clienti possono rilasciare feedback in diversi modi, ad esempio tramite recensioni online, ticket, profili di social media, chatbot, etc. Questi dati vengono raccolti e utilizzati per task di Text Mining e Sentiment Analysis, con l'obbiettivo di identificare e risolvere i problemi principali individuati dagli utenti.

- *Rilevamento di frodi*: il Text Mining trova spazio anche nell'individuazione e prevenzione delle frodi. Gli strumenti forniti dal Text Mining sono diventati fondamentali per le aziende assicurative e finanziarie anche per elaborare, con maggiore efficienza, richieste di garanzia e/o assicurazione.

- *Gestione del rischio*: nell'ambito della gestione del rischio, l'utilizzo del Text Mining ha l'obbiettivo di raccogliere informazioni in merito ai trend industriali e mercati finanziari. I dati vengono, di solito, recuperati da articoli di giornale e/o report analitici. Ad esempio, le società bancarie utilizzano tali approccio per gestire gli investimenti sui vari settori, attraverso la raccolta di dati bancari.

- *Business Intelligence*: in maniera simile all'assistenza clienti, molte organizzazioni stanno sfruttando le tecniche di Text Mining per la Business Intelligence. Le informazioni

ricavate riguardano il comportamento degli acquirenti e le tendenze di mercato. Il Text Mining è utile anche per ottenere vantaggi competitivi rispetto alla concorrenza. Esso consente, infatti, di analizzare i punti di forza, di debolezza, le opportunità e le minacce dell'azienda e dei concorrenti.

- *Analisi dei social media*: nel mondo dei social media, gli strumenti forniti dal Text Mining risultano utili nell'analisi di post, "like", commenti e tendenze degli utenti sulle discussioni. Gli stessi strumenti sono anche utili per comprendere le relazioni e il comportamento degli individui su specifici prodotti, argomenti e contenuti online. Lo scopo finale del Text Mining nella Social Network Analysis è quello di scoprire e utilizzare, a proprio vantaggio, le tendenze e i contenuti virali che coinvolgono il pubblico.

- *Manutenzione*: in questo campo applicativo, il Text Mining garantisce l'automazione dei processi di "Decision Making" attraverso l'identificazione di pattern correlati a guasti. Infatti, è in grado non solo di fornire un quadro completo di operazioni, ma anche di guidare le funzionalità di prodotti e servizi. Garantisce, quindi, una manutenzione reattiva e preventiva dei processi di un sistema.

- *Lotta allo spam*: come accennato nella sezione precedente, la lotta allo spam è diventata un problema onnipresente nella quotidianità. Anche il Text Mining contribuisce attivamente nel fermare la diffusione delle attività malevole, inviate via e-mail. L'obbiettivo è quello di ridurre il rischio di attacchi informatici attraverso filtri antispam ottimizzati per rimuovere email potenzialmente dannose.

- *Assistenza sanitaria*: nell'ambito della medicina, le tecniche di Text Mining sono già molto utilizzate in applicazioni biomediche. Le stesse tecniche stanno trovando spazio nell'assistenza sanitaria, dove i dati utilizzati contengono informazioni di pazienti, visite mediche, prescrizioni mediche, etc. Queste informazioni, unite a quelle contenute nelle cartelle cliniche, incentivano le iniziative di ricerca, la riduzione degli errori sulle prescrizioni mediche e la riduzione dei costi [Uzma Raja, Tara Mitchell, Timothy Day, J. Michael Hardin, 2008].

CAPITOLO 3

Elasticsearch: prime operatività

Terminata la panoramica su Elasticsearch e su ciò che riguarda il task di Text Mining, in questo capitolo verrà introdotta, nel dettaglio, una possibile applicazione del suddetto task. In particolar modo, verrà trattata la realizzazione di una ricerca a livello sintattico basata sul contenuto dei documenti. Verrà svolta una selezione del dataset, da cui verranno estratti i documenti testuali, per poi definire una collezione di documenti, tramite un indice Elasticsearch; esso consentirà il salvataggio e l'indicizzazione dei documenti. La collezione verrà, in seguito, utilizzata per eseguire le possibili ricerche di documenti rilevanti per il testo inserito.

3.1 Descrizione del dataset

Il dataset che è stato selezionato (Figura 3.1) contiene 152.000 articoli di giornale provenienti da due testate giornalistiche italiane, *La Repubblica* e *Il Giornale*. Le informazioni sono contenute all'interno di file *Comma-Separated Values* (CSV), dove ogni record rappresenta un articolo; ciascuna riga contiene, a sua volta, due attributi che rappresentano il titolo e il corpo della notizia. Tutti gli articoli prodotti da entrambe le testate sono archiviati in due file *.csv* distinti.

Il dataset è stato, originariamente, progettato per svolgere un task di generazione di testo italiano chiamato *CHANGE-IT*, durante la campagna di evaluation Elvalita 2020. Dal momento che le due testate appartengono a spettri politici opposti, rispettivamente, "LaRepubblica" di sinistra e "IlGiornale" di destra, l'obbiettivo era quello di svolgere, partendo da un insieme di articoli di una delle due testate, un task di trasferimento di stile, per mezzo di un sistema automatico; lo scopo del task era di riscrivere tutte le testate degli articoli provenienti da *Il Giornale* sullo stile di quelli scritti da *La Repubblica* e viceversa [Gabriele Sarti, 2020].

È possibile consultare il dataset tramite il link:

`https://live.european-language-grid.eu/catalogue/corpus/7373`.

mentre la sua descrizione è riportata all'indirizzo:

`https://huggingface.co/datasets/gsarti/change_it`.

Successivamente, gli articoli sono stati divisi in 126.142 per la parte di training e i restanti per il testing, mantenendo separate le due categorie di articoli salvandoli su file diversi e garantendo un bilanciamento equo sul numero di articoli provenienti da entrambe le testate giornalistiche.

	headline	full_text
0	Mughini e la bufera social per la frase su Ron...	Giampiero Mughini è stato al centro di una buf...
1	Monti lodato da Obama?Era tutta una bufala	Roma - Una telefonata inopportuna, una citazio...
2	Moglie e figli sgozzati in casa: fermato il ma...	Un delitto efferato, spietato, che ha creato "...
3	Business Advantage, assicurare le Pmi in modo ...	Lorenzo Corti Si chiama Business Advantage ed ...
4	Fare un bancomat? Al Sud è un'impresa	Anche la mappa dei bancomat vede una frattura ...
5	Level, il nuovo low cost da Vienna a Malpensa	New entry all'aeroporto di Milano Malpensa con...
6	In Olanda alloggio gratis per i giovani che as...	Ospitalità in cambio di altruismo. Un accordo ...
7	In arresto per corruzione il sindaco di Acireale	Cinque persone sono finite in carcere e tre ai...
8	Delitto Scazzi, Sabrina Misseri sarà liberata ...	Sabrina Misseri uscirà prima del previsto dal ...
9	Dall'impresa di San Siro al tonfo: il Benevent...	Proprio quando l'incubo stava diventando favol...

	headline	full_text
0	James Lovelock: "Dieci anni fa ero certo che l...	L'ULTIMO REGALO DI JAMES LOVELOCK è un elisir ...
1	D'Asaro Biondo (Google): "Spinta all'export pe...	"LE ASSICURO che il livello della tasse che no...
2	Ucraina, presidente chiede riunione straordina...	KIEV - Il Parlamento ucraino, su richiesta del...
3	"Fatevene una ragione: gli antichi romani eran...	Il cartone animato è del 2014, ma le polemiche...
4	Cina, ritrovato un dinosauro di specie sconosc...	HANNO fatto esplodere un'area rocciosa per cos...
5	A Ezio Mauro il Premiolino 2017. Riconosciment...	MILANO - L'ex direttore di Repubblica Ezio Mau...
6	Cambia ancora il codice Antimafia: obbligatori...	ROMA- Cambia ancora il codice Antimafia. Per a...
7	Pordenone, ritrovati i tre ragazzi dispersi su...	ANDREIS (PORDENONE) - Paura e poi sollievo per...
8	Commissione adozioni: "Casi sospetti tra bimbi...	Nella lunga e complicata vicenda delle adozion...
9	Reato di tortura, terzo ok dal Parlamento, il ...	Con 195 voti a favore, 8 contrari e 34 astenut...

Figura 3.1: Struttura del dataset

3.2 Struttura del cluster locale

Risulta doveroso approfondire, in maniera dettagliata, la struttura generale del cluster che verrà utilizzato in questa fase. Il cluster (Figura 3.2) è stato realizzato all'interno di una Macchina Virtuale *Ubuntu*, utilizzando il servizio offerto da *Windows Subsystem for Linux* (WSL), all'interno di un computer locale dove è stato installato il software *Docker* per la gestione di container. La *Virtual Machine* (VM) Ubuntu è stata collegata a Docker e allo strumento *docker-compose*, per consentire la costruzione e l'esecuzione di un'applicazione multi-container, ovvero il cluster in esame.

Ogni nodo rappresenta un singolo container contenente una Versione Elasticsearch (Versione 8.3) o, in alternativa, una Versione di Kibana (Versione 8.3). Il cluster è composto da quattro nodi, tre contenenti Elasticsearch e uno Kibana; il primo insieme di nodi può diventare master e, come già spiegato nel primo capitolo, al momento della creazione del cluster, i corrispettivi nodi si candidano per quel ruolo e solo uno viene eletto master. Nel caso di guasto del master, i due nodi restanti si candidano per prendere il suo posto. Inoltre, tutti i nodi possono contenere i dati; in questo modo, anche se i dati venissero caricati in un nodo specifico, attraverso le impostazioni delle frammentazioni e repliche, Elasticsearch si occuperà di formare i frammenti dei dati, replicarli e distribuirli su tutti i nodi del cluster. Questa azione garantisce maggior velocità in lettura e affidabilità nel mantenere i dati.

Il nodo contenente Kibana ha il compito di interfacciarsi con i nodi Elasticsearch per consentire l'esecuzione delle operazioni sui dati tramite interfaccia grafica. Attraverso questo software è possibile creare collezioni di documenti, denominate indici, per il salvataggio e l'accesso di documenti, attraverso l'invocazione delle API REST; lo strumento permette, anche, di creare dashboard per svolgere un'analisi descrittiva dei dati. L'accesso ai dati attraverso Kibana viene svolto tramite browser; nella implementazione locale corrente, l'accesso avviene tramite la porta di default, ovvero la porta 5601.

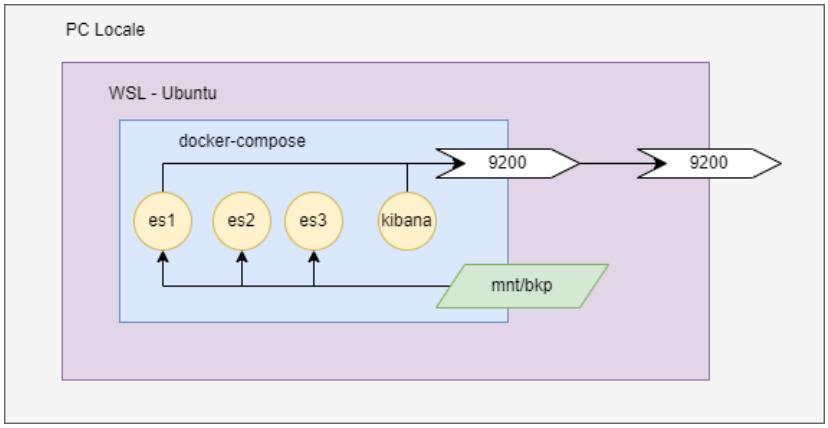

Figura 3.2: Struttura del cluster su macchina locale

3.3 Creazione dell'indice

Dopo la descrizione del dataset e del suo contenuto informativo, e dopo aver analizzato la struttura del cluster, la prossima fase consiste nel caricamento dei dati, all'interno di Elasticsearch, attraverso una specifica collezione, definita da un URI univoco; questa collezione prende il nome di indice. Un *indice* Elasticsearch, da non confondere con l'indice invertito, è come un Database in un modello relazionale tradizionale, ovvero una porzione di spazio dove salvare documenti correlati. La struttura dell'indice utilizzata per gli articoli di giornale viene riportata nella Figura 3.3.

I campi principali, riportati all'interno dell'attributo *mapping*, sono i seguenti:

- *dynamic*: permette di impostare la tipologia di mapping dinamico per i documenti inseriti all'interno dell'indice. Potrebbe capitare che vengano inseriti documenti che contengono alcuni campi sconosciuti al mapping. A seconda del valore associato a *dynamic*, vengono eseguite diverse azioni; in particolare, i valori possibili e le corrispettive azioni, sono i seguenti:
  - *true*: se viene inserito un documento con un nuovo campo, Elasticsearch determinerà automaticamente il suo *datatype* e lo aggiungerà al mapping;
  - *false*: il documento viene inserito, ignorando il campo non indicizzato;
  - *strict*: viene lanciata un'eccezione all'inserimento di un nuovo campo.
- *properties*: vengono indicati i campi contenuti nel documento, ogni campo viene definito con un nome univoco e, all'interno di esso, viene specificato il tipo di valore associato.
- *analyzer*: all'interno di ciascun campo è possibile specificare la tipologia di analyzer, e quindi, il tipo di "tokenizzazione" e normalizzazione dei blocchi di testo per la creazione dell'indice invertito. Esistono diversi analyzer "built-in", ma è possibile, in ogni caso, definire analyzer personalizzati per svolgere indicizzazioni ad hoc [Elastic, 2022f].

Utilizzare l'analyzer di default per indicizzare un documento non è sempre la scelta migliore; infatti esso non garantisce, quasi mai, una corretta eliminazione delle Stopword; questa problematica riguarda, in particolar modo, documenti scritti in una lingua diversa dall'inglese e con un'alta frequenza media di Stopword all'interno dei documenti, come nel caso della lingua italiana. Se l'indicizzazione non viene svolta correttamente, si ottiene una ricerca sul contenuto che risulta inesatta; dal momento che la maggior parte delle query contiene molteplici Stopword, vengono restituiti documenti, insieme a quelli rilevanti, i quali

```
paper_mapping = {
    "settings": {
    "number_of_shards": 1,
    "number_of_replicas": 0
    },
    "mappings": {
        "dynamic": "strict",
        "properties": {
            "headline": {
                "type": "text",
                "analyzer": "italian"
            },
            "body": {
                "type": "text",
                "analyzer": "italian"
            }
        }
    }
}
```

Figura 3.3: Struttura dell'indice

non contengono le parole rilevanti alla ricerca, ma soltanto le Stopword. Vista l'alta frequenza di queste ultime, i documenti irrilevanti vengono riportati tra i risultati di valore più alto. Per implementare una Stoplist adeguata, esistono analyzer "built-in" fatti apposta per risolvere questo problema; nel loro insieme, è presente quello progettato per la lingua italiana. Dal momento che i documenti sono scritti in italiano corretto e non presentano Stopword di altre lingue, utilizzare l'analyzer "built-in" per la lingua italiana risulta la scelta corretta per implementare ricerche di documenti rilevanti [Elastic, 2022a].

Attraverso la struttura definita nella Figura 3.3, sono stati creati due indici distinti, uno per i documenti provenienti da "IlGiornale" e uno per quelli di "LaRepubblica". Il risultato di queste operazioni viene restituito da Elasticsearch attraverso una richiesta di stampa del mapping di uno specifico indice (Figura 3.4).

A questo punto, bisogna estrarre ciascun record dal dataset e convertirlo in un documento *JSON*. Questa azione viene svolta definendo una struttura di tipo dizionario (Figura 3.5), contenente un attributo per il titolo e uno per il corpo; essa verrà utilizzata in fase di inserimento dei documenti all'interno degli indici, definiti in precedenza. Nella Figura 3.6 viene riportato il comando di creazione della struttura di tipo dizionario appena definita.

Una volta definita la struttura per tutti i documenti, è possibile procedere con il caricamento dei suddetti all'interno degli indici Elasticsearch precedentemente definiti. Questa operazione viene svolta attraverso una chiamata API *bulk* (Figura 3.7) dove viene indicata la collezione di documenti da inserire insieme all'indirizzo del cluster.

Il risultato di questa sequenza di operazioni riporta due indici Elasticsearch distinti, ma definiti dalla stessa struttura. Essi contengono tutti i documenti del dataset, opportunamente indicizzati, per eseguire una ricerca sintattica efficiente ed efficace.

3.4 Prime interrogazioni su Elasticsearch

La prossima, e ultima, fase di questa sezione riguarda l'implementazione delle prime ricerche eseguibili da parte di un utente. Ad esempio, quando l'utente vuole ricercare notizie riguardanti un determinato tema (Ambiente, Sport, Finanza, etc.), includendo altre variabili

```
{
 "ilgiornale-train": {
  "mappings": {
   "dynamic": "strict",
   "properties": {
    "body": {
     "type": "text",
     "analyzer": "italian"
    },
    "headline": {
     "type": "text",
     "analyzer": "italian"
    }
   }
  }
 }
}
```

```
{
 "repubblica-train": {
  "mappings": {
   "dynamic": "strict",
   "properties": {
    "body": {
     "type": "text",
     "analyzer": "italian"
    },
    "headline": {
     "type": "text",
     "analyzer": "italian"
    }
   }
  }
 }
}
```

Figura 3.4: Struttura degli indici Elasticsearch


```
{
 "headline": "Moglie e figli sgozzati in casa: fermato il marito per omicidio",
 "full_text": "Un delitto efferato, spietato, che ha creato \"angoscia\" perfino tra egli inquirenti. La donna, Cristina Omes , di 38 anni, e i due piccoli, Giulia e Gabriele, di 5 anni e di 20 mesi, sono stati sgozzati e sui loro corpi ci sono numerose altre lesioni che fanno supporre un accanimento senza alcuna pietà. Per il triplice omicidio, che ha distrutto una famiglia intera sconvolgendo gli abitanti di Motta Visconti , è stato fermato dopo un lungo interrogatorio il marito Carlo Lissi . Formalizzate le accuse, l'uomo è stato portato nel carcere di Pavia. Malgrado avesse inscenato una rapina nel tentativo di sviare le indagini, Lissi è stato ripetutamente interrogato dai carabinieri finché è crollato. Quando gli hanno
```


Figura 3.5: Struttura del dizionario per documenti

```
dict_ilgiornale = ds_ilgiornale.to_dict('records')
dict_repubblica = ds_repubblica.to_dict('records')
```

Figura 3.6: Comando di creazione della struttura del dizionario

```
try:
    res_ilgiornale = helpers.bulk(es, ilgiornale, request_timeout=400)
except Exception as e:
    print(e)
    pass
```

Figura 3.7: API per l'inserimento dei documenti su Elasticsearch

come città, nomi propri, etc., egli riceverà una lista di tutti i documenti pertinenti al testo inserito. Per merito dell'analyzer *italian*, nel calcolo della pertinenza, verranno escluse tutte le Stopword inserite dall'utente nella query.

Dal momento che le uniche istanze disponibili in ciascun documento corrispondono al titolo e al corpo dell'articolo le possibili modalità di ricerca sono le seguenti:

- *Ricerca basata sul titolo* (Figura 3.8): vengono restituiti tutti i documenti, per intero; questi contengono, nel titolo, le parole digitate dall'utente; è stata definita una variante di tale ricerca dove viene restituito solo il campo *headline* per ciascun documento.

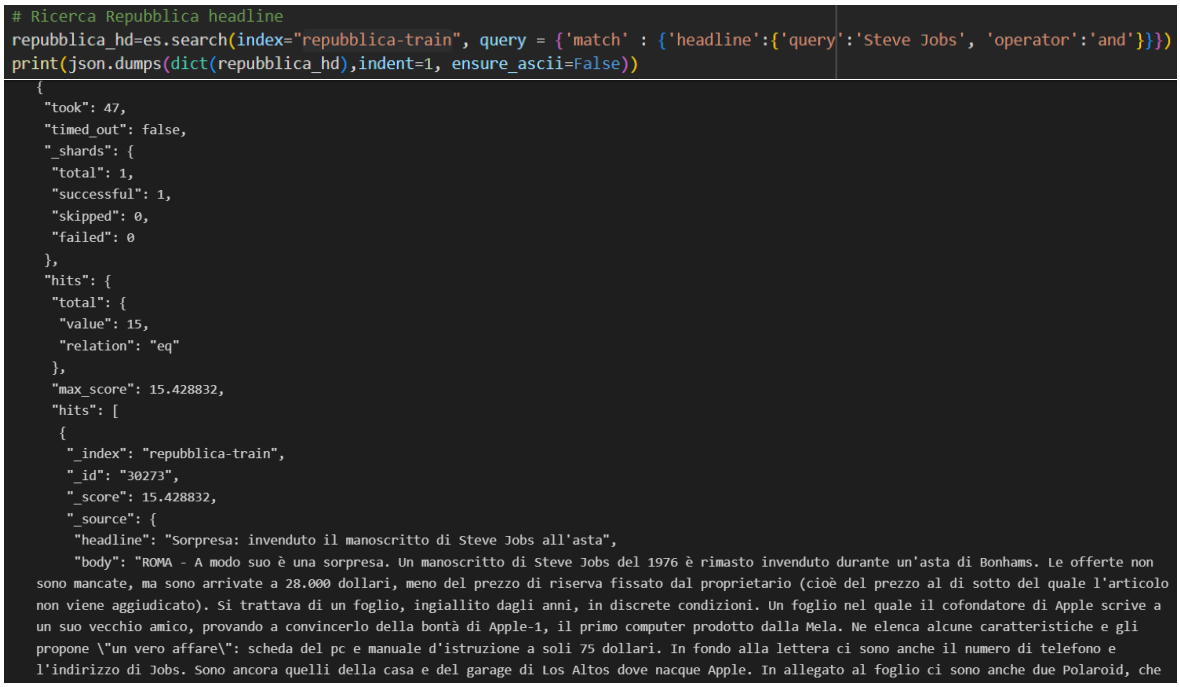

Figura 3.8: Query e risultato della ricerca basata sul titolo

- *Ricerca basata sul corpo* (Figura 3.9): il risultato di questa ricerca è analogo a quello precedente (vengono restituiti i documenti per intero); la differenza riguarda il calcolo della rilevanza. Infatti, quest'ultima, viene calcolata considerando solo le parole all'interno del campo *body*.

- *Ricerca basata su titolo e corpo* (Figura 3.10): corrisponde all'unione delle due ricerche precedenti; la rilevanza viene, quindi, calcolata prendendo in considerazione le parole incluse nel titolo e nel corpo del documento. Anche per questo tipo di ricerca è stata implementata una variante che restituisce solo il titolo dell'articolo di giornale.

```
# Ricerca IlGiornale body
ilgiornale_body=es.search(index="ilgiornale-train", query = {'match' : {'body':{'query':'moglie e figli uccisi', 'operator':'and'}}})
print(json.dumps(dict(ilgiornale_body),indent=1, ensure_ascii=False))
```


```
    "hits": {
     "total": {
      "value": 85,
      "relation": "eq"
     },
     "max_score": 14.511002,
     "hits": [
      {
       "_index": "ilgiornale-train",
       "_id": "47520",
       "_score": 14.511002,
       "_source": {
        "headline": "Ardea, uccide la moglie e si spara davanti ai figli",
        "body": "Roma - Omicidio-suicidio ad Ardea, sul litorale romano. Un uomo di 62 anni, originario di Ancona, al culmine dell'ennesima lite ha
   ucciso a colpi di pistola la moglie, di 15 anni più giovane, poi ha rivolto l'arma contro se stesso e si è ucciso davanti ai figli. Tragedia
   famigliare Tragedia familiare ad Ardea: al culmine di una lite un uomo ha ucciso a colpi di arma da fuco la moglie dalla quale si stava separando poi
   si è ucciso. Intorno alle 13 - spiegano i carabinieri che sono intervenuti - in un'abitazione di Ardea, in via Terni, un uomo, 62 anni, originario
   della provincia di Ancona, al culmine di una lite con la moglie, 47enne romana, ha impugnato un'arma da fuoco, tra le numerose che possedeva, tutte
   regolarmente denunciate e detenute, e ha esploso numerosi colpi contro la donna, uccidendola. Poi ha puntato l'arma contro di sé e ha sparato: è morto
   poco dopo. La coppia era in fase di separazione. Sul posto sono intervenuti i carabinieri della stazione di ardea e della compagnia di Anzio."
       }
      },
   ...
     }
    ]
```


Figura 3.9: Query e risultato della ricerca basata sul corpo

```
# Ricerca IlGiornale headline body
giornale_hb=es.search(index="ilgiornale-train", query = {'bool':{'should':[{'match' :
{'headline':'Festa Brodetto Fano'}}, {'match':{'body': 'Pesce Azzurro brodetto'}}]}})
print(json.dumps(dict(giornale_hb),indent=1, ensure_ascii=False))
```


```
   {
    "took": 53,
    "timed_out": false,
    "_shards": {
     "total": 1,
     "successful": 1,
     "skipped": 0,
     "failed": 0
    },
    "hits": {
     "total": {
      "value": 2412,
      "relation": "eq"
     },
     "max_score": 32.521297,
     "hits": [
      {
       "_index": "ilgiornale-train",
       "_id": "6718",
       "_score": 32.521297,
       "_source": {
        "headline": "Dall'elicottero al brodetto, la giornata di Berlusconi in Molise",
        "body": "Il volo in elicottero sopra gli Appennini, poi il bagno di folla tra le strade di Termoli, infine il brodetto alla termolese. Chiuso il
   secondo giro di consultazioni, Silvio Berlusconi si dedica alle prossime Regionali e riparte dal Molise, dove è andato per tirare la volata a Donato
   Toma, candidato governatore alle elezioni del prossimo 22 aprile. La giornata del Cavaliere è iniziata in elicottero: \"Molise... Sto arrivando!\",
   aveva scritto su Facebook, pubblicando una foto mentre ripassava il discorso osservando le montagne ancora innevate. \"Voliamo in elicottero con il
   presidente Silvio Berlusconi verso il Molise per evitare che i 5 Stelle conquistino la loro prima Regione\", ha scritto poco dopo il fedelissimo
```


Figura 3.10: Query e risultato della ricerca basata sul titolo e corpo

CAPITOLO 4

Abilitazione delle funzionalità di Machine Learning

Nel capitolo precedente è stata illustrata l'implementazione della ricerca sintattica tramite Elasticsearch. Attraverso l'utilizzo delle funzionalità principali del software sono stati caricati e adattati diversi documenti da un dataset, per poi svolgere le suddette ricerche. A partire da questa sezione in poi, verrà trattata la realizzazione della seconda tipologia di ricerca, ovvero la ricerca semantica. In particolare, in questo capitolo verranno indicate tutte le operazioni NLP realizzabili tramite le funzionalità di Machine Learning permesse da Elasticsearch. In merito a questa tematica, verranno presi in considerazione modelli di Machine Learning preallenati, disponibili all'interno della piattaforma Hugging Face. Successivamente, verrà descritto il percorso per l'abilitazione delle suddette funzionalità, inclusa la gestione delle licenze e della compatibilità dei modelli.

4.1 Machine Learning in Elasticsearch

Come già accennato nel primo capitolo, tra le novità presenti nelle ultime versioni di Elasticsearch (8.x) è inclusa la possibilità di utilizzare modelli di Machine Learning preallenati esternamente. Questi ultimi riguardano specifici task del Natural Language Processing e vengono utilizzati per potenziare il motore di ricerca [Elastic, 2022g]. Tra i modelli di ML che hanno avuto successo nell'ambito dell'NLP sono presenti quelli basati su *Google BERT*. Il *Bidirectional Encoder Representations from Transformers* è un modello di elaborazione "Transformer-based" per linguaggio naturale. Nato nel 2018 grazie a Google, BERT è progettato per il preaddestramento di modelli di Deep Learning bidirezionali, con l'obbiettivo di ricercare informazioni all'interno di un testo non etichettato. Una volta terminato l'addestramento, ai modelli BERT ottenuti può essere applicato un Fine-Tuning per consentire il loro utilizzo in una grande varietà di applicazioni NLP. Dal momento del suo rilascio, BERT ha presentato ottimi risultati su diversi task NLP. Per tale ragione è diventato il punto di riferimento della maggior parte delle tecniche moderne di NLP e dei sistemi che ne consentono l'esecuzione, tra i quali è presente anche Elasticsearch [Jacob Devlin, Ming-Wei Chang, Kenton Lee, Kristina Toutanova, 2018].

Per quanto riguarda il suddetto motore di ricerca, BERT viene utilizzano come base per la costruzione delle feature di Machine Learning; esse supportano lo schema di tokenizzazione e i "Transformer" del modello di interfaccia standard di BERT. Elasticsearch usa Libtorch per effettuare predizioni e incorporare i modelli allenati; per consentire il loro utilizzo all'interno di Elasticsearch, essi assumono una rappresentazione di tipo TorchScript. Una volta importato all'interno del cluster, il modello può essere utilizzato per svolgere inferenze sui dati in

ingresso. Nelle prossime sottosezioni verranno illustrati i principali ambiti applicativi dei modelli all'interno di Elasticsearch.

4.1.1 Information Extraction

Come si può intuire dal nome, il compito di un task di Information Extraction è quello di individuare il contenuto informativo da testo non strutturato. Questa procedura permette l'esecuzione della ricerca tramite entità e la classificazione delle parole. L'annotazione semantica è una variante dell'Information Extraction a cui viene aggiunto un rafforzamento semantico. Il suddetto task estende l'Information Extraction attraverso il raggruppamento e la connessione di entità con il loro significato semantico a partire da un grafo della conoscenza [Ontotext, 2022].

Elasticsearch implementa il task di Information Extraction attraverso l'esecuzione di due sottotask principali; il primo svolge una classificazione delle parole in specifiche categorie, esso prende il nome di Named Entity Recognition. Il secondo, denominato Fill-Mask, svolge la predizione di una o più parole mancanti all'interno di un testo non strutturato.

Questi due sottotask verranno illustrati più in dettaglio nelle prossime sottosezioni.

Named Entity Recognition

Il Named Entity Recognition (NER) è un sottotask appartenente all'insieme di applicazioni dell'Information Extraction. Il suo obbiettivo, come già accennato prima, è quello di localizzare e classificare entità nominate all'interno di un testo non strutturato. Le entità nominate sono parole chiave che fanno riferimento a categorie predefinite, come persone, luoghi, organizzazioni, valute, etc. Il NER è un ottimo strumento per svolgere compiti specifici dell'Information Extraction. Esso è in grado, difatti, di identificare le informazioni principali nel testo, di aggiungere struttura e conoscenza del contenuto e di processare grandi collezioni di testo, come articoli di giornale, pagine di enciclopedia, siti web, etc. Tutte queste funzionalità garantiscono una comprensione del soggetto e il raggruppamento di porzioni del contenuto più semplici da realizzare [Elastic, 2022d].

La procedura eseguita da un processo di NER è composta da due processi principali [Christopher Marshall, 2019]:

- *Rilevamento di un'entità nominata*: in questo primo processo viene ricercata una parola, o un insieme di esse, che forma un'entità nominata. Per definizione, un'entità nominata (Named Entity) corrisponde ad un oggetto reale (persona, luogo, organizzazione, etc.), il quale può essere descritto con un nome proprio. Ciascuna parola all'interno dell'entità rappresenta un token (ad esempio Università Politecnica delle Marche è una stringa di quattro token rappresentanti un'entità).

- *Categorizzazione dell'entità*: una volta individuata l'entità nominata, essa viene associata ad un'etichetta che rappresenta la categoria ad essa associata. Nella Tabella 4.1 sono presenti tutte le categorie che un modello NER basato su BERT può associare alle entità.

Nella Figura 4.1 viene riportato un esempio di esecuzione di Named Entity Recognition di una frase.

Fill-Mask e Question Answering

Il secondo sottotask, appartenente al ramo dell'Information Extraction, si occupa di svolgere predizioni di una o più parole mancanti da un testo. Questa applicazione prende il nome di Masked Language o Fill-Mask; essa utilizza il contesto delle parole mascherate

Categoria	Descrizione
O	Identifica tutte le parole che non sono Named Entity (non vengono marcate in output)
B-MIS	Inizio di un'entità generale subito dopo un'altra entità generale
I-MIS	Entità generale
B-PER	Inizio di nome proprio di persona subito dopo un altro nome di persona
I-PER	Nome proprio di persona
B-ORG	Inizio di un nome di un'organizzazione subito dopo un altro nome di organizzazione
I-ORG	Nome proprio di un'organizzazione
B-LOC	Inizio di un nome di luogo subito dopo un altro nome di luogo
I-LOC	Nome proprio di un luogo

Tabella 4.1: Categorie delle entità utilizzabili dal modello BERT-base

Figura 4.1: Esecuzione di un modello per Named Entity Recognition

per svolgere la predizione delle parole. L'esecuzione di questa tipologia di task non richiede l'utilizzo di dati etichettati; infatti, i modelli vengono allenati "mascherando" un insieme di parole all'intero delle frasi e ci si aspetta che il modello indovini quali sono le parole nascoste. Il Masked Language può essere utilizzato per svolgere test semplici e veloci del modello selezionato. Per svolgere il test viene fornito in input un testo non strutturato contenente un placeholder (["MASK"]) per indicare al modello dov'è la parola mancante da inserire [HuggingFace, 2022a].

Il Question Answering è l'ultimo sottotask dell'Information Extraction implementato da Elasticsearch. Il suo obbiettivo è quello di utilizzare il contenuto informativo di un documento per fornire risposta a domande riguardati il testo tokenizzato. In base all'input e all'output forniti esistono diversi tipi di Question Answering; tra i principali possiamo individuare [HuggingFace, 2022b]:

- *Extractive Question Answering*: le risposte vengono estratte dal contesto. Quest'ultimo può essere un testo, una tabella o un documento HTML.
- *Open Generative*: il modello genera un testo automaticamente, sulla base del contesto fornito.
- *Closed Generative*: non viene fornito alcun tipo di contesto; la risposta viene generata completamente da parte del modello.

Il caso d'uso principale del Question Answering riguarda l'automazione delle risposte alle Frequently Asked Questions (FAQ). Attraverso il contesto estratto a partire da una base di conoscenza, il modello è in grado di fornire le risposte alle domande principali dei clienti.

4.1.2 Classificazione del testo

La seconda tipologia di task NLP realizzabili su Elasticsearch riguarda la classificazione delle parole, delle frasi o di tutto il testo, a seconda del tipo di etichetta associabile. Il software è in grado di svolgere tre tipi di classificazione, elencati a partire dal livello informativo più generale, fino a quello più specifico elaborato dal modello.

I modelli di *Language Identification* riconoscono in quale lingua è scritto il testo elaborato. Più il testo in input è lungo e più accuratamente il modelli riusciranno a identificare la lingua; per alcune di esse è sufficiente inserire testi con poche parole (50 parole circa). Sono, invece, necessarie sezioni di testo più ricche per l'identificazione di tutte quelle lingue che presentano parole e forme simili (lingue latine). Se il modello non riesce a riconoscere la lingua, viene restituito un'etichetta di default: "zxx". Il suddetto valore può essere modificato se il modello viene utilizzato in una pipeline di ingestion, cambiando etichetta nel valore desiderato. Una volta costruito il feature set, il modello utilizza Unicode per facilitare il riconoscimento della lingua. Se, ad esempio, il testo contiene segni diacritici (apostrofi, accenti e altri segni che modificano), il modello userà quel tipo di informazione per l'identificazione del testo. I segni diacritici sono tutti i segni che vengono aggiunti ad una lettera per modificare la pronuncia delle parole o distinguere il significato di quelle simili. Al momento della creazione di un cluster Elasticsearch con le funzionalità ML attivate, è già presente un modello di Language Identification (lang_ident_model_1) utilizzabile.

Mentre i task di Language Identification rimangono limitati alla sola identificazione della lingua, la *classificazione del testo* può essere utilizzata in diversi ambiti. Difatti, essa svolge sempre un'analisi del testo, ma con l'obiettivo di assegnare la classe che meglio descrive il testo in esame. Le classi utilizzate dipendono dal modello e dal dataset usati per l'allenamento. A seconda del numero di classi definite, esistono due tipologie principali di classificazione; queste sono:

- *Classificazione binaria*: il testo viene associato ad una delle due classi scelte;
- *Classificazione multi-classe*: il testo viene associato ad una o più classi, le quali sono presenti in numero maggiore di due.

Questa tipologia di classificazione risulta utile per analizzare il testo per marcatori di sentimenti positivo o negativo per svolgere un task di Sentiment Analysis. Un altro esempio di applicazione consiste nel classificare il testo per determinare l'argomento trattato al suo interno ("Sport", "Politica", "Intrattenimento").

Nel caso in cui non si volesse utilizzare o non è presente l'insieme delle classi per l'allenamento del modello, Elasticsearch permette di eseguire modelli per la *Zero-Shot text classification*. Questa tipologia di task permette di classificare un testo direttamente senza svolgere alcun tipo allenamento. Le classi da utilizzare vengono passate al momento del deploy del modello o prima della sua esecuzione. I modelli utilizzati per questo task sono stati allenati su un dataset di grandi dimensioni; attraverso tale allenamento, essi hanno ottenuto una comprensione generale della lingua. La Zero-Shot classification permette, quindi, di classificare ed analizzare il testo anche in assenza di dati di training sufficienti per un modello di classificazione. Risulta, altresì, un task molto flessibile perché consente di aggiustare le classi durante l'esecuzione delle inferenze.

4.1.3 Ricerca e confronto del testo

L'ultima categoria di funzionalità di Machine Learning utilizza rappresentazioni matematiche per svolgere ricerche basate su testo o per confrontare parti di testo distinte. Il *Text Embedding* produce le suddette rappresentazioni matematiche, che prendono il nome

di *embedding*, e trasforma il testo in un array di valori numerici, denominati vettori. Il task viene svolto partendo dal presupposto che pezzi di testo con significato simile presentano rappresentazioni simili. Sulla base di ciò, è possibile determinare sezioni di testo diverse che sono semanticamente simili, diverse, oppure opposte, attraverso una funzione matematica di similarità. Il task svolge la sola funzione di produzione degli embedding; quando un embedding viene creato può essere salvato in un campo *dense_vector* e utilizzato al momento della ricerca. Un esempio di applicazione è quello di utilizzare il vettore in una ricerca basata su kNN per ottenere capacità di ricerca semantica.

4.1.4 Utilizzo di modelli esterni

Oltre alla possibilità di allenare modelli al proprio interno, Elasticsearch permette anche il deploy di modelli preallenati per uno specifico task. In questo processo Kibana e Eland collaborano insieme per preparare e gestire i modelli esterni. Elasticsearch supporta tutti i modelli che sono conformi al modello standard di BERT e utilizza l'algoritmo di tokenizzazione WordPiece. L'insieme delle architetture supportate è il seguente:

- *BERT*;
- *BART*;
- *DPR bi-encoders*;
- *DistilBERT*;
- *ELECTRA*;
- *MobileBERT*;
- *RoBERTa*;
- *RetriBERT*;
- *MPNet*;
- *SentenceTransformers bi-encoders*.

Ogni modello preallenato che supporta una di queste architetture può essere importato su Elasticsearch tramite Eland.

Ci sono diversi modi per usare le feature NLP con Elastic Stack. Una volta stabilito la tipologia di task NLP da svolgere, è necessario scegliere un modello allenato appropriato. Il metodo più semplice è quello di usare un modello a cui è stato applicato un Fine-Tuning per il tipo di analisi da eseguire. Ci sono modelli e dataset disponibili per specifici task NLP su Hugging Face (come nel caso in questione) [Elastic, 2022b].

4.2 Abilitazione delle funzionalità

Dopo aver svolto una panoramica sull'integrazione del Machine Learning all'interno di Elasticsearch, in questa sezione descriviamo la procedura di abilitazione delle suddette funzionalità. Nel capitolo precedente era stata fornita una descrizione del cluster utilizzato per le prime funzionalità. A partire dalla struttura in questione, l'abilitazione delle funzionalità consiste nell'impostare almeno un nodo del cluster con label *ml*. In questo modo, il suddetto nodo sarà in grado di eseguire modelli di Machine Learning, oltre alle altre funzioni permesse

dalle altre label associate. Nel caso in questione tutti i nodi saranno abilitati all'esecuzione dei modelli Transformer.

Nell'implementazione di queste funzionalità è sorta una problematica relativa all'occupazione di memoria. Infatti, gli indici creati precedentemente, sommati ai modelli importati, hanno causato un degrado delle prestazioni della macchina locale, rendendo impossibile l'esecuzione delle operazioni. Per questo motivo è risultato necessario utilizzare una Virtual Machine (VM) esterna per la creazione del cluster e per il salvataggio dei dati all'interno di Elasticsearch. La macchina virtuale è stata costruita utilizzando il servizio fornito da Google Cloud Platform (GCP), il quale consente la creazione di VM in modo semplice e veloce. Il tipo di pagamento imposto per l'utilizzo è proporzionale all'utilizzo effettivo delle risorse della macchina. La struttura del cluster all'interno della VM in GCP viene riportata nella Figura 4.2. Il cluster è stato costruito utilizzando *docker-compose* a partire dalla configurazione del cluster precedente, apportando alcune modifiche ai nodi e aggiungendo un meccanismo di backup dei dati.

La struttura appare leggermente diversa da quella definita precedentemente; difatti, il cluster contiene soltanto tre nodi Elasticsearch senza il nodo Kibana, il quale è presente nella macchina locale. La modifica più importante in questa struttura riguarda il modo in cui il suddetto nodo è in grado di interfacciarsi con i nodi Elasticsearch, seppur presenti in due macchine distinte. Nel cluster utilizzato precedentemente, il nodo Kibana comunicava con i nodi Elasticsearch attraverso la porta 9200 e utilizzando la rete interna messa a disposizione da *docker-compose*. Nel cluster corrente l'esposizione dei nodi verso l'esterno viene effettuata attraverso due passaggi. Il primo passaggio, di configurazione, permette al demone Docker di esporre la porta 9200 del container che ospita un nodo Elasticsearch attraverso la porta 9200 della Virtual Machine. Esso consiste, di fatto, in un Port Forwarding tra un container e la macchina virtuale ospitante. Come configurazione base, le macchine virtuali in GCP non sono esposte su Internet (il loro indirizzo IP non è pubblico). Risulta, quindi, necessaria una fase di configurazione di un secondo Port Forwarding tra la macchina locale e la VM; questo secondo passaggio viene svolto dal comando *gcloud*.

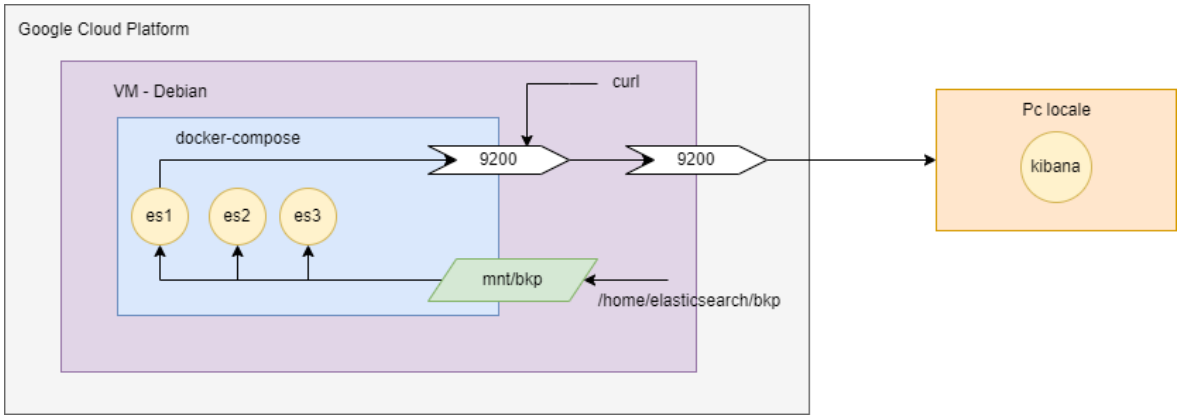

Figura 4.2: Struttura cluster con funzionalità di ML

Per mezzo di questa funzione, una volta accesa la macchina virtuale su GCP, il cluster verrà creato, se non ancora presente, e avviato tramite appositi script lanciati dalla macchina locale. Una volta che il cluster sarà operativo, sempre tramite script, verrà svolto il Port Forwarding della porta 9200 della macchina cloud sulla corrispondente nella macchina locale. In questo modo, una comunicazione impostata sulla porta 9200 locale è direzionata verso la macchina cloud. A questo punto rimane soltanto da creare e/o eseguire un nodo Kibana nella macchina locale senza apportare alcuna modifica alle impostazioni di connessione. Secondo la logica descritta precedentemente, quando Kibana esegue un'interrogazione sulla porta

9200 locale, la risposta verrà fornita dal cluster su macchina cloud. La stessa logica vale anche per l'interfacciamento tramite codice Python, utilizzato nel capitolo precedente. Per mezzo di questa architettura, è possibile utilizzare Elasticsearch in tutte le sue funzioni, incluse le funzionalità di Machine Learning, senza l'effetto collaterale provocato dall'occupazione di memoria in locale. Verranno sfruttate le risorse della macchina virtuale in cloud per sostenere i carichi computazionali e di memoria necessari per i modelli di Machine Learning.

Una caratteristica importante in merito al Port Forwarding implementato riguarda quanti e quali nodi vengono coinvolti nella comunicazione. Infatti, questo tipo di comunicazione con Elasticsearch permette di interfacciarsi con un solo nodo; d'altro canto, se venisse utilizzata un'applicazione che fa un'ingestion di dati, è preferibile dividere questi ultimi e caricarli su più nodi. Tuttavia, questa scelta non causa alcuna problematica rilevante dal momento che, dopo il caricamento dei dati, Elasticsearch si occupa di distribuire questi ultimi sugli altri nodi.

Quando si decide di spegnere la macchina GCP, una volta terminate tutte le azioni all'interno di Elasticsearch, la connessione stabilita con la macchina locale non viene salvata. Perciò, in tutti i riavvii successivi, è necessario rieseguire tutti gli script per ristabilire il Port Forwarding, oltre a quelli per l'attivazione del cluster.

4.3 Gestione della compatibilità

Dopo aver svolto tutti i passaggi descritti precedentemente, è possibile usufruire del nuovo cluster per importare ed eseguire i modelli di Machine Learning all'interno di Elasticsearch. Tutti i modelli importati nel caso in esame sono progettati per svolgere task di Named Entity Recognition e provengono dalla piattaforma Hugging Face. I modelli scelti per la tipologia di documenti da analizzare sono:

- *dslim/bert-base-NER*;
- *philschmid/distilroberta-base-ner-conll2003*;
- *dbmz/bert-base-italian-cased*;
- *Davlan/distilbert-base-multilingual-cased-ner-hrl*.

I primi due modelli vengono citati nella documentazione Elasticsearch nella sezione riguardante i modelli di terze parti compatibili. Gli ultimi due, invece, sono il risultato di un Fine-Tuning a partire dal modello BERT. Il *bert-base-italian-cased* è specifico per la lingua italiana mentre il *Davlan/distilbert-base-multilingual-cased-ner-hrl* include diverse lingue, oltre a quella inglese, tra cui l'italiano. Nonostante questi ultimi due sembrano i più adatti alla tipologia di documenti scelti, è possibile soltanto importarli in Elasticsearch. Gli esperimenti svolti nella fase di utilizzo, attraverso una Ingest Pipeline, portano a validare la tesi secondo cui gli unici modelli eseguibili sono tutti quelli riportati all'interno della documentazione di Elasticsearch. Nel caso di NER è possibile eseguire i primi due modelli citati e i Fine-Tuning degli stessi per la lingua inglese e per l'alfabeto arabo. La scelta dei modelli da utilizzare è, quindi, ricaduta sui primi due modelli più generali, ma che presentano, comunque, un buon riconoscimento di entità.

Per caricare i modelli su Elasticsearch, utilizzando Python, occorre scaricarli dalla piattaforma Hugging Face, sotto forma di oggetto *TransformerModel*, fornito dalla libreria *eland* (Figura 4.3).

Una volta ottenuti i modelli, occorre salvarli in una cartella locale in modo da acquisire, oltre ai file rappresentanti il modello ML, anche i vocabolari associati. Queste due componenti sono necessarie per consentire il corretto caricamento di qualsiasi modello all'interno di

```
# Caricamento di un modello da Hugging Face
tmbert = TransformerModel("dslim/bert-base-NER", "ner")
tmrobert = TransformerModel("philschmid/distilroberta-base-ner-conll2003", "ner")

Downloading: 100%|██████████| 59.0/59.0 [00:00<00:00, 1.71kB/s]
Downloading: 100%|██████████| 829/829 [00:00<00:00, 257kB/s]
Downloading: 100%|██████████| 208k/208k [00:00<00:00, 538kB/s]
Downloading: 100%|██████████| 2.00/2.00 [00:00<00:00, 1.00kB/s]
Downloading: 100%|██████████| 112/112 [00:00<00:00, 45.5kB/s]
Downloading: 100%|██████████| 413M/413M [01:59<00:00, 3.63MB/s]
```

Figura 4.3: Import dei modelli di Machine Learning da Hugging Face

Elasticsearch. Una volta ottenuti i file, si procede con il deploy sul cluster, definendo i modelli sotto forma di oggetto *PyTorchModel* e, a seguire, caricando le due tipologie di file associati ai modelli (Figura 4.4).

```
# Caricamento del modello in TorchScript in ES
ptmbert = PyTorchModel(es, tmbert.elasticsearch_model_id())
ptmbert.import_model(model_path=model_path, config_path=None, vocab_path=vocab_path, config=config)
ptmrpbert = PyTorchModel(es, tmrobert.elasticsearch_model_id())
ptmrobert.import_model(model_path=model_path, config_path=None, vocab_path=vocab_path, config=config)
```

Figura 4.4: Caricamento dei modelli di Machine Learning su Elasticsearch

Al termine del caricamento, sarà possibile utilizzare entrambi i modelli all'interno di Elasticsearch, non prima di averli avviati da Kibana, attraverso una richiesta API apposita (Figura 4.5). Elasticsearch mantiene salvato lo stato del modello anche dopo un riavvio successivo del cluster; pertanto, sarà sufficiente utilizzare l'API di avvio del modello una sola volta, finché non si decide di fermare l'esecuzione.

Un altro dettaglio importante da tenere in considerazione quando si utilizza un modello ML riguarda l'ammontare di risorse necessarie per la sua esecuzione. Prendendo come esempio il modello *dslim/bert-base-NER*, esso necessita di 1GB di memoria centrale per la sua esecuzione. Nel caso in esame di un cluster costruito utilizzando *docker-compose*, Docker assegna a livello applicativo una quantità fissa di memoria centrale che un processo deve occupare. In questo modo un processo vede tutta la RAM a disposizione della macchina ma non dispone di alcuna informazione sulla quantità utilizzabile di essa; tale informazione è conosciuta da Docker. Per questo motivo, il processo di avvio del modello ML crede di avere a disposizione una memoria sufficiente per l'esecuzione quando, in realtà, un limite software impedisce la sua esecuzione.

Per risolvere questa problematica è necessario agire su due fronti. Il primo è a livello hardware attraverso la modifica delle risorse della macchina GCP. Dal momento che viene utilizzata una macchina in cloud, l'aggiunta e la rimozione delle risorse risulta efficiente ed efficace.

La seconda azione correttiva viene svolta lato software tramite una riconfigurazione di Docker. L'obbiettivo di questa modifica è quello di aumentare il limite massimo di memoria utilizzabile dai container Docker contenenti nodi Elasticsearch di tipo Machine Learning. Una tra le diverse alternative implementabili agisce attraverso la modifica del valore di una variabile ambiente all'interno della macchina cloud, con conseguente riavvio del cluster.

Figura 4.5: Attivazione del modello *dslim/bert-base-NER* in Elasticsearch

Nella Figura 4.6 viene riportato lo stato della macchina GCP con il modello avviato, in termini di risorse utilizzate.

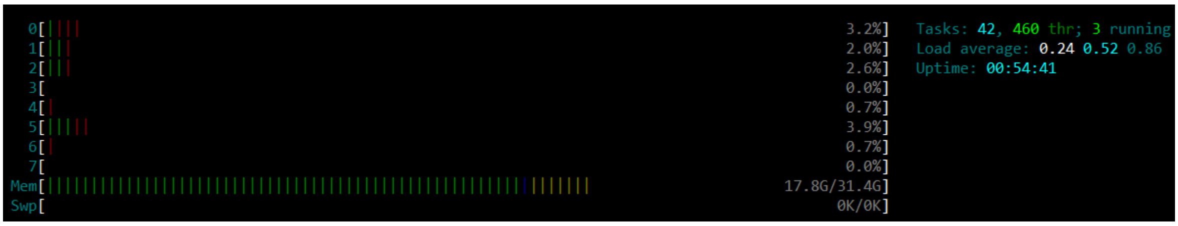

Figura 4.6: Risorse della macchina su GCP con modello avviato

4.4 Gestione delle licenze

Un dettaglio degno di nota presente insieme all'implementazione dei task di Machine Learning su Elasticsearch riguarda la licenza necessaria per l'abilitazione. Se l'esecuzione delle funzionalità base di Elasticsearch (operazioni CRUD sugli indici e ricerche full-text sui documenti associati) viene svolta tramite una licenza gratuita, le funzionalità di Machine Learning sono disponibili gratuitamente solo per un periodo di tempo limitato. Allo scadere dei trenta giorni di prova forniti, sarà necessario rinnovare la licenza dietro pagamento, nel caso si volesse utilizzare ulteriormente il cluster creato. L'alternativa a questa opzione consiste nell'eliminazione del cluster interno, con inclusi i volumi di salvataggio dei dati, e nel ricreare uno da zero, seguito da un reinserimento di tutti i modelli, indici, documenti, etc.

Il periodo di utilizzo del cluster per il caso in esame supera di gran lunga il periodo di prova imposto. Per questo motivo, verrà fatta una cancellazione periodica del cluster creato e ne verrà creato un altro. Al termine della creazione verranno rieseguiti tutti gli script precedentemente eseguiti, per rendere il cluster identico al suo predecessore all'istante prima della cancellazione. Questa soluzione è, quindi, adatta ad un ambiente di sviluppo preliminare, nel quale i dati immagazzinati sono di dimensioni contenute e ri-indicizzabili e, soprattutto, non ci sono richieste di continuità di servizio erogato.

CAPITOLO 5

Configurazione della pipeline di ingestion

L'abilitazione delle funzionalità di Machine Learning porta con sé un miglioramento rilevante delle funzionalità eseguibili da Elasticsearch. Per mezzo di esse, alcune ricerche specifiche, che verranno trattate in questo capitolo, vengono svolte con più efficienza ed efficacia, estendendo, di conseguenza, il campo di applicazione del motore di ricerca. In questa sezione verrà introdotto uno strumento, denominato Ingest Pipeline, che verrà utilizzato per inviare i documenti al modello di Machine Learning per la loro elaborazione. Verrà quindi definita una possibile Ingest Pipeline per svolgere il task di Named Entity Recognition sui documenti precedentemente estratti. Verranno, infine, descritti i diversi casi d'uso che impiegano l'utilizzo della Ingest Pipeline e dei documenti risultati da essa.

5.1 Descrizione della Ingest Pipeline

Allo stato attuale del cluster, ricavato sulla base della configurazione descritta nel capitolo precedente, è consentito utilizzare i modelli di Machine Learning (nel seguito, ML) solo per piccole sezioni di testo, come mostrato nella Figura 4.1. L'implementazione tipica, nonché quella effettuata in questo elaborato, consiste nell'applicare i modelli di ML ad una serie di documenti indicizzati su Elasticsearch o inviati tramite streaming di dati. Per realizzare tale funzione risulta necessario integrare il modello all'interno di uno strumento fornito da Elasticsearch che prende il nome di *Ingest Pipeline*.

Una pipeline di ingestion permette di eseguire trasformazioni sui documenti, come rimozione e aggiunta di campi, estrazioni di valori e arricchimento dei dati, prima che essi vengano indicizzati. Essa è formata da una serie di task configurabili, denominati *processor*, i quali vengono eseguiti secondo una procedura sequenziale, per apportare determinate modifiche nei documenti in input. Una volta che l'ultimo processor della pipeline termina l'esecuzione per un documento, quest'ultimo viene aggiunto in uno specifico indice, specificato nel processo in cui viene eseguita la pipeline. La fase di creazione di una pipeline può essere svolta attraverso l'interfaccia grafica fornita da Kibana, oppure tramite opportune API eseguibili su di esso o, infine, utilizzando apposite librerie. Per poter utilizzare una pipeline di ingestion è necessaria la presenza di almeno un nodo con ruolo *ingest*; nel caso in cui le feature di sicurezza di Elasticsearch sono abilitate, è richiesto, in aggiunta ai nodi, il possesso di tutti i privilegi opportuni [Elastic, 2022e]. Nella Figura 5.1 viene illustrata la struttura macroscopica di una Ingest Pipeline.

Una volta definita la Ingest Pipeline, e dopo aver verificato il suo corretto funzionamento, essa viene applicata all'interno di specifiche chiamate API, fornite da Elasticsearch, quali *bulk* o *reindex*, per consentire l'esecuzione dei processor su più documenti con un'unica azione.

Figura 5.1: Struttura generale di una Ingest Pipeline

Come già accennato, le azioni tipiche di una pipeline comprendono la modifica di documenti, attraverso l'aggiunta e/o la rimozione di informazioni da essi, tramite operazioni di semplice implementazione ed esecuzione. Il caso in esame comprende l'implementazione di azioni più complesse a livello implementativo e computazionale, in quanto integra un modello di Machine Learning, per elaborare molteplici righe di testo, su una mole rilevante di documenti. Il risultato di questa elaborazione permette di aggiungere informazione utile ai documenti per consentire l'esecuzione di ricerche maggiormente fedeli a ciò che l'utente desidera.

Nella prossima sezione vedremo, più nel dettaglio, qual è una possibile realizzazione di una pipeline di ingestion, in grado di utilizzare, al proprio interno, un modello di Machine Learning. Il modello utilizzato all'interno della Ingest Pipeline corrisponde al *dslim/bert-base-NER*, il quale è stato già importato nel cluster attraverso le operazioni svolte in precedenza.

5.2 Costruzione della pipeline

A partire dalle conoscenze introdotte nella sezione precedente, il passo successivo comprende la definizione di una Ingest Pipeline, in grado di utilizzare un modello di Machine Learning per elaborare i documenti caricati all'interno del cluster. Come è possibile notare nella Figura 5.2, la struttura della pipeline può essere divisa in tre parti principali, definite da tre attributi distinti. Tali attributi sono:

- *processors*;
- *inference*;
- *script*;
- *on_failure*.

L'attributo principale di qualunque processo di costruzione di una pipeline è, sicuramente, l'attributo *processors*, in quanto contiene, al suo interno, tutti i task che verranno eseguiti all'interno della pipeline stessa. Azioni tipiche, riportate all'interno di questo attributo,

```
mlpipe = IngestClient(es)
mlpipe.put_pipeline(id='ner-pipeline-bert-base', body={
"description": "NER pipeline",
  "processors": [
    {
      "inference": {
        "model_id": "dslim__bert-base-ner",
        "target_field": "ml.ner",
        "field_map": {
          "body":"text_field"
        }
      }
    },
    {
      "script": {
        "lang": "painless",
        "if": "return ctx['ml']['ner'].containsKey('entities')",
        "source": "Map tags = new HashMap(); for (item in ctx['ml']['ner']['entities'])
      }
    }
  ],
  "on_failure": [
    {
      "set": {
        "description": "Index document to 'failed-<index>'",
        "field": "_index",
        "value": "failed-{{{ _index }}}"
      }
    },
    {
      "set": {
        "description": "Set error message",
        "field": "ingest.failure",
        "value": "{{_ingest.on_failure_message}}"
      }
    }
```

Figura 5.2: Struttura della Ingest Pipeline per NER

riguardano l'aggiunta e la rimozione di attributi, la modifica del testo per impostare le lettere in maiuscolo o in minuscolo, etc. Tra i *processor* definibili è presente *inference*, il quale permette di svolgere processi di inferenza su una serie documenti, specificando il modello di Machine Learning da utilizzare. Nel caso in esame, la pipeline sarà formata da un processor di tipo *inference*, nel quale viene specificato, nel campo *model_id*, il nome del modello per NER da applicare ai documenti in ingresso. Nella sua definizione, il processor contiene, oltre al modello ML utilizzato, anche le informazioni sull'attributo contenente il testo da elaborare, definito dentro *field_map*, e l'attributo in cui viene salvato il risultato, ottenuto dall'esecuzione, nel campo *target_field*. Per quanto riguarda il caso in esame, è stato utilizzato il campo *body* di ciascun documento, come sorgente di testo per il modello, dal momento che esso possiede un contenuto informativo maggiore rispetto al campo *headline*.

Esiste, inoltre, un'altra tipologia di processor che consente di eseguire script in linguaggio JAVA per la realizzazione di espressioni ad-hoc, da utilizzare all'interno della pipeline. Nel caso in esame, per ciascun documento, il processor *script* viene eseguito successivamente al processo *inference*, una volta che quest'ultimo ha terminato l'estrazione delle Named Entity. Lo script definito si pone l'obbiettivo di estrarre tutte le entità nominate dall'attributo generato dal processor precedente, insieme alla rispettiva etichetta, per poi salvarle dentro una struttura dati di tipo *HashMap*. In questa struttura dati, sono presenti oggetti di tipo *HashSet*, assegnati a diversi indirizzi dell'HashMap, i quali vengono definiti in base al valore dell'etichetta delle entità. Sulla base della struttura definita, ciascuna Named Entity rilevata verrà inserita all'interno dell'HashSet, in corrispondenza al valore della label ad essa associata. Una volta elaborate tutte le entità nominate, l'HashMap viene utilizzata per aggiungere un nuovo attributo all'interno del documento analizzato. La struttura dell'attributo creato è

identica a quella della struttura dati utilizzata, dove le entità vengono raggruppate in base alla rispettiva classe di appartenenza. Nella Figura 5.3 viene raffigurato lo script definito poc'anzi, nella sua interezza.

```
{
  "script": {
    "lang": "painless",
    "if": "return ctx['ml']['ner'].containsKey('entities')",
    "source": "Map tags = new HashMap(); "
              "for (item in ctx['ml']['ner']['entities']) {
                " if (!tags.containsKey(item.class_name)) tags[item.class_name] = new HashSet();
                "   tags[item.class_name].add(item.entity);
              "}"
              "ctx['tags'] = tags;"
  }
}
```

Figura 5.3: Definizione dello script all'interno della Ingest Pipeline

Durante l'esecuzione della pipeline di ingestion, potrebbe capitare che, in alcuni documenti, il modello non è in grado di identificare o definire con certezza una o più entità nominate. Secondo l'impostazione di default, l'esecuzione di una Ingest Pipeline si arresta quando un processor si imbatte in un errore o fallisce la sua esecuzione. La soluzione a questa problematica comprende la definizione di un attributo *on_failure* all'interno della pipeline; attraverso questo campo, l'interruzione dei processor viene gestita con un'azione correttiva e viene passata l'esecuzione ai processor successivi.

All'interno della pipeline definita per il caso in esame, l'azione correttiva viene implementata per separare i documenti processati correttamente da quelli non elaborabili, a causa di un errore presente al loro interno. Questi ultimi vengono salvati all'interno di un indice e verrà aggiunto ad essi un attributo contenente una descrizione dell'errore incontrato. La struttura dell'azione correttiva appena definita viene rappresentata nella Figura 5.4. Scendendo più nel dettaglio, la prima azione definita si occupa di indicizzare il documento verso un indice diverso da quello di destinazione, il quale avrà lo stesso nome di quest'ultimo ma sarà preceduto dalla stringa *failed-*. La seconda azione, invece, si occupa di aggiungere un attributo al documento; in questo attributo verrà inserito il messaggio generato durante l'elaborazione del documento.

Una volta definita per intero la struttura della pipeline di ingestion si passa alla sua esecuzione, non prima di aver salvato la pipeline all'interno del cluster Elasticsearch. Risulta, inoltre, necessario controllare che sia presente almeno un nodo in grado di eseguire le pipeline di ingestion. Allo stato attuale del cluster, tale requisito viene soddisfatto fin da subito, dal momento che, a tutti i nodi è assegnato il ruolo *ingest*. L'esecuzione di una Ingest Pipeline avviene all'interno di tutte le operazioni in grado di processare una serie di documenti; in tali operazioni viene dichiarato il nome della pipeline da utilizzare nell'esecuzione del processo. Le funzioni a cui si fa riferimento includono, come vedremo per il caso in esame, le operazioni di *bulk* e quelle di *reindex*.

Nella Figura 5.5 viene riportata la definizione della chiamata API per il processo di reindex che utilizza la pipeline precedentemente definita. Quest'ultima viene inserita all'interno dell'attributo *dest* dove è presente, oltre al nome dell'indice su cui salvare i documenti, il nome della pipeline da utilizzare. Dal momento che un processo di reindexing contenente un'azione di inferenza richiede una grande quantità di tempo per l'esecuzione, il processo viene eseguito tramite chiamata asincrona, evitando l'arresto dell'esecuzione allo scadere del timeout. In aggiunta alla chiamata asincrona, viene aumentato anche il timeout di *scroll*;

```
"on_failure": [
  {
    "set": {
      "description": "Index document to 'failed-<index>'",
      "field": "_index",
      "value": "failed-{{{ _index }}}"
    }
  },
  {
    "set": {
      "description": "Set error message",
      "field": "ingest.failure",
      "value": "{{_ingest.on_failure_message}}"
    }
  }
]
```

Figura 5.4: Definizione dell'azione correttiva ad un errore, all'interno della Ingest Pipeline

questo valore sta ad indicare per quanto tempo una vista dell'indice deve essere mantenuta per la ricerca basata su scroll.

```
# Ingestion dei documenti "LaRepubblica" nella pipeline
# Durata 10 ore; 7 documenti non processati
es.reindex(body={
    "conflicts": "proceed",
    "source": {
        "index": "repubblica-train"
    },
    "dest":{
        "index":"repubblica-train-ner",
        "pipeline":"ner-pipeline-bert-base"
    }
}, wait_for_completion=False, scroll="60m")
```

Figura 5.5: Definizione dell'API di reindex asincrona

Una volta eseguita la chiamata API di reindex precedentemente definita, è possibile controllare lo stato di avanzamento del task attraverso una chiamata API, inserendo l'identificatore ricevuto come risposta dalla chiamata precedente. La struttura della chiamata API e la relativa risposta ricevuta vengono riportate nella Figura 5.6, la quale mostra, oltre allo stato del task, anche le informazioni in merito al numero di documenti elaborati e al tempo impiegato per l'esecuzione.

Al termine dell'esecuzione del task saranno presenti due nuovi indici, contenenti gli stessi documenti presenti negli indici di partenza, suddivisi per testata giornalistica. La differenza tra le due tipologie di documento risiede nella presenza di tutte le informazioni contestuali al documento, le quali risultavano nascoste nei documenti degli indici di partenza. Nel caso siano presenti documenti non correttamente elaborati, verranno definiti due ulteriori indici, nei quali verranno inseriti tutti questi documenti insieme al messaggio di errore generato.

```
    # Impostare come 'task_id' il codice del task stampato dal reindex per "IlGIornale"
    es_tasks = TasksClient(es)
    print(json.dumps(dict(es_tasks.get(task_id="EOk-x9XWSHCoCZCpt5-PIw:21229")),indent=1, ensure_ascii=False))
```

```
Output exceeds the size limit. Open the full output data in a text editor
{
 "completed": true,
 "task": {
  "node": "EOk-x9XWSHCoCZCpt5-PIw",
  "id": 21229,
  "type": "transport",
  "action": "indices:data/write/reindex",
  "status": {
   "total": 63701,
   "updated": 0,
   "created": 63701,
   "deleted": 0,
   "batches": 64,
   "version_conflicts": 0,
   "noops": 0,
   "retries": {
    "bulk": 0,
    "search": 0
   },
   "throttled_millis": 0,
   "requests_per_second": -1.0,
   "throttled_until_millis": 0
```

Figura 5.6: Controllo dello stato di esecuzione del task tramite chiamata API

5.3 Definizione dei casi d'uso

Sulla base della pipeline costruita e del risultato ottenuto dalla sua esecuzione nella fase precedente, si procede, ora, con la definizione dei possibili casi d'uso, e quindi, dei contesti principali in cui è possibile utilizzare tale pipeline. Per il caso in esame sono state ipotizzate quattro possibili applicazioni; alcune di esse vengono realizzate soltanto attraverso un processo di inferenza, mentre altre possono essere realizzate anche con una ricerca full-text standard. In merito a queste ultime applicazioni, verrà svolto un confronto tra la soluzione che coinvolge l'utilizzo del Named Entity Recognition e la controparte corrispondente, senza alcuna componente di Machine Learning. Le due alternative verranno confrontate utilizzando come metriche di confronto, oltre all'efficacia, l'efficienza e la complessità di definizione.

I casi d'uso citati verranno descritti più in dettaglio nelle prossime sottosezioni.

5.3.1 Ricerca di un'entità

La prima applicazione realizzabile con gli strumenti a disposizione riguarda lo sviluppo di una ricerca full-text, incentrata su una specifica entità (persona, luogo, organizzazione, etc.). Questo tipo di applicazione può essere realizzata con l'ausilio di tre tipologie distinte di ricerca. La prima utilizza query basate su *multi_match*, le quali vengono già implementate da Elasticsearch senza ricorrere all'utilizzo di modelli di Machine Learning. La seconda è basata su *simple_query_string*, e anch'essa viene messa a disposizione, fin da subito, da Elasticsearch. La terza tipologia richiede, invece, l'integrazione delle funzionalità di Machine Learning, in quanto sfrutta i corrispettivi campi, nella ricerca dei documenti.

Partendo dal primo tipo di ricerca definito, le query che sfruttano il *multi_match* presentano molta ambiguità nei risultati ottenuti dalla loro esecuzione. Difatti, insieme ai documenti

rilevanti al contesto delle parole da ricercare, la ricerca riporta anche i documenti che presentano le parole utilizzate all'interno della query, fuori dal contesto in esame. Ad esempio, se un utente intendesse ricercare tutti i documenti che hanno, come soggetto, una persona denominata "Andrea Pazienza", una ricerca tramite query *multi_match* restituisce, insieme ai documenti rilevanti alla ricerca, tutti quelli che contengono le sole parole "Andrea" e/o "Pazienza". Questi documenti presentano, al proprio interno, le parole inserite nella query, ma fuori dal proprio contesto. Sempre in riferimento a questo esempio, un documento errato potrebbe riguardare una persona di nome "Andrea", il quale possiede molta "pazienza".

Nella Figura 5.7 viene riportata la definizione della query di ricerca che restituisce, sulla base dell'esempio appena definito, i documenti errati, insieme a quelli corretti.

```
larepubblica_search_ft=es.search(index="repubblica-train", query = {
    'multi_match':{
        'query': 'Andrea Pazienza',
        'fields':['headline','body']
    }}, size=1000, source="headline")
print(json.dumps(dict(larepubblica_search_ft),indent=1, ensure_ascii=False))
```

Figura 5.7: Definizione della query di ricerca non ottimizzata

Insieme ad essa viene, inoltre, mostrata nella Figura 5.8 una sezione dei documenti restituiti dalla query ricerca, nella quale è possibile notare documenti non conformi al tipo di richiesta definita.


```
{
 "_index": "repubblica-train",
 "_id": "45089",
 "_score": 14.022264,
 "_source": {
  "headline": "\"Vi racconto mio fratello Andrea Pazienza. Rimpianti? La notte in cui è morto non sono andata a un concerto con lui\""
 }
},
{
 "_index": "repubblica-train",
 "_id": "605",
 "_score": 13.8196945,
 "_source": {
  "headline": "'Trent'anni senza', la mostra che apre il festival Arf! è dedicata al genio di Andrea Pazienza"
 }
},
{
 "_index": "repubblica-train",
 "_id": "14614",
 "_score": 12.393711,
 "_source": {
  "headline": "Pazienza, Tamburini e i formidabili anni di Frigidaire"
 }
},
{
 "_index": "repubblica-train",
 "_id": "2873",
 "_score": 11.833321,
 "_source": {
  "headline": "\"Io, velista in carrozzina: così realizzo il sogno di vivere senza barriere\""
 }
},
```


Figura 5.8: Documenti restituiti dalla query di ricerca non ottimizzata

Le soluzioni alla problematica dell'ambiguità definita poc'anzi possono essere implementate in due modi differenti. Il primo consiste nell'utilizzare query di tipo *simple_query_string*, per rendere la ricerca maggiormente mirata al contesto delle parole. La seconda metodologia, invece, realizza una ricerca basata sull'elaborazione dei documenti da un modello di Named Entity Recognition, con la possibilità di sfruttare le entità estratte per ricercare i soli docu-

menti contenenti la sequenza di parole esatte inserite nella query di ricerca. Una possibile definizione di una query di tipo *simple_query_string* viene riportata nella Figura 5.9, insieme ai documenti estratti dalla chiamata API utilizzata per avviare la ricerca, illustrati nella Figura 5.10.

```
larepubblica_search_ft=es.search(index="repubblica-train", query = {
    'simple_query_string':{
        'query': '\'Andrea Pazienza\'',
        'fields':['headline','body']
    }}, size=1000, source="headline")
print(json.dumps(dict(larepubblica_search_ft),indent=1, ensure_ascii=False))
```

Figura 5.9: Definizione della query ottimizzata senza integrare il Machine Learning


```
{
 "_index": "repubblica-train",
 "_id": "21131",
 "_score": 32.806423,
 "_source": {
  "headline": "Se Andrea Pazienza avesse 60 anni"
 }
},
{
 "_index": "repubblica-train",
 "_id": "2325",
 "_score": 29.458796,
 "_source": {
  "headline": "Andrea Pazienza, con Repubblica i disegni mai visti"
 }
},
{
 "_index": "repubblica-train",
 "_id": "55736",
 "_score": 28.055248,
 "_source": {
  "headline": "Andrea Pazienza, una mostra per ricordarlo a trent'anni dalla morte. E spuntano due inediti"
 }
},
{
 "_index": "repubblica-train",
 "_id": "45089",
 "_score": 27.890265,
 "_source": {
  "headline": "\"Vi racconto mio fratello Andrea Pazienza. Rimpianti? La notte in cui è morto non sono andata a un concerto con lui\""
 }
```


Figura 5.10: Documenti restituiti dalla query ottimizzata senza integrare il Machine Learning

Oltre a definire un metodo risolutivo, attraverso strumenti standard forniti da Elasticsearch, è stata costruita una soluzione basata sull'informazione ottenuta dall'esecuzione della Ingest Pipeline. Difatti, le entità nominate, presenti nei documenti elaborati, possono essere impiegate per ottimizzare la ricerca del caso in esame. Un esempio di query di ricerca, basata sull'esempio precedentemente definito, viene riportata nella Figura 5.11, mentre i documenti estratti vengono riportati nella Figura 5.12.

```
larepubblica_search_ner=es.search(index="repubblica-train-ner", query = {
    'term':{
        'tags.PER.keyword': "Andrea Pazienza"
    }}, size=1000, source="headline")
print(json.dumps(dict(larepubblica_search_ner),indent=1, ensure_ascii=False))
```

Figura 5.11: Definizione della query ottimizzata tramite Machine Learning

```
{
 "_index": "repubblica-train-ner",
 "_id": "605",
 "_score": 12.424208,
 "_ignored": [
  "body.keyword",
  "ml.ner.predicted_value.keyword"
 ],
 "_source": {
  "headline": "'Trent'anni senza', la mostra che apre il festival Arf! è dedicata al genio di Andrea Pazienza"
 }
},
{
 "_index": "repubblica-train-ner",
 "_id": "4834",
 "_score": 12.424208,
 "_ignored": [
  "body.keyword",
  "ml.ner.predicted_value.keyword"
 ],
 "_source": {
  "headline": "Giornalismo, narrativa e fumetto, in libreria “Il Pollo”"
 }
},
{
 "_index": "repubblica-train-ner",
 "_id": "21131",
 "_score": 12.424208,
 "_ignored": [
  "body.keyword",
  "ml.ner.predicted_value.keyword"
 ],
 "_source": {
  "headline": "Se Andrea Pazienza avesse 60 anni"
 }
},
```


Figura 5.12: Documenti restituiti dalla query ottimizzata tramite Machine Learning

Sulla base dei risultati ottenuti dall'esecuzione delle tre tipologie di soluzione si può concludere che la seconda e la terza sono in grado di restituire tutti i documenti coerenti con la richiesta inviata, escludendo, quindi, tutti quelli non interessanti. Tuttavia, è doveroso chiedersi quale delle due soluzioni sia la più complessa nella sua definizione, tenendo in considerazione anche il tempo impiegato per ricercare e restituire i documenti. In termini di complessità di scrittura, la seconda soluzione, basata sulla ricerca di tipo *simple_query_string*, comporta la scrittura di una query più complessa rispetto alla sua controparte che utilizza il modello ML. Per quanto riguarda il tempo di risposta, come è possibile notare nella Figura 5.13, la soluzione che integra la componente di ML richiede meno tempo, rispetto alla seconda soluzione definita.

Per tali motivazioni, impiegare una query che sfrutta le informazioni ricavate dall'esecuzione di un modello di Machine Learning non solo presenta una maggiore semplicità di implementazione, ma richiede, altresì, un minor tempo di risposta per la ricerca. Questo perché, di fatto, la query è basata su un'analisi che viene svolta non all'atto della sua esecuzione, bensì dalla Ingest Pipeline, nel tempo in cui avviene l'ingestion dei documenti iniziali.

5.3.2 Ricerca delle tendenze

Se il caso d'uso appena analizzato permette un'implementazione senza utilizzare alcuna componente di Machine Learning, la ricerca delle tendenze richiede, necessariamente, l'utilizzo di un modello di ML in grado di estrarre informazioni nascoste nel testo. Questo caso

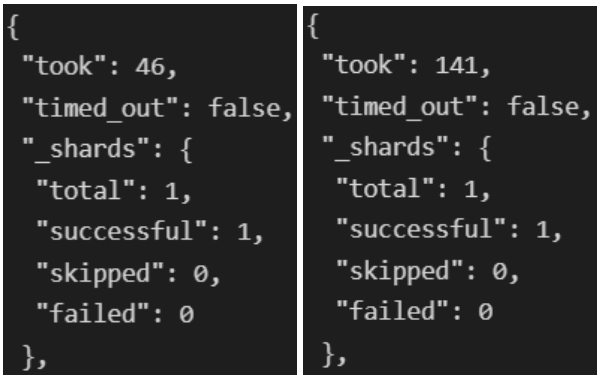

Figura 5.13: A sinistra, tempo di esecuzione della query senza Machine Learning. A destra, tempo di esecuzione della query tramite Machine Learning

applicativo è caratterizzato da uno streaming di dati, i quali vengono direzionati verso la macchina in cui è in esecuzione il cluster Elasticsearch. I dati inviati vengono, poi, raggruppati in un insieme di documenti dove, insieme al testo e al titolo degli articoli di giornale, viene riportata anche la data di pubblicazione. Questi campi vengono estratti e utilizzati per svolgere un'analisi dei trend, premettendo, inoltre, un filtraggio basato sulla data di pubblicazione.

Per quanto riguarda il caso in esame, l'invio dei dati in streaming viene simulato attraverso una variante della funzione di *bulk*, dove verrà integrata, al suo interno, la pipeline definita nel capitolo precedente. I documenti scelti da inviare via streaming contengono i dati degli articoli provenienti dallo stesso dataset di partenza, ai quali viene aggiunto un campo ulteriore relativo alla data di pubblicazione. Come conseguenza della modifica della struttura dei documenti, sono stati definiti due ulteriori indici Elasticsearch, con struttura identica a quelli utilizzati precedentemente, ma con l'aggiunta di un attributo di tipo data. A questo punto, viene applicata la stessa procedura di trasformazione dei dati, provenienti dal dataset, descritta nel Capitolo 3. Una volta ottenuti i nuovi documenti in formato JSON, questi vengono inseriti all'interno della chiamata API per l'esecuzione della funzione *streaming_bulk*. Nella sua definizione è possibile indicare il nome della pipeline di ingestion da utilizzare per elaborare i documenti, prima che essi vengano salvati nell'indice di destinazione.

Nella Figura 5.14 viene riportata una possibile definizione della funzione per l'invio di dati in modalità streaming, tramite la chiamata API precedentemente descritta.

```
errors = []
success, failed = 0, 0
for success, response in helpers.streaming_bulk(client=es, actions=repubblica, pipeline="ner-pipeline-bert-base", raise_on_error=False, request_timeo
    if not success:
        errors.append(response)
        failed += 1
    else:
        success += 1
if errors:
    print(errors)
success, failed
```

Figura 5.14: Definizione dell'algoritmo di simulazione di invio dei dati in streaming

Terminata l'elaborazione dei documenti, questi ultimi possono essere impiegati per svolgere il compito principale del caso d'uso definito, ovvero utilizzare le entità identificate per svolgere un'analisi dei trend. La sua realizzazione viene svolta, lato applicativo, attraverso una API di ricerca. La richiesta viene costruita in maniera tale che svolga un'azione di aggregazione sulle Named Entity, riportando, per prime, quelle che compaiono con maggiore frequenza nei documenti.

Una possibile analisi eseguibile con i dati a disposizione viene riportata nella Figura 5.15, mentre nella Figura 5.16 viene mostrato il risultato ottenuto. Nell'esempio illustrato, viene svolta un'aggregazione di tutte le entità appartenenti ad una specifica categoria (Luogo), all'interno di tutti gli articoli pubblicati in un determinato intervallo di tempo.

```
larepubblica_trend=es.search(index="repubblica-train-stream", size=0, source="headline", aggs={
    'word-trend': {
        'terms':{
            'field':'tags.LOC.keyword',
            'size':10000
        }
    }},
    query={
        "bool": {
            "must": [
                {
                "match_all": {}
                }
            ],
            'filter':[{
                'range': {
                    'date': {
                        'gte': '2002-01-01',
                        'lte': '2012-12-31'
                    }
                }
            }]
        }
    })
print(json.dumps(dict(larepubblica_trend),indent=1, ensure_ascii=False))
```

Figura 5.15: Definizione della query di aggregazione

5.3.3 Ricerca filtrata sulla tipologia classe

Un'altra possibile applicazione che coinvolge, esclusivamente, l'integrazione del Machine Learning nelle query di ricerca, riguarda l'utilizzo del tipo di classe delle entità come strumento di filtraggio. Data una query di ricerca, il campo filtro funziona esattamente come una normale query; esso permette, quindi, di ricercare documenti che sono rilevanti per il valore inserito, con l'unica differenza che non contribuisce al calcolo dello score di rilevanza del documento. Ad esempio, se volessimo cercare tutti i documenti che riguardano le mostre d'arte e restituire soltanto quelle che coinvolgono le organizzazioni, è necessario definire un campo *filter*, oltre ai campi tipici di una query. In questo modo, lo score dei documenti verrà calcolato senza tenere in considerazione quante volte una o più organizzazioni sono riferite nel testo, evitando, quindi, la comparsa di documenti non rilevanti per la specifica ricerca.

Nella Figura 5.17 viene riportata una possibile definizione di un'interrogazione filtrata, riferita all'esempio appena descritto; la risposta ottenuta dalla sua esecuzione viene, invece, riportata nella Figura 5.18 .

5.3.4 Visualizzazione decorata del testo

L'ultimo caso d'uso definito riguarda la visualizzazione decorata di testi, sulla base di determinati criteri scelti dall'utente stesso. Andando più nel dettaglio, viene definita un'applicazione in grado di sottolineare e/o colorare una specifica entità nominata, in base ai parametri restituiti dal modello di NER. Tra i suddetti parametri messi a disposizione,

```
"aggregations": {
 "word-trend": {
  "doc_count_error_upper_bound": 0,
  "sum_other_doc_count": 0,
  "buckets": [
   {
    "key": "Italia",
    "doc_count": 744
   },
   {
    "key": "Roma",
    "doc_count": 249
   },
   {
    "key": "Europa",
    "doc_count": 240
   },
   {
    "key": "Russia",
    "doc_count": 156
   },
   {
    "key": "Milano",
    "doc_count": 151
   },
```


Figura 5.16: Elenco delle entità più rilevanti estratte dalla query di aggregazione

è possibile selezionare la classe di appartenenza dell'entità, attraverso strumenti grafici di selezione, come, ad esempio, un *radio button*. Allo stesso modo, è possibile estrarre il valore di classificazione delle entità per rendere la decorazione più selettiva. In questo modo verranno nascoste tutte le entità che presentano una misura di classificazione inferiore ad un valore di soglia, impostato dall'utente con un apposito slider. Per la realizzazione di tale richiesta è necessario restituire, oltre al testo contenente le entità nominate in rilievo, anche i parametri di classificazione delle entità, presenti all'interno dei documenti.

Nel caso in esame, la decorazione del testo viene svolta utilizzando le sole informazioni, relative al valore di classificazione e alla classe di appartenenza delle entità. Oltre a definire la chiamata API per la query di ricerca, vengono definite due variabili, contenenti, rispettivamente, la classe da selezionare e il valore di classificazione minimo richiesto per la selezione. Una possibile definizione di tali elementi viene illustrata nella Figura 5.19. Un dettaglio degno di nota in quest'ultima riguarda il campo *source* della query; se nei casi precedenti veniva ogni volta estratto il titolo del documento, in questo caso vengono estratti gli attributi *predicted_value* e *entities*, creati tramite la Ingestion Pipeline di riferimento.

L'attributo *ml.ner.predicted_value* contiene il testo prodotto dal modello di Machine Learning, all'interno del quale sono racchiuse le entità e le rispettive classi di appartenenza. Il secondo attributo, invece, contiene tutte le informazioni riferite alle Named Entity, prodotte dal modello di Named Entity Recognition e suddivise per etichetta di classe.

A partire dalla risposta, ottenuta dalla chiamata API, il prossimo passo riguarda l'estrazione delle informazioni necessarie a partire dalla risposta appena acquisita. La classe e il valore di classificazione, relative a ciascuna entità, verranno confrontati con i valori inseriti dall'utente, attraverso l'interfaccia grafica realizzata. Per ogni entità che soddisfa le condizioni inserite, il testo viene modificato, applicando una decorazione in tutti i punti in cui è presente l'entità e la classe corrispondente.

```
larepubblica_search_filter=es.search(index="repubblica-train-ner", source="headline", query={
  'bool': {
    'must': [
      {
        'match': {
          'body': 'mostra'
        }
      }
    ],
    'filter': [
      {
        'exists': {
          'field': 'tags.ORG'
        }
      }
    ]
  }})
print(json.dumps(dict(larepubblica_search_filter),indent=1, ensure_ascii=False))
```

Figura 5.17: Definizione della query di ricerca filtrata


```
{
 "_index": "repubblica-train-ner",
 "_id": "25395",
 "_score": 6.1929817,
 "_ignored": [
  "body.keyword",
  "ml.ner.predicted_value.keyword"
 ],
 "_source": {
  "headline": "Indiana e gli altri. A Roma si rappresenta l'amore"
 }
},
{
 "_index": "repubblica-train-ner",
 "_id": "31228",
 "_score": 6.1929817,
 "_ignored": [
  "body.keyword",
  "ml.ner.predicted_value.keyword"
 ],
 "_source": {
  "headline": "I quadri nascosti tornano alla luce. Due mostre a Berna e Bonn scoprono le opere di Cornelius Gurlitt"
 }
},
{
 "_index": "repubblica-train-ner",
 "_id": "57927",
 "_score": 6.187843,
 "_ignored": [
  "body.keyword",
  "ml.ner.predicted_value.keyword"
 ],
 "_source": {
  "headline": "Emergency, “Dire, fare, baciare”: le vignette di Fabio Magnasciutti in mostra all'infopoint di Roma"
 }
```


Figura 5.18: Documenti restituiti dalla query di ricerca filtrata

```
# Definizione vincoli di evidenziatura testo
label_ilgiornale="PER"
class_probability_ilgiornale=0.8

# Ricerca dei documenti
ilgiornale_pretty_view=dict(es.search(index="ilgiornale-train-ner", size=1000, source=['ml.ner.predicted_value','ml.ner.entities'], query={
  "term": {
      "tags.PER.keyword": "Piero Angela"
    }}))
```

Figura 5.19: Definizione della query di ricerca e dei vincoli per la visualizzazione del testo

Al termine di questa operazione, viene restituito sullo schermo il corpo del documento decorato; le suddette operazioni vengono eseguite per tutti i documenti che sono stati selezionati dalla query precedente. Una rappresentazione dell'algoritmo appena descritto viene riportata nella Figura 5.20, mentre il risultato finale ottenuto dalla sua esecuzione, viene mostrato nella Figura 5.21.

```
# Per ciascun documento vengono confrontate le etichette e la probabilità di classificazione di ogni entità nominata
# Se superano una certa soglia viene cambiato il font dell'entità nominata
for document in documents:
    body = document.get("_source").get("ml").get("ner").get("predicted_value")
    entities = dict(document).get("_source").get("ml").get("ner").get("entities")
    doc=list()
    for entity in entities:
      if(entity.get("class_name")==label_ilgiornale and float(entity.get("class_probability"))>=class_probability_ilgiornale):
        body = body.replace('['+entity.get("entity")+']', termcolor.colored('['+entity.get("entity")+']', 'blue'))
        named_entity=entity.get("entity").split(' ')
        if(len(named_entity)>1):
          ne=''
          for part in named_entity:
            if(ne ==''):
              ne=part
            else:
              ne=ne+"+"+part
          body = body.replace('('+entity.get("class_name")+'&'+ne+')', termcolor.colored('('+entity.get("class_name")+'&'+ne+')', 'blue'))
        else:
          body = body.replace('('+entity.get("class_name")+'&'+entity.get("entity")+')', termcolor.colored('('+entity.get("class_name")+'&'+
          entity.get("entity")+')', 'blue'))
    print(body+"\n \n \n")
```

Figura 5.20: Estrazione delle metriche di classificazione delle entità e stampa del testo decorato

```
Qualche momento di imbarazzo a Domenica In . [Piero Angela](PER&Piero+Angela) ospite nel salotto domenicale di Rai Uno si è lasciato andare ad una
lunga intervista con [Cristina Paordi](PER&Cristina+Paordi) in cui ha raccontato la sua carriera e anche qualche retroscena della sua vita privata. Lo
storico conduttore di SuperQuark di fatto ha anche parlato della sua passione per la musica e così la Parodi gli ha chiesto qualche informazione sul
Festival [di Sanremo](MISC&di+Sanremo). A Domenica In il tema del Festival è il più discusso perché tra poche settimane prenderà il via l'edizione 2018
guidata da [Claudio Baglioni](PER&Claudio+Baglioni), [Michelle Hunziker](PER&Michelle+Hunziker) e [Pierfrancesco](PER&Pierfrancesco) Favino. La
[Parodi](ORG&Parodi) ha chiesto un pronostico ad [Angela](PER&Angela) con la classica domanda: "Secondo lei, chi vincerà Sanremo?". [Angela]
(PER&Angela) è rimasto spiazzato e di fatto ha replicato in modo chiaro, gelando così la conduttrice: "Non ho mai visto Sanremo, da pianista e jazzista
non vedo di buon occhio il mondo della canzone". E poi: "Sono andato a [Sanremo](LOC&Sanremo) una dola volta con [Rita Levi Montalcini]
(PER&Rita+Levi+Montalcini) per una iniziativa per raccogliere fondi contro la sla". Alla Parodi non è rimasta che la scelta di cambiare argomento e
andare avanti con l'intervista.



Un lieve malore ha colpito ieri il giornalista [Piero Angela](PER&Piero+Angela) . Il popolare conduttore, 86 anni, è stato colto da malore mentre si
trovava in una delle zone nei pressi di Redipuglia, teatro di battaglie della Prima [guerra](ORG&guerra) mondiale e dove aveva preso parte
all'inaugurazione di un museo multimediale dedicato appunto alla [Grande](LOC&Grande) guerra. Si è trattato di una semplice insolazione, come
riferiscono fonti di viale Mazzini, e [Angela](PER&Angela), dopo gli accertamenti, è stato dimesso dall'ospedale ed è tornato in aereo a [Roma]
(LOC&Roma).
```

Figura 5.21: Estrazione delle metriche di classificazione delle entità e stampa del testo decorato

CAPITOLO 6

Validazione e risultati ottenuti

Il capitolo appena concluso ha esposto la parte centrale di questo elaborato di tesi; in esso è stato trattato il processo di realizzazione di una possibile pipeline di ingestion, in grado di utilizzare le funzionalità di Machine Learning (ML), messe a disposizione da Elasticsearch. A partire dalla Ingest Pipeline realizzata, sono state illustrate le possibili applicazioni che coinvolgono il suo diretto utilizzo, insieme ai risultati ottenuti dalla sua esecuzione. In questo capitolo verranno illustrate le metriche utilizzate per la valutazione dello strumento, a partire dai casi d'uso descritti precedentemente e dai rispettivi risultati ottenuti. Sulla base di esse verranno, quindi, presentati gli aspetti positivi e negativi dei diversi tipi di soluzioni impiegate, in particolare sulla Ingest Pipeline costruita ed utilizzata per l'elaborazione dei documenti.

6.1 Definizione delle metriche di valutazione

Per svolgere una valutazione in grado di definire i punti di forza e di debolezza degli strumenti impiegati per raggiungere specifici obbiettivi risulta necessario definire misure appropriate, capaci di descrivere completamente le soluzioni impiegate. In questo modo, sarà possibile fornire una descrizione esaustiva, partendo dalla costruzione degli strumenti fino ad arrivare alla misurazione delle performance in fase di esecuzione, passando per le risorse impiegate. Con questa premessa, sono state definite tre misure principali; queste sono:

- *Source Lines of Code (SLOC)*: consiste in una metrica software nella quale viene utilizzato il numero di linee di codice sorgente per dare una misura della complessità di un software.
- *Occupazione di memoria secondaria*: riferita, principalmente, alla dimensione degli indici all'interno del cluster, in termini di occupazione di memoria secondaria.
- *Tempo di elaborazione e di ricerca dei documenti*: questa metrica riguarda il tempo richiesto dalle azioni di *reindex* e *bulk* che implementano la Ingest Pipeline per elaborare e indicizzare i documenti all'interno di Elasticsearch e tramite lo streaming di dati. Essa coinvolge, inoltre, il tempo richiesto dalle query di ricerca per elaborare e fornire i risultati.

6.2 Validazione della Ingest Pipeline

Le metriche appena definite verranno adottate per validare la pipeline di ingestion, utilizzando l'indicatore SLOC per misurare la sua complessità di definizione. Le altre due

misure verranno impiegate, principalmente, in tutte le applicazioni che implementano la pipeline, utilizzando, come riferimento, anche i risultati ottenuti dalla sua esecuzione.

6.2.1 Complessità di definizione

Partendo dalla sua definizione, la pipeline di ingestion utilizzata presenta una struttura standard non troppo complessa da definire. Difatti, tralasciando la complessità intrinseca del linguaggio JSON, la parte che richiede più attenzione, nella struttura della Ingest Pipeline, risiede nella scrittura dello script per l'elaborazione dei documenti. In merito alla sua definizione, tale script svolge una funzione di manipolazione del documento, composta da azioni non troppo difficili da realizzare. L'unica problematica rilevante riguarda l'impossibilità di scrivere codice indentato su diverse righe; occorre, infatti, definire il corpo dello script su un'unica riga di testo, rendendo più difficile l'identificazione degli errori presenti al suo interno.

La stessa conclusione viene, inoltre, applicata in merito alla definizione di tutte le chiamate API per la ricerca full-text all'interno dei documenti. L'unica differenza riguarda la possibilità di scrivere il corpo della query utilizzando l'indentazione su più righe, rendendo la definizione di quest'ultima ancora più chiara e semplice nella sua struttura.

In base alle considerazioni definite per questa metrica, si può concludere che la definizione della struttura della pipeline e delle query di ricerca risulta di semplice realizzazione.

6.2.2 Tempo di indicizzazione e interrogazione

Se la definizione della pipeline corrisponde ad un aspetto positivo, per via della sua bassa complessità, non si può dire lo stesso in merito al tempo richiesto per l'elaborazione dei documenti all'interno della pipeline. Infatti, l'esecuzione di un processo di reindex per tutti i documenti appartenenti ad una singola testata giornalistica richiede un tempo di esecuzione pari a dieci ore. Di conseguenza, vengono impiegate un totale di venti ore per svolgere un processo di inferenza su tutti i documenti, all'interno dei rispettivi indici, con inclusa l'azione di salvataggio nei nuovi indici definiti. Per quanto riguarda, invece, l'azione di bulk per lo streaming di dati dalla macchina locale, è stata caricata solo la metà di tutti i documenti all'interno del cluster, impiegando un tempo di esecuzione totale pari a undici ore. Sulla base di ciò viene stimato che il tempo di elaborazione totale per processare tutti i documenti del dataset, tramite streaming di dati, corrisponde, circa, a ventidue ore.

In contrapposizione a questo dispendio di tempo, presente nelle due azioni implementate, l'esecuzione delle query di ricerca richiede un lasso di tempo nettamente inferiore per la sua esecuzione, dell'ordine dei millisecondi. Questo risulta un aspetto molto positivo in tutte le applicazioni del caso in quanto è richiesto una maggior numero di interrogazioni, rispetto al numero di caricamenti dei dati all'interno del cluster. In aggiunta a ciò, tutte le interrogazioni non richiedono l'integrazione della pipeline per svolgere la propria funzione, dal momento che, esse vengono svolte su documenti prodotti dal processo di inferenza stesso.

Sulla base di tali considerazioni si può affermare che, complessivamente, l'integrazione delle tecniche di ML all'interno di Elasticsearch presenta tempi di esecuzione accettabili. Nello specifico, i processi di inferenza per i documenti richiedono molto tempo per la loro esecuzione, ma vengono eseguiti a intervalli di tempo molto lunghi e solo all'atto dell'inserimento di un nuovo documento. Per quanto riguarda, invece, l'esecuzione delle ricerche full-text, essa presenta un'alta efficienza nella generazione della risposta. Quest'ultima risulta una caratteristica vincente in quanto le query di ricerca vengono eseguite ripetutamente in tutte le applicazioni definite in precedenza.

6.2.3 Archiviazione richiesta

L'ultima metrica adottata per la validazione della pipeline riguarda la quantità di memoria utilizzata per salvare tutti gli indici Elasticsearch, inclusi quelli definiti a tempo di esecuzione. Come è possibile notare nella Tabella 6.1, i documenti ottenuti dall'esecuzione della Ingest Pipeline richiedono un'occupazione di memoria molto maggiore rispetto ai documenti originali. La motivazione dietro a questo aumento di spazio riguarda non solo l'archiviazione delle entità estratte ma anche il salvataggio del testo elaborato, contenente le Named Entity in rilievo, dentro un ulteriore attributo. Oltre alla presenza del campo aggiuntivo, contenente il corpo dell'articolo, sono presenti tutte le informazioni delle entità nominate, prodotte dal processo di inferenza.

Tipologia documenti	La Repubblica	Il Giornale	Totale (Mb)
Documenti originali	215,9	143.5	359.4
Documenti da reindex	969,9	663	1632,9
Documenti da stream_bulk (30.000 doc.)	215,9	149.7	365,6

Tabella 6.1: Occupazione di memoria dei diversi tipi di documenti, all'interno degli indici Elasticsearch

Tuttavia, queste informazioni aggiuntive contribuiscono solo ad una piccola parte della maggiore richiesta di archiviazione. Infatti, il contributo più rilevante risiede nell'aumento della dimensione dell'indice invertito per i nuovi indici Elasticsearch creati tramite reindexing. All'atto della creazione di questi ultimi, le proprietà definite dall'*analyzer* degli indici originali non vengono trasferite negli attributi corrispondenti. Questo comporta un'indicizzazione errata dei campi, contenenti il titolo e il testo degli articoli. Un modo per risolvere questa problematica consiste nella creazione degli indici di destinazione prima dell'esecuzione del reindex. Questa azione risulta necessaria quando viene implementato il processo di inferenza tramite streaming di dati. In merito a quest'ultima implementazione, sempre nella Tabella 6.1, per archiviare 30.000 documenti processati è richiesta una quantità di memoria di poco superiore a quella dei documenti originali.

Con i dati a disposizione si può affermare che se tutti i documenti elaborati vengono salvati all'interno di indici creati manualmente, essi richiedono un'archiviazione maggiore, ma in misura nettamente inferiore a quella richiesta dai documenti archiviati in indici creati automaticamente. Questo aumento generale dell'occupazione di storage non è, quindi, direttamente legato all'utilizzo di tecniche di ML e può essere risolto prima della fase di reindexing. Ad esempio, è possibile definire un template per l'indice di destinazione, il quale svolge il compito di selezionare e indirizzare i campi dell'indice sorgente, verso campi destinazione, definiti equivalentemente. Per scelta di implementazione, infatti, Elasticsearch, nella fase di reindexing, non "eredita" le tipologie dei campi, ma mette in atto, come ogni nuova indicizzazione, le euristiche di definizione dei tipi che vanno sotto il nome di *dynamic field mapping* [Elastic, 2022c].

In conclusione, l'archiviazione richiesta per il salvataggio dei documenti è l'aspetto da prendere maggiormente in considerazione. Difatti, se l'indice di destinazione non è costruito in modo non ottimale comporta non solo un'indicizzazione errata, ma anche un'inefficienza nell'utilizzo della memoria secondaria.

CAPITOLO 7

Discussione

Con la conclusione del capitolo in merito alla validazione, termina la parte relativa al processo di realizzazione della ricerca semantica e, con essa, anche la descrizione della realizzazione del framework implementato. In questo capitolo viene proposta una discussione critica in merito al lavoro svolto per questo elaborato. In particolare, verranno illustrate tutte le lezioni apprese (Lesson Learned) durante lo sviluppo del framework, sia per quanto riguarda gli aspetti positivi, sia per quelli negativi. Oltre alle Lesson Learned verranno esaminati alcuni sistemi correlati, con i quali è possibile realizzare una soluzione analoga a quella implementata, come sua alternativa.

7.1 Lezioni Apprese

Durante lo sviluppo del framework gli strumenti software utilizzati hanno mostrato comportamenti positivi, i quali hanno garantito il raggiungimento degli obbiettivi in maniera ottimale. Allo stesso modo, sono strati riscontrati aspetti negativi per i quali è stato necessario svolgere azioni correttive, oppure un'implementazione alternativa. Sia i punti di forza sia quelli di debolezza hanno contribuito alla raccolta delle Lezioni Apprese, le quali permettono di accrescere la conoscenza per questo campo di studio. In questa sezione vengono riportate tutte le Lezioni Apprese acquisite durante il processo di sviluppo del framework.

7.1.1 Risorse richieste da Elasticsearch

Partendo dallo strumento software utilizzato, un cluster Elasticsearch costruito tramite Macchine Virtuali, all'interno di una macchina locale, non presenta alcuna problematica per interrogazioni su piccole quantità di dati. Tuttavia, se viene utilizzata la stessa configurazione per interrogazioni su grandi quantità di dati e, sopratutto, se vengono utilizzate le componenti di Machine Learning (ML), si verifica un forte degrado delle prestazioni, rendendo il sistema inutilizzabile. Per soddisfare la maggiore richiesta di risorse da parte del cluster risulta necessario costruire quest'ultimo su una macchina più performante o, come visto per il caso in esame, è necessario utilizzare una Virtual Machine su cloud. Quest'ultima soluzione corrisponde a quella più vantaggiosa, in quanto l'azione di modifica delle risorse risulta semplice e con effetto immediato.

7.1.2 Capacità del framework

Per quanto riguarda il framework costruito, le librerie utilizzate permettono di svolgere gran parte delle azioni permesse da Elasticsearch attraverso chiamate API rivolte verso il

cluster, ovunque esso sia. Oltre a quelle per l'esecuzione delle query di ricerca, sono presenti chiamate API per la creazione di indici, per l'indicizzazione di una serie di documenti in un'unica chiamata, anche tramite streaming di dati, e per l'azione di reindex di documenti già inseriti. È, inoltre, possibile svolgere il deploy di modelli di ML esterni e definire Ingest Pipeline per implementare, o meno, questi ultimi. L'unica chiamata API, non ancora concessa dalle librerie, è quella che consente di abilitare e disattivare l'esecuzione dei modelli di Machine Learning, la quale è realizzabile solo con l'ausilio di Kibana. Fortunatamente, l'azione di abilitazione deve essere svolta una sola volta, successivamente al deploy del modello. Infatti, se l'esecuzione del cluster viene terminata con il modello caricato in memoria centrale, in tutti i riavvii successivi, insieme al cluster, viene avviato anche il modello di ML.

Durante lo sviluppo del framework è risultato necessario eliminare e ricreare l'intero cluster, a causa della scadenza della licenza di prova di Elasticsearch, la quale consente di utilizzare le componenti di Machine Learning. Questo ha comportato anche la perdita di tutti i dati definiti e caricati all'interno del cluster, portando quest'ultimo al suo stato iniziale di default. Il framework è risultato utile anche in questa situazione, in quanto, per mezzo delle chiamate API, è possibile riportare il cluster al momento prima della cancellazione, attraverso l'esecuzione degli script definiti. L'azione di ripristino risulta, quindi, di semplice realizzazione e con tempi di esecuzione largamente accettabili.

7.1.3 Funzionalità Machine Learning

Sebbene Elasticsearch permette l'importazione di qualsiasi modello di ML esterno, basato su modello BERT e affini, è possibile utilizzare solo una cerchia ristretta di essi. Tutti i modelli utilizzabili sono stati preallenati a partire da un dataset molto generico, contenente testo in lingua inglese. Utilizzare uno di questi modelli su un testo scritto in lingua diversa può portare ad un'estrazione di informazioni errate o imprecise. Nonostante questo limite caratteristico di Elasticsearch, i modelli di Named Entity Recognition sono stati in grado di estrarre la maggior parte delle entità, con errori di classificazione vari ed eventuali, comunque presenti.

7.1.4 Processi di elaborazione dei documenti

Le query di ricerca non solo vengono eseguite in maniera efficace ma forniscono una risposta efficiente a partire dai documenti ricercati; purtroppo, non si può dire la stessa cosa per tutti i processi di elaborazione che utilizzano la Ingest Pipeline per svolgere inferenze sui documenti. Ciò che è emerso durante lo sviluppo dimostra che i modelli di ML possono elaborare un solo documento alla volta, in quanto esso viene eseguito in un solo nodo Elasticsearch. Se viene implementata una elaborazione parallela della pipeline, solo alcuni documenti verranno correttamente processati dal modello, mentre gli altri saranno inseriti nell'indice contenente i documenti non elaborabili.

Sempre in merito a questa categoria di processi, risulta necessario prestare particolare attenzione all'indice in cui vengono salvati i documenti. Nel caso in cui non esiste un indice, oppure non viene indicato un *template* da utilizzare per l'indicizzazione dei documenti, Elasticsearch utilizza il *dynamic field mapping*, nel quale viene impiegato un *analyer* per la lingua inglese. Nel caso in cui vengono elaborati documenti in un'altra lingua, come nel caso in esame con l'italiano, nel processo di indicizzazione vengono considerate tutte le stopword della lingua, portando ad un'indicizzazione sbagliata dei documenti. Risulta, quindi, necessario costruire, a posteriori, un indice adatto ai documenti da elaborare o, in alternativa, definire un template in grado di indirizzare i campi dell'indice sorgente verso i campi di destinazione, con una configurazione adatta per il testo elaborato.

7.2 Sistemi correlati

Un altro aspetto da prendere in considerazione riguarda la possibilità di definire metodologie di realizzazione di framework in grado di integrare un linguaggio di programmazione con altri strumenti software simili ad Elasticsearch, disponibili sul mercato. In questa sezione vengono riportati alcuni software alternativi al motore di ricerca utilizzato, nei quali possono essere integrate componenti di Machine Learning per svolgere compiti analoghi a quelli realizzati.

7.2.1 Amazon OpenSearch

Il primo software alternativo al motore di ricerca nasce dall'azione di *fork* svolta su Elasticsearch stesso, come conseguenza del conflitto tra Elastic ed Amazon, già trattato nel Capitolo 1. *Amazon OpenSearch* nasce, quindi, dalla Versione 7.10 di Elasticsearch, sotto licenza Apache 2.0; il software include tutte le funzionalità di Elasticsearch, fino alla Versione soggetta alla fork, e si promette di essere più potente di quest'ultimo. Negli anni successivi alla fork, anche OpenSearch si è mosso verso l'integrazione delle funzionalità di Machine Learning per svolgere task di Natural Language Processing (NLP). Per svolgere task di Text Analysis tramite OpenSearch è richiesta l'integrazione di quest'ultimo con *Amazon Comprehend*. In questo modo, la prima componente si occupa del salvataggio dei dati, attraverso l'esecuzione di tutte le funzioni che riguardano l'indicizzazione e l'analisi di testo non strutturato. La seconda, invece, implementa un servizio di NLP in grado di svolgere Text Analysis tramite tecniche di Machine Learning. Nella Figura 7.1 viene mostrata l'architettura del cluster implementabile automaticamente, tramite il codice messo a disposizione su GitHub.

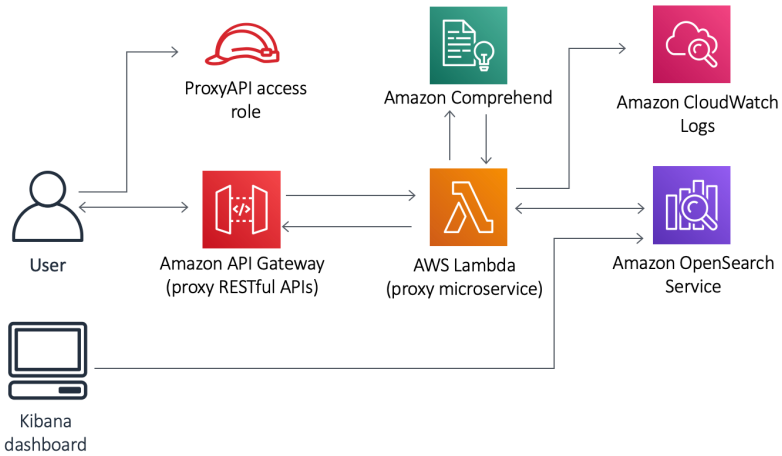

Figura 7.1: Architettura di Open Search implementabile con le impostazioni di default

Come si può notare, OpenSearch e Comprehend consentono la visualizzazione delle frasi, delle parole chiave oppure delle entità estratte dai documenti, in un task di Named Entity Recognition, per mezzo di una dashboard di Kibana pre-configurata dalle suddette componenti. Inoltre, l'accesso e le metriche per l'analisi dei dati vengono gestite da *Amazon CloudWatch Logs*. Nella stessa struttura è presente, anche, un microservizio di proxy, invocato da *Amazon API Gateway*, per la gestione della configurazione di preprocessing, degli indici nativi e delle altre capacità di ricerca native. Il microservizio in questione prende il nome di *AWS Lambda* ed è in grado di interagire con CloudWatch e OpenSearch per usufruire di tutte le funzioni messe a disposizione dalle due componenti. In questo modo, quando viene inviato del testo tramite chiamata API, il microservizio di proxy invia la richiesta a

Comprehend per analizzare il testo. Successivamente si prosegue con l'indicizzazione del testo tramite una chiamata a OpenSearch, insieme alla pubblicazione dei log e delle metriche, la quale viene svolta da CloudWatch. Infine, i dati indicizzati possono essere visualizzati per mezzo della dashboard di Kibana, messa a disposizione dal cluster.

7.2.2 Solr

Se OpenSearch è una soluzione che nasce a partire da Elasticsearch, *Solr* (Figura 7.2) nasce prima della creazione del motore di ricerca, e rappresenta, in un certo senso, la ragione per cui esiste Elasticsearch, in quanto quest'ultimo si riprometteva di coprire le lacune nella ricerca che erano presenti in Solr. Tuttavia questo non ha portato alla completa scomparsa di Solr; al contrario, Solr sta tutt'ora crescendo come un progetto a se stante, tenendo testa ad Elasticsearch su diversi fronti. Solr presenta molteplici similarità con la sua controparte. Entrambi sono costruiti sopra Apache Lucene ed entrambi permettono la frammentazione dei documenti (o sharding), i quali vengono costruiti con un mapping "schemaless" per consentire il riconoscimento automatico degli schemi di un dataset. Altre similarità riguardano la ricerca full-text, l'indicizzazione in tempo reale, il clustering dinamico, etc.

Figura 7.2: Logo di Solr

Come è accaduto per Elasticsearch e la sua controparte OpenSearch, descritta precedentemente, negli ultimi tempi anche Solr si è rivolta sempre più verso l'applicazione di task per NLP, ad esempio attraverso l'integrazione con *Apache OpenNLP*. Quest'ultimo è definito come una libreria basata su Machine Learning, costruita appositamente per svolgere task di NLP in Java. La libreria permette l'esecuzione di funzioni comuni dell'NLP, quali tokenizzazione, Language Detection, Part-Of-Speech Tagging, Lemmatizzazione e, soprattutto, Named Entity Recognition. OpenNLP permette di allenare i modelli per NLP per poi usarli in una qualunque applicazione Java, attraverso le librerie messe a disposizione. Applicando l'integrazione con Solr per mezzo di Apache Lucene, è possibile utilizzare le capacità di OpenNLP per analizzare i documenti durante il processo di indicizzazione.

Non è possibile applicare modelli Transformer, come invece consente di fare Elasticsearch, in quanto non è stata definita alcuna metodologia per impiegare questi modelli di ultima generazione utilizzando solamente OpenNLP. Ciò ha portato ad una carenza di strumentazione di NLP dentro l'ecosistema Java, dal momento che tutti i processi moderni di Natural Language Processing vengono realizzati in Python. Una soluzione a questa problematica consiste nella configurazione di un servizio remoto per svolgere processi di inferenza, attraverso modelli allo stato dell'arte, mediante chiamate API. L'obbiettivo che ci si potrebbe porre consiste nell'utilizzare i modelli Transformer dall'applicazione Java, rendendo il processo più performante e facile da manutenere. Questa soluzione può essere realizzata tramite *ONNX*, il quale fornisce uno standard per modelli di ML e, allo tesso tempo, fornisce runtime per diversi linguaggi di programmazione, come Java. Per mezzo di ONNX Runtime è, quindi, possibile utilizzare modelli Transformer allenati nell'ecosistema Python, a partire da OpenNLP.

7.2.3 ArangoDB

ArangoDB viene considerato, a tutti gli effetti, un ottimo strumento alternativo ad Elasticsearch per lo sviluppo di un framework. Definito sotto la licenza open source Apache V2, esso si comporta come un motore di ricerca simile ad Elasticsearch, ma, allo stesso tempo, è in grado di elaborare dati definiti sotto modello chiave-valore e a grafo, oltre a quelli definiti come documenti. L'aspetto positivo di ArangoDB riguarda la possibilità di elaborare i vari formati, attraverso un unico linguaggio di query, che prende il nome di *ArangoDB Query Language*, abbreviato in AQL. Questo strumento trova utilizzo, in modo particolare, quando vengono utilizzati Elasticsearch e altri database, come *MongoDB*, ed è necessario semplificare le operazioni, attraverso l'utilizzo di un'unica soluzione.

Figura 7.3: Logo di ArangoDB

Negli ultimi anni anche ArangoDB si è mosso sempre di più verso l'implementazione di ricerche full-text. In questo è stato favorito dal linguaggio di query del software, caratterizzato da una semplicità di apprendimento anche per la realizzazione di query complesse. In concomitanza alla ricerca full-text, viene implementata anche la possibilità di svolgere task di Natural Language Processing, come il Word Embedding, per la rappresentazione numerica del testo in vettori. La motivazione dietro a questo tipo di rappresentazione è basata sull'assunzione secondo cui le parole che compaiono in contesti simili e condividono significato analogo dovrebbero avere embedding con struttura simile tra loro.

Per quanto riguarda, invece, l'implementazione di un task per l'estrazione di entità, simile al Named Entity Recognition, il software dà la possibilità di svolgere task di *Entity Resolution*. Questa categoria di processi consente di svolgere una disambiguazione di record di entità, rappresentate più volte in uno o più database, in maniera simile alla *Data Deduplication*. Difatti, quest'ultima si occupa di raggruppare tutti i dati provenienti da sorgenti diverse in una singola entità del mondo reale. Il task di Entity Resolution trova spazio in campi applicativi che riguardano la Fraud Detection, la raccolta di informazioni sull'identità degli individui per le procedure di *Know Your Customer* (KYC) e *Customer 360*, etc. Quest'ultima tipologia di applicazioni fornisce una rappresentazione dei dati di un cliente a tuttotondo, includendo tutte le interazioni che riguardano, ad esempio, una richiesta svolta su un sito Web, l'invio di un ticket per assistenza o l'acquisto di un prodotto.

A differenza della soluzione software definita in precedenza, incentrata sull'implementazione di modelli di ML su linguaggio Java, ArangoDB permette di realizzare tutti i task di NLP tramite linguaggio Python. Attraverso l'utilizzo di apposite librerie (principalmente, arango e pyArango), è possibile realizzare framework per interfacciare Python con ArangoDB, in maniera molto simile a quanto svolto con Elasticsearch.

CAPITOLO 8

Conclusioni

In questo elaborato è stato trattato il processo di definizione di un framework per realizzare un'integrazione di Elasticsearch e Python, con l'obbiettivo di svolgere task di Natural Language Processing. Per la sua realizzazione è stato scelto un dataset di riferimento contenente testo non strutturato in linguaggio naturale. Successivamente, sono state estratte tutte le informazioni all'interno del dataset e queste sono state inserite all'interno di una struttura dati con formato compatibile con il motore di ricerca. I documenti ottenuti sono stati caricati all'interno di appositi indici, creati precedentemente tramite una struttura ad hoc. Quest'ultima è stata definita per consentire un'indicizzazione corretta del testo, in cui la tabella dell'indice invertito non presenta alcuna stopword della lingua italiana, con cui il testo è scritto. Una volta terminato il processo di indicizzazione, sono state svolte diverse tipologie di ricerca sintattica, consentite dalle funzioni standard del motore di ricerca.

La seconda parte dell'elaborato ha trattato il processo di integrazione delle funzionalità di Machine Learning, all'interno di Elasticsearch, per realizzare una ricerca di tipo semantica. Una volta attivata la licenza Elasticsearch, attraverso un cambio di infrastruttura per l'esecuzione del cluster, è stato selezionato un modello che realizza un task di Named Entity Recognition, il quale è compatibile con il motore di ricerca. Una volta integrato il modello, all'interno di una Ingest Pipeline, quest'ultima è stata applicata su tutti i documenti precedenti, ottenendo una loro copia con informazioni in merito al contesto descritto dal testo. Infine, sono stati definiti quattro casi d'uso che comprendevano l'utilizzo degli ultimi documenti ottenuti, dimostrando il campo di applicazione di Elasticsearch insieme alle componenti di Machine Learning integrate.

Un possibile sviluppo futuro, a partire dal framework realizzato, riguarda l'integrazione dell'analisi dei modelli, la quale viene svolta attraverso un modello di embedding, in grado di valutare il caso d'uso della ricerca semantica. Un altro sviluppo futuro implementabile riguarda il tracciamento e la valutazione delle evoluzioni dei modelli allenati; dal momento che i modelli eseguibili su Elasticsearch vengono scelti a priori dagli sviluppatori, risulta necessario valutare se il suddetto limite viene superato o meno nelle versioni successive del software. Un ultimo sviluppo implementabile riguarda l'accoppiamento di Elasticsearch con una pipeline Dataflow. In questo modo verrà fornito il massimo supporto per i modelli di Machine Learning, non necessariamente preallenati, mentre Elasticsearch verrà utilizzato per la ricerca sintattica, aggiunta alla capacità di svolgere ingestion dei tag, generati dal modello utilizzato. Queste componenti innovative potranno essere utilizzate per costruire una soluzione sperimentale per poi definire un processo di prodottizzazione di quest'ultima.

Bibliografia

AMAZON (2022), «Text Analysis with Amazon OpenSearch Service and Amazon Comprehend», .

CHRISTOPHER MARSHALL (2019), «What is named entity recognition (NER) and how can I use it?», super.AI. (Citato a pagina 27)

ELASTIC (2022a), «Built-in Analyzer», Elastic. (Citato a pagina 22)

ELASTIC (2022b), «Compatible third party NLP models», . (Citato a pagina 30)

ELASTIC (2022c), «Dynamic Field Mapping», . (Citato a pagina 51)

ELASTIC (2022d), «Extract Information», Elastic. (Citato a pagina 27)

ELASTIC (2022e), «Ingest Pipelines», . (Citato a pagina 35)

ELASTIC (2022f), «Language Analyzers», Elastic. (Citato a pagina 21)

ELASTIC (2022g), «Natural Language Processing», Elastic. (Citato a pagina 26)

ELASTIC (2022h), «What's new in Elasticsearch 8.3», . (Citato a pagina 5)

GABRIELE SARTI (2020), «Dataset CHANGE-IT», . (Citato a pagina 19)

GORMLEY, C. e TONG, Z. (2015), *Elasticsearch: The Definitive Guide*, O'Reilly Media, Inc.

HUGGINGFACE (2022a), «Fill-Mask», . (Citato a pagina 28)

HUGGINGFACE (2022b), «Question Answering», . (Citato a pagina 28)

IBM (2022), «Cos'è un API RESTful», . (Citato a pagina 9)

JACOB DEVLIN, MING-WEI CHANG, KENTON LEE, KRISTINA TOUTANOVA (2018), «BERT: Pre-training of Deep Bidirectional Transformers for Language Understanding», . (Citato a pagina 26)

JEFF ZEMERICK (2022), «Using Modern NLP Models from Apache OpenNLP with Solr», .

NEELAM TYAGIMAYI (2021), «What is Text Mining? Process, Methods and Applications», . (Citato a pagina 15)

ONTOTEXT (2022), «What is Information Extraction?», . (Citato a pagina 27)

RAJESH KUMAR (2018), «Understanding REST HTTP method – GET, POST, PUT, HEAD, DELETE in Elasticsearch», . (Citato a pagina 9)

SARKAR, D. (2015), *Text Analytics with Python*, Apress.

STEVEN BIRD, E. L., EWAN KLEIN (2009), *Natural Language Processing with Python: Analyzing Text with the Natural Language Toolkit*, O'Reilly Media, Inc.

TIBCO (2020), «Cos'è il text mining?», . (Citato a pagina 15)

TIBCO (2022), «Cos'è un API», . (Citato a pagina 9)

TWAIN TAYLOR (2021), «Top 8 Elasticsearch alternatives after its license change», .

UZMA RAJA, TARA MITCHELL, TIMOTHY DAY, J. MICHAEL HARDIN (2008), «Text mining in healthcare. Applications and opportunities», . (Citato a pagina 18)

Siti web consultati

- Amazon Web Services – `https://aws.amazon.com/`
- AnalyticSteps – `https://www.analyticssteps.com`
- Arxiv Cornell University – `https://arxiv.org`
- Hugging Face – `https://huggingface.co`
- DevOpsSchool – `https://www.devopsschool.com`
- Elastic – `https://www.elastic.co/`
- HuggingFace – `https://huggingface.co`
- IBM – `https://www.ibm.com`
- Medium – `https://medium.com`
- Ontotext – `https://www.ontotext.com`
- OpenSource Connections – `https://opensourceconnections.com`
- PubMed – `https://pubmed.ncbi.nlm.nih.gov`
- TechGenix – `https://techgenix.com`
- Tibco – `https://www.tibco.com`
- Wikipedia – `www.wikipedia.org`
- Wikipedia-BERT – `https://en.wikipedia.org/wiki/BERT_(language_model)`
- Wikipedia-Elasticsearch – `https://en.wikipedia.org/wiki/Elasticsearch`
- Wikipedia-Named Entity – `https://en.wikipedia.org/wiki/Named_entity`
- Wikipedia-Named Entity Recognition – `https://en.wikipedia.org/wiki/Named-entity_recognition`

Ringraziamenti

Il primo ringraziamento va al Prof. Domenico Ursino, che mi ha guidato in questo lavoro di tesi, fornendomi la sua più completa disponibilità e costanza. Ringrazio nuovamente il Prof. Ursino per avermi aperto le porte a questo ramo molto interessante dell'Informatica, offrendomi la possibilità di approfondirlo, anche attraverso l'esperienza di tirocinio svolta con Injenia.

Ringrazio tutti i membri del Team di Machine Learning di Injenia S.R.L., in particolare Simone e Riccardo, che mi hanno seguito nel lavoro svolto in questo elaborato. Ringrazio Giulio per aver condiviso insieme questo percorso, all'interno del Team, e per accompagnarmi nel prossimo percorso che ci aspetta con Injenia.

Ringrazio, inoltre, i miei compagni di studio: Edoardo, Lorenzo, Andrian, Denis e Alex per tutto il supporto che mi hanno dato in questi anni. Ringrazio, in particolare modo, Fiorenza che, più di chiunque altro, mi ha accompagnato in questo percorso, superando i vari ostacoli che si paravano dinnanzi e condividendo i momenti di soddisfazione che venivano di conseguenza. Grazie, sopratutto, per avermi sostenuto in questi mesi di duro lavoro, con la tua completa disponibilità e tutto il sostegno che mi hai fornito per far si che arrivassi a raggiungere il mio obbiettivo.

Ultima, ma di certo non meno importante, ringrazio tutta la mia famiglia, primi fra tutti i miei genitori, per avermi dato la possibilità di iniziare e concludere anche questo percorso universitario, possibilità per nulla scontata per il momento storico odierno. Grazie per il sostegno costante nel lavoro svolto e in quello futuro, e per avermi insegnato a dare sempre il meglio di me e ad avere maggiore fiducia nelle mie capacità. Ringrazio, inoltre, mia sorella che, nonostante i chilometri di distanza, anche dall'altra parte del mondo, mi ha insegnato a dare la giusta importanza alla cosa di più valore in assoluto: il tempo, e dare il giusto tempo alle cose che contano di più in assoluto.